Thesis proposed to achieve the degree of master in computer science/databases

Privacy in Databases

Mathy Vanhoef

Academic years:	2010—2011 & 2011—2012
Promoter:	Prof. dr. Jan Van den Bussche

Abstract

In the current digital age more and more data is being collected, yet it's unclear how to assure adequate privacy. The question of how to analyze large amounts of data while preserving privacy now prevails more than ever. In the course of history there have been many failed attempts, showing that reasoning about privacy is fraught with pitfalls. This caused an increased interest in a mathematically robust definition of privacy.

We will prove that absolute disclosure prevention is impossible. In other words, a person that gains access to a database can always breach the privacy of an individual. This motivated the move to assuring relative disclosure prevention. One of the most promising definitions in this area is differential privacy. It addresses all the currently known attacks, has plenty of practical implementations, and knows many extensions that make it applicable in a wide range of situations.

Networked data also poses challenging privacy issues. Both active and passive attacks, where the underlying structure of the network is used to de-anonymize individuals, are discussed in detail. Degree anonymization and algorithms to create a degree anonymous graphs will also be given. Although still an active research topic, privacy of networked data is considered increasingly important given the rise of social media.

Preface

This thesis started as a custom proposal to study security, and in particular privacy, from a theoretic point of view. What started by reading a research article on the foundation of private data analysis quickly turned into a broad subject covering many aspects of privacy in databases.

My deepest gratitude goes to my promoter, Prof. dr. Jan Van den Bussche, for making time in his already full schedule to help me successfully complete this work. During our rare but intense meetings he provided invaluable feedback and insights. Often I was critical at first, but once he completed the explanation of his interpretation, my understanding of the matter increased significantly and broadened my point of view. Even after a meeting he would continue to spend time on finding a more sound proof, and once found I would receive an e-mail late at night explaining his findings. I'm very thankful for his indispensable help.

Finally, I would like to thank my parents and brothers, whose support during the more stressful moments has been invaluable.

Contents

Part I

Foundations of Privacy

Chapter 1

Introduction

Before delving into the details we will give a short introduction on privacy. This is especially needed because privacy is a very broad notion and can be tackled from multiple angles. In this chapter section *What is Privacy?* and *What About Cryptography?* is based on the first lecture of Naor on *Foundations of Privacy* [Nao10a]. *Tensions in Releasing Data* is based on the PhD thesis of Sweeney [Swe01]. The remaining introductory content was gathered from different sources when reading about—and working on—the subject.

1.1 What is Privacy?

Before we begin it's necessary to define what we mean by privacy. Unfortunately this is an extremely overloaded term and is hard to define precisely. The following quotes demonstrate this point:

> "Privacy is a value so complex, so entangled in competing and contradictory dimensions, so engorged with various and distinct meanings, that I sometimes despair whether it can be usefully addressed at all." — Robert C. Post [Pos01]

> "Privacy is the right to be let alone."
> — Samuel D. Warren and Louis D. Brandeis [WB85]

> "Privacy is our concern over our accessibility to others: the extent to which we are known to others, the extent to which others have physical access to us, and the extent to which we are the subject of others." — Ruth Gavison [Gav80]

> "There is no one that has a good definition of what privacy actually is."
> — Willem Debeuckelaere (translation mine) [pan11]

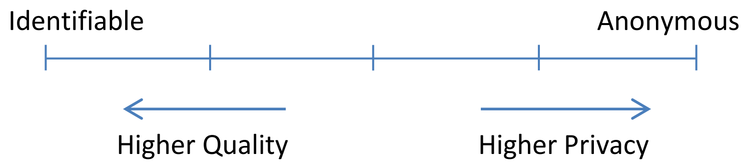


Figure 1.1: Tension between quality and privacy

> “Privacy is like oxygen. We really appreciate it only when it is gone.” — Charles J. Sykes

Especially in the current information age, privacy seems to be a fragile concept. Huge amounts of data are being collected, people publish their information on social networks, public surveillance information is being collected (e.g., security cameras), telephone companies are required to store certain information, internet service providers are storing sensitive information, and so on.

In this work we will investigate if there are mathematical definitions of privacy that make sense and can be achieved by algorithms. The advantage of this theoretical approach is that we can prove that certain methods will achieve a specific form of privacy. Today, the most promising definition is *differential privacy*. But before we explain that further it’s important to know the background and the history of the research being done in privacy, as this will give you a better insight of the surrounding problems one must take into account.

However, not every problem related to privacy is currently solved, and there are still many open questions and problems. Whilst differential privacy and its variants certainly have promise, the definition is not always easily applicable and is in some cases too strong.

1.2 Tensions in Releasing Data

When releasing data there is always a tension between quality and anonymity. On the one hand, releasing the full dataset provides the best quality, while the other extreme of not releasing any data provides the best privacy [Swe01].

Consider figure 1.1: At one end we have the data that is fully identified, and on the other end is the anonymous data derived from the original data. When we want to provide any form of privacy, no matter how minimal, the original data must be modified to ensure anonymity, and is thus distorted. So when we move away from the fully identified end of the spectrum, towards the anonymous end, the resulting data will be less useful. In particular, this means that there are some tasks that could be computed on the original

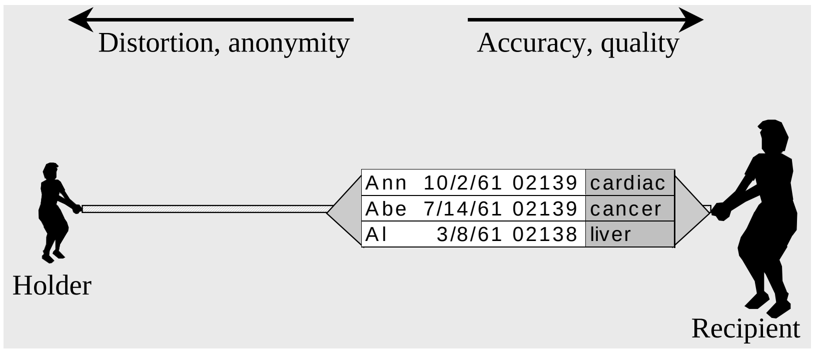


Figure 1.2: Recipient's needs overpower privacy concerns [Swe01]

data, but not on the released data (or the error margin of the result would be too high to be considered useful).

From this we realize that the desired anonymity and usefulness depends on the sensitivity of the data itself and depends on the task to which the analyst puts the data. That is, given a particular task, there exists a point on the continuum in Figure 1.1 where the released data is as anonymous as possible, yet the data still remains useful for the task. This can be characterized as a tug-of-war between the people and analysts: The people want maximum anonymity while the analyst wants the most useful data.

1.2.1 Preference for Accuracy

There are several examples where access to accurate information is more important than anonymity. In medical sectors this is especially true: To provide the best care for patients the original, fully identified health information, must be available to doctors. This is illustrated in figure 1.2.

However, note that it's possible the information may not be released to *everyone*. In our example we expect that doctors follow the Hippocratic Oath and keep the information of patients private. Nevertheless, if we consider the doctor to be the analyst, it still holds that the original information is given and no modifications were made so ensure any form of privacy.

1.2.2 Preference for Anonymity

The opposite extreme is also a realistic scenario. In this case the need to preserve the confidentiality of the information overshadows the possible utility it would have for an analyst. Information related to national security or military agencies are a good example. Here privacy and confidentiality is achieved by not releasing the information at all.

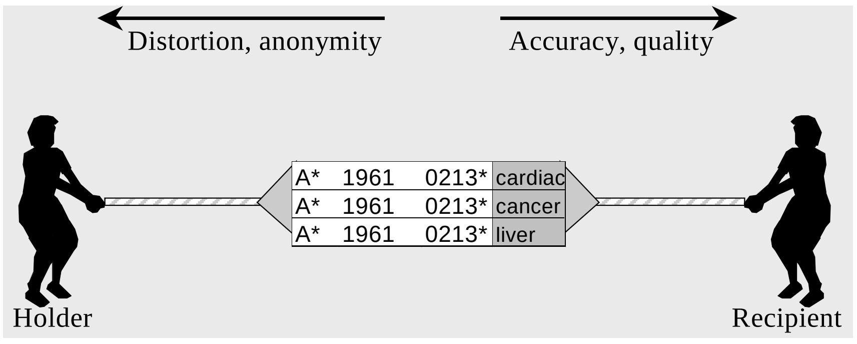


Figure 1.3: Balance between privacy concerns and uses of the data [Swe01]

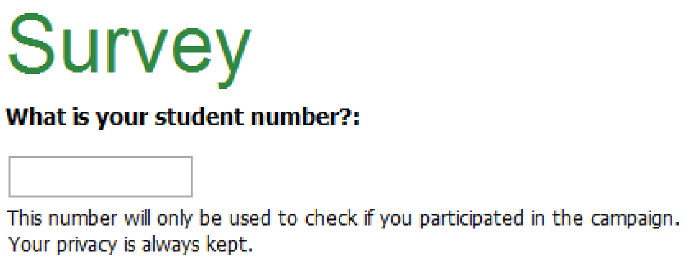
1.2.3 Balance between Accuracy and Anonymity

Finally there is the common situation where we must achieve a reasonable balance between accuracy and anonymity. This is also were the remainder of the work will focus on: Problems were we don't want to release the original data, but still want to calculate and release global statistics and properties of the data. See figure 1.3 for a graphical representation.

Take the case of medical research where, as an example, researchers want to search for patterns in diseases, genotype-phenotype correlations, and so on. At the moment the analyst either receives the original patient data or no data at all [Swe01]. This is because there are currently no guarantees that removing certain information, or perturbing it in a certain way, provides enough privacy for the patient. In this thesis we will seek for methods—that can be applied in situations like these—that provably guarantee certain definitions of privacy. If such methods can be found it's reasonable to assume that the anonymized data will be easier accessible by researcher.

1.3 Information Age

Currently we are in a unique period where for the first time it's possible to cheaply store huge amounts of data. Before the introduction of the computer, gathering information was a slow process. Analyzing this data was an even more painstaking activity. Because of these limitations the issue of privacy was easier to tackle: Removing personal identifying attributes (e.g., name, address, etc.) was enough to provide a reasonable amount of privacy. These days however this is not enough. As we will see, a combination of so called *quasi-identifiers* allow one to uniquely identify an individual. For example, in an attack where medical data and voter data were combined, researchers were able to re-identify the medical records of the governor of Massachusetts [Swe02]. This was possible because with only the combina-



Survey

What is your student number?:

This number will only be used to check if you participated in the campaign.
Your privacy is always kept.

Figure 1.4: Will your privacy really be kept?

tion of Zip, Birth date and Gender they were able to unique identify a large part of the population.

Social Pressure

Another interesting aspect is that society pressures you to make certain choices that decrease your anonymity. As pointed out by Marlinspike in his presentations at Blackhat [Mar10], the choice of owning a cell phone for example is not necessary a real choice. At first sight the choice may be whether or not to own a simple piece of technology. But as people begin to use the technology more and more, they become dependent on it, and the society around it changes. People used to make more plans while talking to someone in real life, but nowadays this is decreasing and they rely on their cell phone to arrange meetings.

So in our example it's no longer just the choice of owning a cell phone, but the choice of being part of society or not. This is a much bigger choice than just owning a gadget and possibly a choice that lowers your privacy and anonymity.

1.4 Guarantees of Privacy

When filling in surveys or sharing your information online, you often see the phrase "your privacy is always kept" or similar (see image 1.4). One has to ask himself what this means exactly. Since there are currently no well-known methods to actually ensure privacy, this sentence doesn't have much meaning. All is says it that the organization or person(s) in question will do his best to provide some privacy. But since no guarantees are given, the users is—most of the time—left in the dark about what is actually done to provide anonymity.

As an example that organizations can fail to protect your privacy we will explain the AOL search fiasco. In chapter 3 we present more cases where the privacy of individuals was not adequately protected.

The AOL Search Fiasco

In 2006 AOL released 20 million search queries from 65,0000 AOL users. The data includes all searches from those users for a three month period, as well as whether they clicked on a result, what that result was and where it appeared on the result page [Arr06]. In an attempt to protect anonymity, the username of each individual has been changed to a random ID. However, different searches made by the same person still have the same random ID.

It didn't take long before the first person, no. 4417749, was identified as Thelma Arnold [BZ06]. She conducted hundreds of searches ranging from "numb fingers" to "60 single man" to "dog that urinates on everything". Based on her search terms it was straightforward to deduce her age, where she lived, her last name, and so on.

1.5 Statistical Databases

The issue of privacy was first investigated with respect to statistical databases. In this context it is called *statistical disclosure control*: Revealing accurate statistics about a set of respondents while preserving the privacy of individuals. It has a notable history with extensive literature spanning statistics, theoretical computer science, security, databases and cryptography [Dwo11]. For an overview of this long history see the survey of Adam and Wortmann [AW89]. This extensive history is a testament to the importance of privacy.

Certain results extend beyond the realm of statistical databases and can be used in more general settings. Nevertheless, since the focus is on statistical data, we will provide a few examples why these databases can be of enormous social value.

1.5.1 Genotypes and Phenotypes (dbGaP)

In an effort to ease complex analyses of genetic associations with phenotypic and disease characteristics, the National Center for Biotechnology Information has created a database to store—among other things—individual-level information on phenotype, genotype, sequence data and the associations between them [MFJ$^+$07]. Access to individual data is only allowed to authorized researchers, and there are strict guidelines on how to store and use this data. Although the high-density genomic data is de-identified, it still remains unique to the individual and could potentially be linked to a specific person if used in conjunction with other databases (this is what we will call a *composition attack* or *linkage attack*).

It's clear that access to this kind of data can tremendously help researchers in their work. Unfortunately they must first get permission in order to obtain the data, and then follow strict guidelines. We believe that new privacy mechanisms can ease access to this data, while maintaining a provable form of privacy.

1.5.2 Internet Usage

Another huge source of information is internet usage. Your search queries are saved, the websites you visit can be tracked, the items you bought on website such as amazon can be recorded, Internet Service Providers keep logs of your overall internet usage, you provide information yourself to social networks, and so on. This last item, namely that of social networks, is especially interesting. Here people willingly publish all their information online. One has to ask himself if everyone is actually aware of the huge privacy concerns this raises. The founder of WikiLeaks, Julian Assange, even critiqued facebook—a popular social network website—as “an appalling spy machine”. His reasoning behind this claim is that all the information saved by facebook is accessible to U.S. intelligence by using legal and political pressure [AE11].

It’s clear that overall this is a huge amount of data that can potentially be abused. But on the other hand it can also be invaluable information to improve services. Search engines can optimize your search results based on previous queries, e-commerce website can suggest similar products, and so on.

Unfortunately it’s unclear how well the privacy of the users is protected in these scenarios. Ideally it should be possible to analyze this information real-time, without the need to store it afterwards, all while preserving privacy. This is not an easy problem to solve, especially if the algorithms to analyze the data are complex. Nevertheless, we believe that advances in the area of privacy under continual observation[1] will one day offer an acceptable and usable solution.

1.6 Models for Information Flow

1.6.1 Interactive vs. Non-Interactive

There are two natural modes when releasing data. In the first one—called the non-interactive mode—a modified version of the database is published and everyone can access this database. This public database is called the sanitized database, and the process of creating it is depicted in figure 1.5. Historically this is a commonly used method, where one simply removed identifying attributes such as name, address, social security number, and so on. But now more and more data is being stored, and because of the increase in processing power, these older methods no longer provide the

[1]This is a concept that extends the notion of differential privacy to guarantee confidentiality while analyzing data in *realtime*.

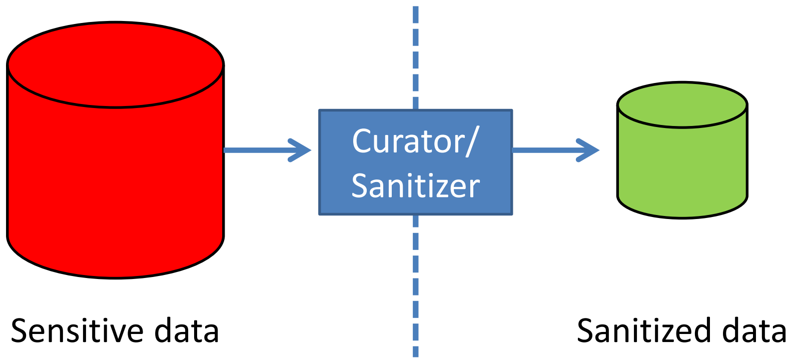


Figure 1.5: Non-interactive view: A sanitized version of the database is published.

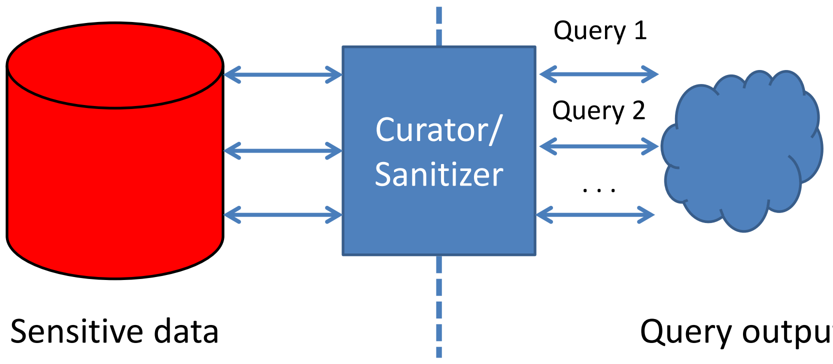


Figure 1.6: Interactive view: The sanitizer answers specific queries.

privacy one would expect (Chapter 3 gives examples of why such a strategy doesn't guarantee privacy). An example of a non-interactive sanitizer is k-anonymity which is explained in section 3.3. An advantage of the non-interactive mode is that once the sanitized database has been created, the original data can be destroyed.

The second possibility is called the interactive mode. Here no data is published, instead users can submit queries to the curator. The curator will then answers these questions in such a way so the privacy of individuals in the database is preserved (see figure 1.6). Privacy can be preserved by adding noise to the output, allowing only a limited number of queries, rejecting certain types of queries, and so on. An example of an interactive mechanism is ϵ-differential privacy which is explained in chapter 5. In both cases we assume that the curator is a trusted entity.

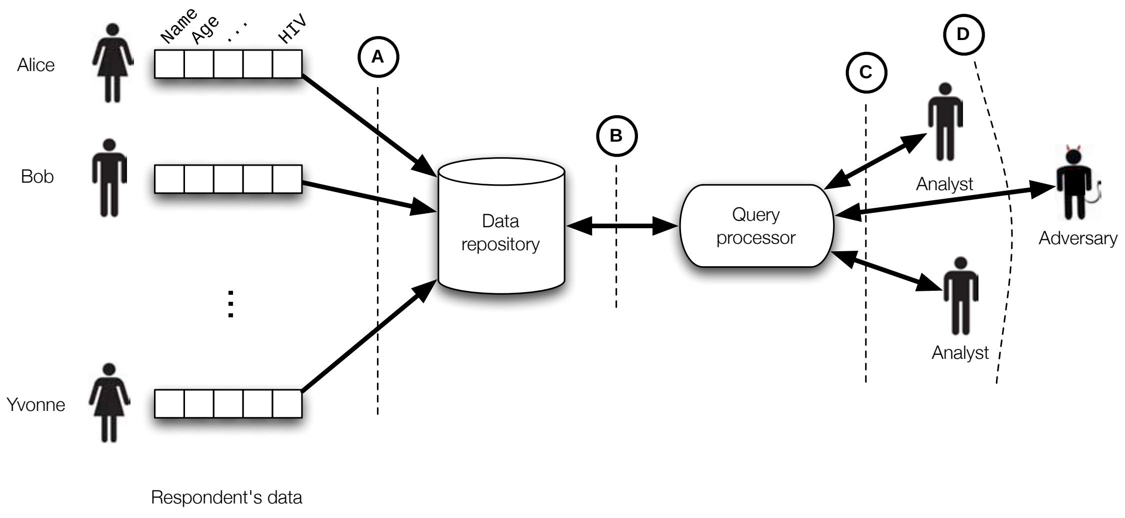


Figure 1.7: Model of information flow: Individuals provide their private information on which analyst want to derive aggregate statistics from. [Hay10]

1.6.2 Information Flow

An interactive or non-interactive mechanism that provides privacy can be achieved in several ways. Figure 1.7 gives a model of the information flow starting at the individual and ending at the analyst. Here individuals share their private information and these are collected in a database that is controlled by the data curator. Analysts can perform calculations on it to learn aggregate statistics, while an adversary shouldn't be able to learn sensitive information about specific individuals.

To protect privacy we must control the flow of information somewhere along its path. As figure 1.7 suggest this can be done in several ways:

A. Input Perturbation Users can add noise to the data they hand over. An adversary is therefore never sure whether the sensitive information about a specific individual is correct, while aggregate statistics can still be calculated.

B. Transformed Data Release Once the data of all individuals is collected and saved in a database, one can attempt to transform this data such that sensitive information about individuals can no longer be derived from it. A well-known example is *k*-anonymity.

C1. Query Auditing The data curator can monitor the queries being performed on the data. If the answer to a query might contain sensitive information on an individual, the query is refused and not executed.

C2. Query Answer Perturbation The analyst can query the data, but noise is being added to the answers in order to guarantee privacy. A well-known example is differential privacy.

D. Access Control The dataset can be given to trusted researchers and never released publicly. This should prevent malicious individuals from obtaining the data.

1.7 What About Cryptography?

Strangely the application of cryptography is sometimes suggested as a potential solution. Although cryptography is relevant, it does not solve the privacy problem. Cryptography can be used to encrypt, and thus hide, information from an adversary. But in the end the recipient still receives the original, unmodified information. So even if encryption is used the recipient will obtain the original information including all individual-specific data, and the privacy of these individuals will still be violated.

To further illustrate this point, we will briefly cover Secure Function Evaluation, which is sometimes offered as a solution to the privacy issue. It's a form of secure multi-party computation where entities want to distributively compute a function over their inputs, without exposing their private input. This is a well-studied problem in the field of cryptography. At first sight this may seem to provide privacy and anonymity, unfortunately this is not the case.

1.7.1 Secure Function Evaluation

Secure Function Evaluation (SFE) deals with the problem of how to distributively compute a function $f(X_1, X_2, \ldots, X_n)$ where:

1. Each X_i is only known to party i.
2. The parties should only learn the final output denoted by ξ. In other words the private input of each party won't be exposed to other parties.

Requirement 2 holds even if there are parties that maliciously attempt to obtain more information. Practical implementations that achieve this definition depend on, or are restricted by, the number of parties, the power of the adversary, the communication channel, ... The problem lies in the fact that the result ξ might itself disclose information. To be precise, the definition of SFE states that nothing will be revealed about any input X_i other than we can be reasonably deduced by knowing the output ξ.

So if there are two parties a and b, and the function f would be $sum(a, b)$, the result ξ would yield the input of the other party. This is not acceptable if we want to guarantee privacy.

Chapter 2

Probability Theory

In this chapter we will give a brief overview of the basics of probability theory. This is done both to refresh the most important concepts, and to introduce the notations and terminology that will be used throughout the remainder of this thesis. To avoid any ambiguity we will start by stating the exact definitions. Then we will cover a few properties that will be used in forthcoming proofs. The mathematical definitions given in this chapter are based on those in the book of Grimmett and Welsch [GW86].

2.1 Introduction

Probability theory deals with the study of experiments. An experiment (or trail) is defined as *a course of action whose consequence is not predetermined* and is denoted by $\mathcal{E}$. An experiment can be defined mathematically using a probability space.

Definition 2.1. *A probability space is a tuple* $(\Omega, \Sigma, \Pr)$ *where:*

- Ω is called the sample space and is defined as the set of all possible outcomes of the experiment $\mathcal{E}$.
- $\Sigma \subseteq 2^\Omega$ is the event space and must be a *sigma-algebra* on Ω. The notation 2^Ω represents the power set of Ω. The conditions a sigma-algebra must meet are given in definition 2.2.
- $\Pr$ is a *probability measure* on the event space Σ. Its requirements are given in definition 2.3.

The sample space can be seen as a collection of all elementary results, or outcomes, of an experiment. An event $E \in \Sigma$ is then a set of outcomes, and thus a subset of the sample space. The event space is the set of all outcomes of $\mathcal{E}$ which can be considered interesting. If Ω is a discrete sample space, then usually Σ is equal to 2^Ω. The definition of a sigma-algebra is as follows

Definition 2.2 (Sigma-algebra)**.** *Let Ω be any set and $\Sigma \subseteq 2^\Omega$. We call Σ a sigma-algebra over Ω if*

1. It's not empty: $\Sigma \neq \varnothing$
2. It's closed under complementation: $\forall A \in \Sigma : \Omega - A \in \Sigma$
3. It's closed under countable unions: If $A_1, A_2, \ldots$ is a countable sequence of events in Σ, then $\bigcup_{i\geq 1} A_i \in \Sigma$

Lemma 2.1. *If Σ is a sigma-algebra over Ω it it follows from definition 2.2 that $\Omega \in \Sigma$.*

Proof. From $\Sigma \neq \varnothing$ it follows that there exists an event $E \in \Sigma$. Because Σ is closed under complementation we also have $\Omega - E \in \Sigma$. As it's also closed under countable unions we finally get $(\Omega - E) \cup E \in \Sigma$ which is equivalent to $\Omega \in \Sigma$. □

In this thesis we will use the notation $\Pr[E]$ to denote the probability of event E. In most cases we will assume it's evident what the sample space Ω and event space Σ are, and won't explicitly define them. Before we can continue further we must define what a probability measure is.

Definition 2.3 (Probability Measure)**.** *Let Ω be a sample space on $\mathcal{E}$ and Σ be the event space of $\mathcal{E}$. A Probability measure on (Ω, Σ) is then a function $\Pr : \Sigma \to [0,1]$ which satisfies:*

1. $\Pr[\Omega] = 1$
2. Let $A_1, A_2, \ldots$ be a countable sequence of pairwise disjoint events in Σ. Then:
$$\Pr\left[\bigcup_{i\geq 1} A_i\right] = \sum_{i\geq 1} \Pr[A_i] \tag{2.1}$$
   This is known as the *countable additivity* property.

This definition can come in other, equivalent, variations. We have chosen for the simplest one. Many definitions also require that $\Pr[\varnothing] = 0$. In our case however, this is a property that follows from the definition.

Lemma 2.2. *From the definition of a probability measure* $\Pr$ *on* (Ω, Σ) *it follows that* $\Pr[\varnothing] = 0$.

Proof. We know that $\Pr[\Omega] = 1$. As the sets Ω and $\varnothing$ are disjoint, we can apply the countable additivity problem to $\Omega \cup \varnothing = \Omega$:

$$\begin{aligned} 1 &= \Pr[\Omega] \\ &= \Pr[\Omega \cup \varnothing] \\ &= \Pr[\Omega] + \Pr[\varnothing] \\ &= 1 + \Pr[\varnothing] \end{aligned}$$

From this we can see that $\Pr[\varnothing] = 0$. □

2.1.1 Conditional Probability

One may sometimes possess incomplete information about the actual outcome of an experiment without knowing its outcome exactly. If A and B are events in Σ and we are given that B occurs, then, taking into account this information, the probability of A may no longer be $\Pr[A]$. We can see that A occurs if and only if $A \cap B$ occurs. This suggests that the new probability of A is proportional to $\Pr[A \cap B]$ [GW86].

Definition 2.4. *The conditional probability of event A given event B is defined as*

$$\Pr[A \mid B] \overset{\text{def}}{=} \frac{\Pr[A \cap B]}{\Pr[B]}$$

If $\Pr[A \mid B] = \Pr[A]$ we call the events A and B independent. We can now prove that this definition gives rise to a new probability space.

Theorem 2.3. *If $B \in \Sigma$ and $\Pr[B] > 0$ then (Ω, Σ, Q) is a probability space where $Q : \Sigma \to \mathbb{R}$ is defined by $Q(A) = \Pr[A \mid B]$.*

Proof. We need to verify that Q is a probability measure on (Ω, Σ). Clearly we already have that the image of Q is the interval $[0, 1]$. We also have that

$$Q(\Omega) = \Pr[\Omega \mid B] = \frac{\Pr[\Omega \cap B]}{\Pr[B]} = 1$$

It only remains to check that Q satisfies equation 2.1. So let $A_1, A_2, \ldots$ be a countable sequence of pairwise disjoint events. Then

$$
\begin{aligned}
Q\left(\bigcup_{i\geq 1} A_i\right) &= \frac{1}{\Pr[B]}\Pr\left[\left(\bigcup_{i\geq 1} A_i\right)\cap B\right] && (2.2)\\
&= \frac{1}{\Pr[B]}\Pr\left[\bigcup_{i\geq 1}(A_i\cap B)\right] && (2.3)\\
&= \frac{1}{\Pr[B]}\sum_{i\geq 1}\Pr[A_i\cap B] && (2.4)\\
&= \sum_{i\geq 1} Q(A_i) && (2.5)
\end{aligned}
$$

Where equality 2.4 follows from the fact that Pr is a probability measure on (Ω, Σ). □

2.1.2 Discrete Variables

Intuitively a random variable is a numerical representation of the outcome of an experiment. Random variables can be either discrete or continuous. Here we will begin by defining discrete random variables.

Definition 2.5 (Discrete Random Variable [GW86]). *Let $\mathcal{E}$ be an experiment with probability space $(\Omega, \Sigma, \Pr)$. A discrete random variable is then a function $X : \Omega \to \mathbb{R}$ such that:*

1. The image of X is a countable subset of $\mathbb{R}$
2. $\forall x \in \mathbb{R} : \{\omega \in \Omega : X(\omega) = x\} \in \Sigma$

If X is a random variable the notation $x \in X$ is defined as a shorthand notation of $x \in Im(X)$. In other words, it means that x can be any value of the image of the random variable X. We must note that there exists more general definitions where the image of the random variable can be a set different than $\mathbb{R}$ [FG96]. Such definitions are based on measure theory and are out of scope for this thesis. Based on our definition we can define the probability mass function of a discrete random variable as follows.

Definition 2.6. *Given a probability space $(\Omega, \Sigma, \Pr)$, the probability mass function of a discrete random variable X is the function $p_X : \mathbb{R} \to [0, 1]$ defined as*

$$\forall x \in \mathbb{R} : p_X(x) = \Pr[\{\omega \in \Omega : X(\omega) = x\}]$$

Remark that if x is not an element of the image of X, the considered set is in definition 2.6 empty. From lemma 2.2 it then follows that $p_X(x) = \Pr[\varnothing] = 0$. Instead of the notation $p_X(x)$ we will frequently use the equivalent notation $\Pr[X = x]$. A consequence of using the probability measure to define the probability mass function is the property that

$$\begin{aligned} \sum_{x \in X} p_X(x) &= \sum_{x \in X} \Pr[\{\omega \in \Omega : X(\omega) = x\}] && (2.6) \\ &= \Pr[\Omega] = 1 && (2.7) \end{aligned}$$

By using the probability mass function (pmf) we can define the cumulative distribution function (cdf) as:

$$F(x) = \Pr[X \leq x] \overset{\text{def}}{=} \sum_{y \leq x} \Pr[X = y] \tag{2.8}$$

2.1.3 Continuous Variables

Discrete random variables take only countable many values and their cumulative distribution functions generally look like step functions. On the other hand, the cdf of a continuous random variable is smooth. More formally this gives rise to the following definition

Definition 2.7 (Continuous Random Variable [GW86])**.** *Let $\mathcal{E}$ be an experiment with probability space $(\Omega, \Sigma, \Pr)$. A continuous random variable is then a function $X : \Omega \to \mathbb{R}$ whose cumulative distribution function may be written in the form*

$$F(x) = \Pr[X \leq x] = \int_{\infty}^{x} f(u)\mathrm{d}u$$

for all $x \in \mathbb{R}$, for some non-negative function f. We say that X has probability density function f.

Analog to discrete variables, we again note that there exists more general definitions where the image of the random variable can be a set different than $\mathbb{R}$ [FG96].

A continuous variable is thus defined by its cumulative distribution function (cdf) $F(x)$ which equals

$$F(x) = \Pr[X \leq x] = \Pr[X < x]$$

The seconds equality holds because $\Pr[X = x] = 0$. Interestingly this means that although the probability of an event is defined as being zero, the event can still occur[1]. As is clear from the definition, the probability density function (pdf) of a continuous random variable is the derivative of the cdf:

$$f(x) = \frac{d}{dx} F(x)$$

It describes the relative likelihood for this random variable to occur at a given point. We can calculate the probability that it falls within an interval by taking the integral of $f(x)$ over the interval. It must also satisfy the following relation:

$$\int_{-\infty}^{+\infty} f(x) \mathrm{d}x = 1 \tag{2.9}$$

As a note of caution, in some papers the notation $\Pr[X = x]$ is abused to denote the probability density function $f(x)$ when talking about continuous variables. In this thesis we have avoided this abuse of notation to prevent ambiguity. Sometimes we will use the *proportional to* symbol $\propto$ to state that a random variable has a certain probability density function. The notation $f(x) \propto g(x)$ means that there exists a constant $k \in \mathbb{R}$ such that $f(x) = kg(x)$. An example would be:

$$f(x) \propto \exp\left(-\frac{|x - \mu|}{b}\right) \tag{2.10}$$

This is a shorthand notation for the pdf without specifying the constant that is needed for it to satisfy equation 2.9. In equation 2.10 the constant k would be $1/2b$. A similar notation can be used for the probability mass function of discrete variables in combination with equation 2.6.

Finally, the term *probability distribution* is used to refer to either the probability mass function or the probability density function, depending if the context is about discrete or continuous variables respectively.

2.1.4 Transformation of a Random Variable

Many times it's useful to transform a random variable X to a new random variable Y using a given function f. This new random variable Y will be denoted by $f(X)$. The definition of $f(X)$ is

Definition 2.8. *Given a random variable X over the probability space $(\Omega, \Sigma, \Pr)$ and a function $f : \mathbb{R} \to \mathbb{R}$, we define $f(X)$ to be the function*

[1]This is because probability spaces are based on measure theory which doesn't really distinguish between sets of measure zero (infinitely unlikely) and the empty set (impossible).

$$\begin{aligned} f(X)\colon \Omega &\to \mathbb{R}, \\ \omega &\mapsto f(X(\omega)) \end{aligned}$$

This results in $f(X)$ being a random variable over the probability space $(\Omega, \Sigma, \Pr)$.

Lemma 2.4. *If X is a discrete random variable over $(\Omega, \Sigma, \Pr)$ we have that*

$$\Pr[f(X) = t] = \sum_{s \in S | f(s) = t} \Pr[S = s]$$

Proof. We begin with the definition of the probability mass function and the function $f(X)$:

$$\begin{aligned}
\Pr[f(X) = t] &= \Pr[\{\omega \in \Omega \mid f(X)(\omega) = t\}] && (2.11) \\
&= \Pr[\{\omega \in \Omega \mid f(X(\omega)) = t\}] && (2.12) \\
&= \Pr\left[\bigcup_{s \in S | f(s) = t} \{\omega \in \Omega \mid X(\omega) = s\}\right] && (2.13) \\
&= \sum_{s \in S | f(s) = t} \Pr[\{\omega \in \Omega \mid X(\omega) = s\}] && (2.14) \\
&= \sum_{s \in S | f(s) = t} \Pr[X = s] && (2.15)
\end{aligned}$$

In equation 2.12 the definition of the function $f(X)$ is used. We observe that in the union of equation 2.13 all the sets are pairwise disjoint. Hence we can transfrom this to equation 2.14 by the countable additivity property. □

2.1.5 Linearity of Expectation

To start from scratch we will begin with the definition of the expected value of a discrete random variable.

Definition 2.9. *If X is a discrete random variable, the expectation of X is denoted by $\mathrm{E}[X]$ and defined by*

$$\mathrm{E}[X] = \sum_{x \in X} x \Pr[X = x]$$

For a continuous variable this becomes $\mathrm{E}[X] = \int x f(x) \mathrm{d}x$. Linearity of expectation now roughly states that the expected value of a sum of random variables is equal to the sum of the individual expectations. The property is very useful when analyzing randomized algorithms and probabilistic methods. Written formally this becomes

Property 2.5. *Let $a, b \in \mathbb{R}$ and X and Y be any two random variables. We have that*

$$\mathrm{E}[aX + bY] = a\mathrm{E}[X] + b\mathrm{E}[Y]$$

The formula is even applicable if the variables are dependent, and is true for both continuous and discrete random variables.

2.2 Entropy

2.2.1 Self-information

Self-information (SI) is a concept in information theory that is a measure of uncertainty associated with a particular event [Kum01]. It can also be interpreted as a measure of the information content associated with the outcome of a random variable.

Definition 2.10. *Let E be an event and $\Pr[E]$ denote the probability of this event. Then the self-information of event E is*

$$I(E) = \log(1/\Pr[E]) = -\log \Pr[E]$$

If you consider tossing a coin, the chance of tail or heads is fifty-fifty. Thus the self-information of the event tail is $I(\{tail\}) = -\log 0.5 = 1$. Another example is throwing a fair dice. Here any result has self-information $I(E) = -\log \frac{1}{6} = 2.585$. The term is sometimes also called the amount of surprise of *surprisal* [Kum01]. The name comes from the fact that a highly improbable outcome has a large amount of self-information. Another commonly used synonym is *information content*.

The reason for using the log function in the definition is to make the self-information additive. The additive property means that the SI of events A and B equals the sum of the SI of the individual events. In other words: $I(A \text{ and } B) = I(A) + I(B)$ if A and B are independent events. This makes intuitive sense: The information obtained from the occurrence of two independent events is the sum of the information obtained from the occurrence of the individual events. If there is a dependence between the two events this property does not hold.

2.2.2 Shannon Entropy

As self-information measures the amount of uncertainty of a particular event, shannon entropy measures the uncertainty contained in a probability distribution. It can also be seen as a measure of the average information content one is missing when one does not know the outcome of the random experiment. In short it can be seen as a measure of unpredictability or disorder.

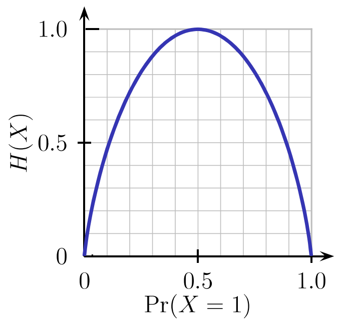


Figure 2.1: Shannon entropy of a family of coin toss events where the probability $\Pr[X = 1]$ ranges from 0 to 1.

Definition 2.11. *Let X be a discrete random variable with probability mass function $p_X(x)$. Then the shannon entropy of X is*

$$H(X) = -\sum_{x \in X} P_X(x) \log P_X(x)$$

If $P_X(x) = 0$ for some x, we will let the term $0 \log 0$ be equal to 0. See figure 2.1 for a graph of the shannon entropy of a binary coin toss event. The probability on the x-axis denotes the probability of the event $X = 1$, and this also implies the probability of event $X = 0$. It can be shown that the shannon entropy is equal to the expected amount of self-information of the discrete random variable.

2.2.3 Min-entropy

The disadvantage of the shannon entropy is that it can be very high, although the outcome of the experiment can still be predicted in most cases. This can for example happen if there is one event with probability 0.99 and there are countless other events with a much smaller probability. Shannon entropy is therefore not a good measure when we want to specify that something is hard to guess. Instead we will use min-entropy for this purpose.

Definition 2.12. *Let X be a discrete random variable with probability mass function $p_X(x)$. The min-entropy of X is*

$$H_\infty(X) = \min_{x \in X} (-\log P_X(x)) = -\log \left(\max_{x \in X} P_X(x) \right)$$

Min-entropy and shannon entropy are actually instances of Rényi entropy, which is a family of functions to measure the randomness of an experiment. Rényi entropy is parameterized by α. Setting α to 1 gives shannon entropy, while setting α to ∞ gives min-entropy. For this reason we use the notation $H_\infty(X)$ to denote the min-entropy of an experiment.

2.3 Laplace distribution

The Laplace distribution is a continuous probability distribution. It is parameterized by its mean μ and the scale factor b.

Definition 2.13 (Laplace Distribution). *The probability density function of the Laplace distribution is defined as*

$$h(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

Here the "|" symbol doesn't stand for conditional probability, but signifies that μ and b are parameters of the function h. Most of the time we will let μ be zero and drop this parameter. To state that a random variable X follows the Laplace distribution the notation $X \sim Lap(b, \mu)$ can be used. If μ is zero we will frequently use the abbreviated notation $Lap(b)$. We have the following useful property.

Property 2.6. *If $h(y)$ denotes the pdf of the Laplace distribution $Lap(b)$, the inequality $\frac{h(y)}{h(y')} \leq \exp\left(\frac{1}{b}|y-y'|\right)$ holds.*

Proof. Let $h(y)$ denote the pdf of the Laplace distribution. We have

$$\frac{h(y)}{h(y')} = \frac{\exp\left(-\frac{|y|}{b}\right)}{\exp\left(-\frac{|y'|}{b}\right)} \tag{2.16}$$

$$= \exp\left(\frac{1}{b}(|y'| - |y|)\right) \tag{2.17}$$

$$\leq \exp\left(\frac{1}{b}|y-y'|\right) \tag{2.18}$$

Where the last inequality is a consequence of subadditivity, since it states that $\forall a, b \in \mathbb{R} : |a+b| \leq |a| + |b|$. Filling in $a = y' - y$ and $b = y$ results in

$$|y'| \leq |y' - y| + |y|$$
$$|y'| - |y| \leq |y' - y|$$

This concludes the proof of Property 2.6. □

2.4 Inequalities

2.4.1 Boole's inequality

Boole's inequality gives a very rough upper bound on the probability of certain events. The advantage of this property is that it holds even if the given events are dependent, and is only based on the probability space.

Property 2.7. *(Boole's inequality) For any finite or countable set of events $E_1, E_2, \ldots, E_n$, it holds that:*

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr[E_i] \tag{2.19}$$

It follows from the definition of a probability measure, namely from the countable additivity property. A consequence is that the probability function Pr also satisfies the following statement

$$\forall A, B : \Pr[A \cup B] \leq \Pr[A] + \Pr[B]$$

Boole's inequality is also known as the *union bound*.

2.4.2 Markov Bound

Markov's inequality relates probabilities to the expectation of a discrete random variable. We require the random variable to be non-negative.

Property 2.8. *(Markov Inequality) Let X be a non-negative random variable and $v \in \mathbb{R}$. Then*

$$\Pr[X \geq v] \leq \frac{\mathrm{E}[X]}{v}$$

Proof. The proof is straightforward if you begin with the definition of the expected value:

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{x \in X} x \Pr[X = x] && (2.20)\\
&= \sum_{x < v} x \Pr[X = x] + \sum_{x \geq v} x \Pr[X = x] && (2.21)\\
&\geq 0 + v \Pr[X \geq v] && (2.22)\\
\Pr[x \geq v] &\leq \frac{\mathrm{E}[X]}{v} && (2.23)
\end{aligned}$$

Where inequality 2.22 is true because we assumed X is non-negative. □

2.4.3 Hoeffdings' Inequality

Another useful inequality is was found by Wassily Hoeffding in 1963 [Hoe63]. It gives an upper bound on the probability for the sum of independent random variables to deviate from its expected value. Often it's used to prove that the result of a randomized algorithm is accurate with high probability.

Theorem 2.9 (Hoeffdings' Inequality)**.** *If $X_1, X_2, \ldots, X_n$ are independant random variables with $\forall i\colon a_i \leq X_i \leq b_i$ and with*

$$X = \frac{X_1 + X_2 + \ldots + X_n}{n}$$

then for $t > 0$

$$\Pr\left[X - \mathrm{E}\left[X\right] \geq t\right] \leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{2.24}$$

For a proof of this theorem we refer the reader to the original paper of Hoeffding [Hoe63]. The following upper bound trivially follows from theorem 2.9.

Theorem 2.10 (Hoeffdings' Inequality)**.** *If $X_1, X_2, \ldots, X_n$ are independent random variables with $\forall i\colon a_i \leq X_i \leq b_i$ and with $\overline{X}$ as in theorem 2.9 then for $t > 0$*

$$\Pr\left[\left|X - \mathrm{E}\left[X\right]\right| \geq t\right] \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{2.25}$$

Proof. The proof is follows from the equality

$$\Pr\left[\left|X - \mathrm{E}\left[X\right]\right| \geq t\right] = \Pr\left[X - \mathrm{E}\left[X\right] \geq t\right] + \Pr\left[-X + \mathrm{E}\left[X\right] \geq t\right] \tag{2.26}$$

On the first probability in the sum one can directly apply Hoeffdings' inequality. For the second probability we define the new variables $X'_i = -X_i$ and

$$X' = \frac{X'_1 + X'_2 + \ldots + X'_n}{n} = -X$$

notice that $\forall i\colon -b_i \leq X'_i \leq -a_i$. After doing this we can also apply the Hoeffding inequality on the second probably which is now equal to

$$\Pr\left[X' - \mathrm{E}\left[X'\right] \geq t\right]$$

Adding both bounds together completes the proof □

2.5 Probabilistic Turing Machine

In the following chapters we will often use randomized algorithms (also called randomized functions). They can be viewed in two equivalent ways. One way of viewing randomized algorithms is to allow the Turing machine to take random steps. This can be defined by letting the transition function have two possible legal next moves. An internal *coin toss* is then made to decide which result is chosen. Here both options have probability $\frac{1}{2}$. This is also how probabilistic Turing machines are defined commonly, as is for example done in the book of Sipser [Sip06]. The total amount of random choices made during the execution of a Turing machine are called the *internal coin tosses* of the machine [Gol01]. The output of the Turing machine is now no longer a string, but a random variable that can assume all possible strings as a value. This random variable will be denoted by $M(x)$ where x is the input string. We can then let $\Pr[M(x) = y]$ denote the probability that machine M on input x will output y [Gol01]. The probability space is taken over all possible outcomes of the internal coin tosses.

An alternative and equivalent way of viewing randomized algorithms is by giving the outcome of the internal coin tosses as auxiliary input. Hence the real input given to the machine will be (x, r) where x is the normal input and r represents a possible outcome of the coin tosses. Given Turing machine $M(x)$ that uses internal coin tosses, we will let $\overline{M}(x, r)$ stand for the corresponding deterministic Turing machine where the coin tosses are given as the auxiliary bitstring r. Since we will assume M always halts, there exists a $z \in \mathbb{N}$ such that M uses at most z coin tosses on every input x. Using these assumptions the probability space $(\Omega, \Sigma, \Pr)$ describing the distribution of the internal coin tosses becomes

- The sample space Ω is equal to $\{0,1\}^z$. That is, it's equal to all possible sequences of coin tosses.
- The event space Σ is equal to 2^Ω.
- Since we consider the coin tosses to be uniformly distributed, the probability measure for every ω in Ω is defined as $\Pr[\{\omega\}] = \frac{1}{2^z}$.

Here the internal coin tosses of the algorithm are represented by an element $\omega \in \Omega$. Without loss of generality we will assume that the algorithm M outputs only bitstrings. Based on this probability space, the random variable $M(x)$ is then the function

$$\begin{aligned} M(x)\colon \Omega &\to \{0,1\}^z, \\ \omega &\mapsto \overline{M}(x, \omega) \end{aligned}$$

Using these definitions we can rewrite the formula of the probability of event $M(x) = y$ to a more convenient and intuitive form:

$$\begin{aligned}
\Pr[M(x) = y] &= \Pr[\{\omega \in \Omega \mid M(x)(\omega) = y\}] && (2.27)\\
&= \Pr\left[\{\omega \in \Omega \mid \overline{M}(x,\omega) = y\}\right] && (2.28)\\
&= \sum_{\omega \in \Omega \mid \overline{M}(x,\omega)=y} \Pr[\{\omega\}] && (2.29)\\
&= \sum_{\omega \in \Omega \mid \overline{M}(x,\omega)=y} \frac{1}{2^z} && (2.30)\\
&= \frac{\left|\{\omega \in \Omega \mid \overline{M}(x,\omega\}\right|}{2^z} && (2.31)
\end{aligned}$$

For equation 2.28 we used the definition of $M(x)(\omega)$. Equation 2.29 follows from the countable additivity property. Hence we can conclude with the following straightforward formula

$$\Pr[M(x) = y] = \frac{|\{r \in \{0,1\}^z \mid \overline{M}(x,r) = y\}|}{2^z} \tag{2.32}$$

2.6 Statistical Distance

2.6.1 Definition

Often it can be useful to have a measure that quantifies the distance between two random variables or probability distributions. We will mainly use the notion of ϵ-close:

Definition 2.14 (ϵ-close)**.** *Let X and Y be two random variables over the same image S. Their statistical distance is*

$$\Delta(X,Y) \stackrel{\text{def}}{=} \max_{A \subseteq S} |\Pr[X \in A] - \Pr[Y \in A]| = \epsilon$$

If two distributions have statistical distance at most ϵ, we say they are ϵ-close. From this definition we can deduce that the statistical distance ranges between 0 and 1. For discrete random variables we have another formula to calculate the statistical distance.

Lemma 2.11. *Given two discrete random variables X and Y over the same image S we have that*

$$\Delta(X,Y) = \frac{1}{2}\sum_{s \in S} |\Pr[X = s] - \Pr[Y = s]|$$

Proof. Let X and Y be two discrete random variable over the same image S and let $A = \{s \in S \mid \Pr[X = s] > \Pr[Y = s]\}$. This choice of A makes $\Pr[X \in A] - \Pr[Y \in A]$ as large as possible and positive, meaning we have the following

$$\begin{aligned}
\Delta(X, Y) &= \max_{A' \subseteq S} \left|\Pr\left[X \in A'\right] - \Pr\left[Y \in A'\right]\right| && (2.33)\\
&= \Pr[X \in A] - \Pr[Y \in A] && (2.34)\\
&= \sum_{a \in A} (\Pr[X = a] - \Pr[Y = a]) && (2.35)\\
&= \sum_{a \in A} |\Pr[X = a] - \Pr[Y = a]| && (2.36)
\end{aligned}$$

Since both X and Y are discrete random variables equation 2.35 is valid. Equation 2.36 follows from the fact that we chose A such that for all a in A it's true that $\Pr[X = a] > \Pr[Y = a]$. Now let $A^c = S - A$. We have

$$\begin{aligned}
\Delta(X, Y) &= \Pr[X \in A] - \Pr[Y \in A] && (2.37)\\
&= 1 - \Pr[X \in A^c] - 1 + \Pr[Y \in A^c] && (2.38)\\
&= \Pr[Y \in A^c] - \Pr[X \in A^c] && (2.39)\\
&= \sum_{a \in A^c} (\Pr[Y = a] - \Pr[X = a]) && (2.40)\\
&= \sum_{a \in A^c} |\Pr[Y = a] - \Pr[X = a]| && (2.41)\\
&= \sum_{a \in A^c} |\Pr[X = a] - \Pr[Y = a]| && (2.42)
\end{aligned}$$

Where equation 2.41 is correct since $\forall a \in A^c : \Pr[Y = a] > \Pr[X = a]$. Adding up equation 2.36 and 2.42 we get

$$2\Delta(X, Y) = \sum_{a \in S} |\Pr[X = a] - \Pr[Y = a]|$$

Which is what we needed to prove. □

2.6.2 Properties

One important property is that the statistical distance cannot increase by applying a—possibly randomized—function. Although the property applies to both discrete and continuous variables, we will focus on discrete variables, since that is what we will need in order to prove the impossibility result in chapter 4.

Property 2.12. *Given two discrete random variables X and Y over the same image S, and a randomized function f, we have the following property*

$$\Delta(f(X), f(Y)) \leq \Delta(X, Y)$$

Proof. The proof when f can be a randomized function is cumbersome to write down in detail, and it analogous to the proof where only deterministic functions are considered. Therefore we will only give a proof when M is a deterministic function. So let X and Y be two discrete random variables over the same image S, and f a deterministic function with image T. We get

$$\begin{aligned}
\Delta(f(X), f(Y)) &= \frac{1}{2}\sum_{t\in T} \left|\Pr\left[f(X)=t\right] - \Pr\left[f(Y)=t\right]\right| && (2.43)\\
&= \frac{1}{2}\sum_{t\in T} \left|\sum_{s\in S|M(s)=t} \Pr\left[X=s\right] - \sum_{s\in S|M(s)=t} \Pr\left[Y=s\right]\right| && (2.44)\\
&= \frac{1}{2}\sum_{t\in T} \left|\sum_{s\in S|M(s)=t} \left(\Pr\left[X=s\right] - \Pr\left[Y=s\right]\right)\right| && (2.45)\\
&\leq \frac{1}{2}\sum_{t\in T} \sum_{s\in S|M(s)=t} \left|\Pr\left[X=s\right] - \Pr\left[Y=s\right]\right| && (2.46)\\
&= \frac{1}{2}\sum_{s\in S} \left|\Pr\left[X=s\right] - \Pr\left[Y=s\right]\right| && (2.47)\\
&= \Delta(X, Y) && (2.48)
\end{aligned}$$

Equality 2.44 follows from the definition of $f(X)$ and the fact that X and Y are discrete random variables. The inequality follows from the subadditivity property[2] of the absolute value. From this we can conclude that $\Delta(f(X), f(Y)) \leq \Delta(X, Y)$. Remark that if f is an injective function the inequality becomes an equality. □

This property implies that the acceptance probability of any algorithm on inputs from X differs from its acceptance probability on inputs from Y by at most $\Delta(X, Y)$. This is exactly what we will need in chapter 4.

[2]In this situation a common synonym of the subadditivity property is the triangle inequality.

Chapter 3

History

Reasoning about privacy is fraught with pitfalls [BZ06, Swe02, NS08, NSR11]. To further appreciate the difficulty of finding a proper privacy definition, we will look at previous attempts and investigate why they failed. This will show you why certain intuitive methods don't work and illustrates the importance of taking into account auxiliary information. We will then discuss k-anonymity and its variants. Finally we will explain the composition attack, which shows that k-anonymity and related definitions are fundamentally flawed.

3.1 Previous Attacks

In this overview of previous attacks, a de-identified database is a database where personal identifiable information such as the name, address or social security number of individuals are removed. The term re-identification denotes the process of de-anonymizing the de-identified data by associating it with the original individual. Most of the time this is be done by using auxiliary information. This can be *any* additional information that an adversary has access to, including older versions of the original database.

3.1.1 The AOL Search Fiasco

We briefly explained this event in section 1.4. As stated, AOL released around 20 million search queries from 65,0000 AOL users. In an attempt to preserve privacy they replaced the usernames with a randomly generated number and also carefully redacted obvious disclosive queries (examples are when individuals search for their own name or security number) [Dwo11]. However the complete search history of an individual can still be extracted. In turn this allowed people to identify individuals based on their search history, and thus clearly breaching their privacy.

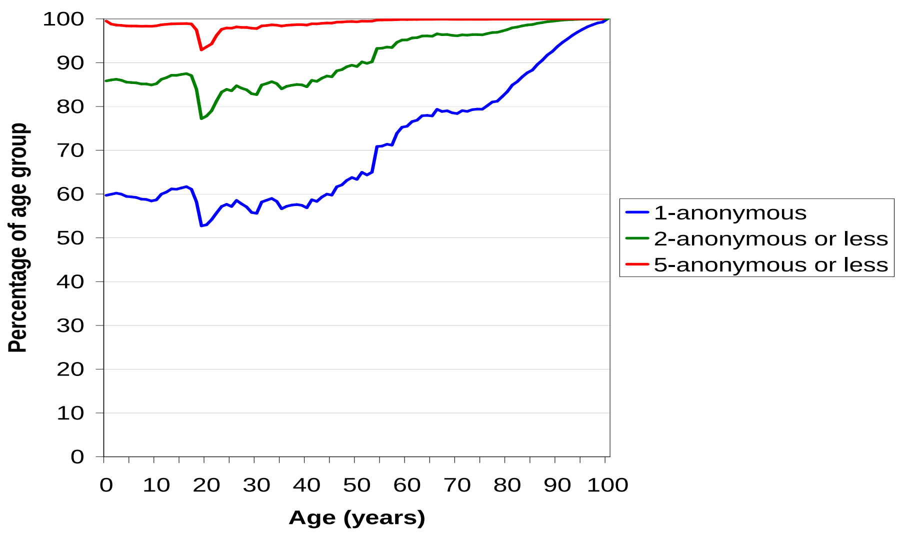


Figure 3.1: Anonymity of the U.S population when given gender, ZIP code and full date of birth [Gol06]

However we cannot view this as a true example that preserving privacy is a difficult task. This because no real care was given to anonymizing the search results. This can be deduced from the observation that the person who was responsible for releasing the data was quickly fired [And06]. A supervisor was also fired, and the chief technology officer resigned over the incident [MB06]. The release of this data is even more surprising because a few months earlier it was ruled that Google did not have to hand over any user's search queries to the U.S. government [Won06]. The government originally requested two months' worth of users' search queries as part of a subpoena.

3.1.2 The HMO records

The first notable demonstration of a relatively large privacy breach is the de-anonymization of the medical records of the governor of Massachusetts. This was done in a so called *linkage attack* between two different databases.

For more background on the attack, we begin with the following observation: One can uniquely identify 63% of the U.S. population with only knowing the gender, ZIP code and date of birth of an individual [Gol06]. The ZIP code is the place where the person currently lives. This is illustrated in figure 3.1. The blue line shows how many percent of the population can be uniquely identified given {gender, location, full date of birth}. The

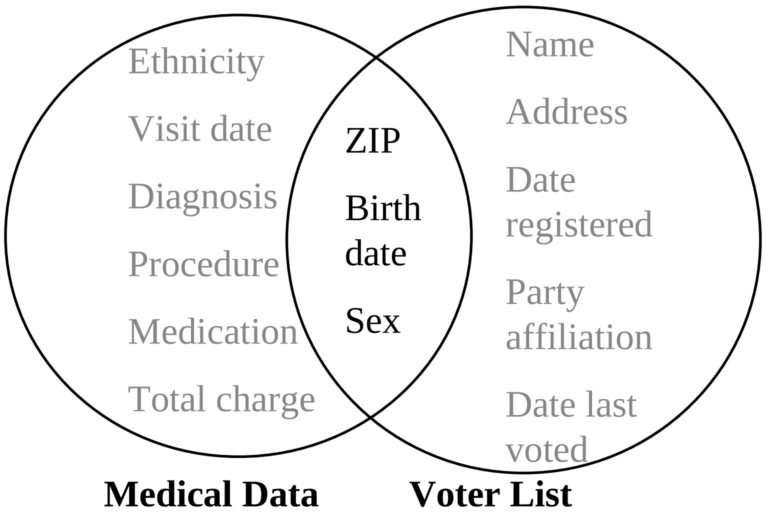


Figure 3.2: Re-identification by linkage attack [Swe01]

green and red lines show that even if a person is not uniquely identifiable, they still enjoy little privacy. For example, we can see that nearly all people are 5-anonymous or less. This means at most 5 other individuals have the same values for the given attributes. The definition of k-anonymity is explained in more detail in section 3.3.

An interesting observation is that people around the age of 20 seem to have more privacy. The cause of this peak is the college and university towns that have high concentrations of individuals between the ages of 18 and 22 [Gol06]. Also note that for ages above 50 the privacy decreases.

The HMO linkage attack is now performed by combining two databases that share the attributes {*sex, ZIP, birth date*} (see figure 3.2). The first database is the voter registration list for Cambridge Massachusetts, the second one the Massachusetts Group Insurance Commission (GIC) medical encounter data. In her attack, Sweeney bought the voter registration list. It included the attributes ZIP code, birth date, gender, name and address of each voter.

Medical data is also being collected, where the National Association of Health Data Organizations (NAHDO) recommends the collection of certain attributes. They include the patient's ZIP code, birth date, gender and diagnosis. A de-identified subset of this data, containing information for approximately 135,000 state employees, was handed over to the Group Insurance Commission (GIC). This was done because the GIC is responsible for purchasing health insurance for state employees. They later gave this data to researchers and sold a copy to industry because they believed the data was anonymous [Swe01].

Even though the medical databases did not contain explicit personal identifiers such as the name or social security number of a patient, individuals can still be identified by combining both databases. Using the attributes ZIP code, gender and birth day Sweeney was able to identify the medical records of William Weld. He lived in Cambridge Massachusetts and was a governor of Massachusetts, thus his information was in both tables and could be linked together.

3.1.3 Netflix

Netflix is a company that offers on-demand internet streaming video. To improve their movie recommendation algorithm, Netflix released a dataset containing more than 100 million movie ratings made by 480 thousand users on nearly 18 thousand movie titles [BL07]. To increase interest of the community, they offered a \$1 million prize to those who could improve the current system by at least 10%.

In an attempt to secure the privacy of its users they removed personally identifying attributes. All that remained are ratings (a score between 1 and 5), the date the rating was given, the anonymous ID of the user that gave the rating, and finally the title of the movie that was rated. Netflix also stated that only less than one-tenth of the complete dataset was published, and that data was also perturbated [Net]. First we will briefly cover the details of their anonymization process, and then we will describe how the dataset was de-anonymized.

We must also note that a similar attack was first done on the MovieLens dataset by Frankowski, Cosley, et al [FCS$^+$06]. In fact, the algorithms used in the NetFlix attack were improved versions of the methods used in the MovieLens attack.

Anonymization Process

Netflix stated that the released dataset was chosen randomly. However, when we plot the number of ratings X against the number of subscribers in the released dataset who have made X ratings, we seem to get an unusual result. As figure 3.3 shows, there are only few subscribers that rated less than 20 movies. One would actually expect that there are many people who only rate a few movies. A possible explanation is that only subscribers with more than 20 ratings were sampled, and that afterwards some movies were deleted.

Two subscribers of Netflix have located their own data in the released data set and inspected how the data were perturbed. Since they can see their original movie ratings we can compare how many ratings Netflix modified

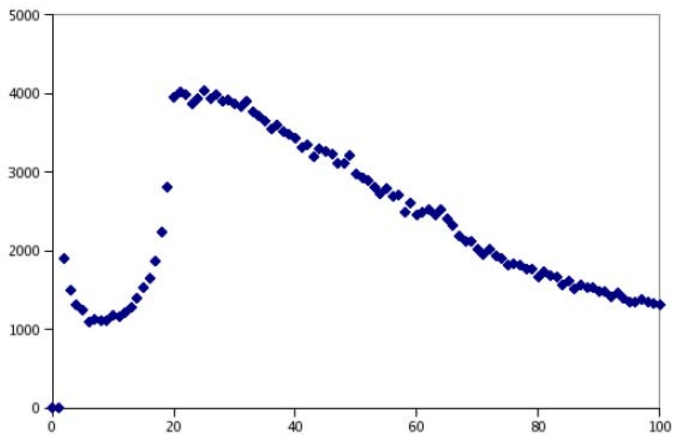


Figure 3.3: Netflix dataset: Number of subscribers with x ratings. The x-axis stands for the number of ratings a user has given in total, and the y-axis for how many people gave x ratings in total [NS08].

before releasing the data set. One of the subscribers had only 1 out of 306 ratings altered, and the other one had 5 out of 229 altered. This is only a small amount noise, and didn't effect the de-anonymization algorithm much. Here it's important to take into account that the dataset was released to develop improved recommendation algorithms. If there would have been a large amount of perturbation it would have greatly decreased the dataset's utility.

De-anonymization

In their paper Narayanan and Shmatikov [NS08] showed that you need very little auxiliary information to de-anonymize the average subscriber in the Netflix dataset. The algorithm they used even worked if the knowledge of the adversary is imprecise and contains a few incorrect values.

The actual attack was only a proof of concept where the anonymized Netflix dataset was linked with data crawled from the Internet Movie Database (IMDb). They only gathered a small sample of 50 IMDb users because IMDb's terms of service has restrictions on crawling. With this sample they were able to identify the records of two users. Because they had no absolute guarantees that these two matches were correct, they compared the matching score of the best candidate record to the second-best candidate record. They observed that the second-best candidate has a dramatically lower matching score, and thus concluded that the best match is highly unlikely to be a false positive.

You might argue that knowing which movies a person has watched is not an important privacy breach. This is not true, since one might be able to deduce important information from the movies you watch. The political orientation of a person may be revealed by his opinions about movies such as “Power and Terror: Noam Chomsky in Our Times” and “Fahrenheit 9/11”. Other information such as his religious view, sexual orientation, and so on can be deduced from the movies the person in question has viewed. Even if there is only one person in the Netflix dataset whose privacy is compromised, this is still unacceptable. After all, the privacy of *every* user must be guaranteed.

3.1.4 Kaggle

De-anonymization is not just a threat to privacy. It can also be used for gaming machine learning contests, as was done by Narayanan, Shi and Rubinstein [NSR11]. Before delving into the details, we will first explain the contest they were able to game.

Their target was the Kaggle social network challenge that promotes research on link prediction between nodes in social networks. In other words, contestants had to predict the relationship between two individuals in a social network (note that in this example relationships are represented by directed edges). The dataset that was created for the contest was made by crawling Flickr—an online social network to share photos—and saving the obtained nodes and edges in an anonymized database. A directed edge was included between two nodes if the corresponding person has marked the other person as a friend. The participants were given 7,237,983 edges of this network and using it they had to predict whether the relationships represented by an additional 8,960 edges are real or fake [Kag10]. To prevent cheating they anonymized the identities of all the nodes. Note that in this situation de-anonymizing the dataset doesn’t breach privacy since the data is already publicly available on the internet. Nevertheless the attack is an important example of the difficulty of releasing anonymized social network graphs.

The researchers first created their own graph by crawling Flickr themselves, and this graph will be referred to as the Flickr graph. We will call the graph released for the Kaggle contest as, not surprisingly, the Kaggle graph. The actual de-anonymization of the Kaggle graph was done in two steps. The first step was the biggest challenge and consists of de-anonymizing an initial number of nodes. In the second step the de-anonymization is propagated by selecting an arbitrary, still anonymous node, and finding the most similar node in the Flickr graph. The similarity of nodes is measured using the already de-anonymized nodes. We will now briefly cover both of these steps. Note that we will describe an almost identical attack in full detail in chapter 10.

Seed Identification

In this step a small subset of nodes are de-identified. Since both the Kaggle graph and the Flickr graph contain millions of nodes, it's essential to first reduce the search space. For this they reasoned that it's sensible to expect that nodes with high in-degree in both graphs roughly correspond to each other [NSR11]. Indeed, in the top 30 nodes in the Kaggle graph and the top 30 nodes in the Flickr graph (with respect to their in-degree), there were 27 highly likely correspondences. Hence, in order to reduce the search space, we can pick two small subsets of nodes, where all nodes have a high in-degree, knowing that with high probability there will be a large correspondence between these subsets.

The next step is to find the actual correspondence between these high in-degree nodes. To do this the cosine similarity of the sets of in-neighbors is calculated for every pair of nodes, for both the graphs. The set of in-neighbors of a node are all the nodes with a directed edge from themselves to the node under consideration. Here the cosine similarity between two set of nodes X and Y is defined as

$$sim(X, Y) = \frac{|X \cap Y|}{\sqrt{|X||Y|}}$$

The cosine similarity is then used to set the weight of an "edge" between the two respective high in-degree nodes, even if there wasn't an edge between those nodes in the original graph. We can then search for a mapping between these two weighted graphs that minimizes the differences of the corresponding edge weights. This is a known problem called *weighted graph matching* which can be solved using existing algorithms or heuristics. We end up with a mapping between the nodes in the small subgraphs, and continue with the propagation step.

Propagation

In order to propagate the de-anonymization, the algorithm stores a partial mapping between the nodes and iteratively extends this mapping. This extension is done by randomly picking an unmapped node in the Kaggle graph and finding the most similar node in the Flickr graph. If the similarity is high enough the nodes get mapped, otherwise no mapping is performed. To measure the similarity we again use the cosine similarity, but only consider the already mapped neighbors of the Kaggle node and the Flickr node. Here nodes mapped to each other are treated as identical when calculating the cosine similarity.

Conclusion

While the de-anonymization of the released social network graph didn't breach the privacy of any individual (since the data set was gathered by crawling public Flickr profiles), this is nevertheless a clear demonstration that *only* publishing nodes and edges can still result in privacy breaches. In fact, anonymizing a social network graph for public release is still an open problem and will be discussed in more detail in the second part of this work.

3.2 Previous Ideas

There have been several common suggestions to preserve privacy when analyzing data [Dwo11]. However, most of these don't provide the privacy guarantees we are looking for. It's worth repeating these previous attempts though, because it cannot be stressed enough that preserving privacy can be a difficult task with unexpected problems. This also gives additional insight into the privacy problem.

Large Query Sets

One frequent idea is to prohibit the execution of queries about a specific individual or small set of individuals. It's easy to show that this strategy is inadequate when we know a specific person X is in the database. What one can do is ask two large queries. The first one can be "How many people in the database are HIV positive?" and the second one "How many people, not named X, are HIV positive?". From both results we can deduce the HIV status of person X. Both were large queries yet the privacy of an individual was still breached.

Query Auditing

Another strategy is evaluating each query to the database in the context of the query history, to determine if a response would be disclosive. If the query is deemed disclosive it is refused and no answer is given. This method can be used to detect the previous example about the HIV status deduction about a specific individual. Though this might seem promising, monitoring all the queries is computationally infeasible [KPR00]. Even more problematic is that the refusal to respond to a query itself may be disclosive [KMN05]. For these reasons this approach is useless in practice.

Sampling

Strangely, sampling is sometimes offered as a potential solution to the privacy problem. Here a random number of individuals in the database will be chosen at random, and their information will be released. Properties and statistics can then be calculated on the returned subsample. When picking a sufficiently large subsample the results are representative of the dataset as a whole.

Considering the subsample is normally small when compared to the database size, it's unlikely that a specific individual is sampled. However each time a person is sampled his privacy is breached. This is unacceptable considering we want to protect the privacy of *every* individual in the database! So again this approach is not useful in real situations.

Randomized Response

Randomized response is a rather powerful mechanism that does provide a good form of privacy. Here the data is randomized before it is stored, and statistics can then be computed from this randomized data. Randomized response works by first letting the individual toss a coin. Based on this outcome he either answers the question truthfully or makes up a random answer. Privacy is guaranteed because of the uncertainty of how to interpret a given value. In other words every individual has plausible deniability: The responder is able to claim, with reasonable probability, that he gave a random answer.

This method was created for situations where individuals don't trust the data collector (curator). The chance that a person returns random data is selected so that when calculating statistics on all the returned data, the error is kept within an acceptable margin. While randomized response does provide privacy, the disadvantage is that it can introduce a lot of noise in the data, and that the approach becomes untenable for complex data.

Randomized Output

In randomized output randomly noise is added to the output. This is different from randomized response because there the data itself is randomized, while for randomized output the curator has access to the original data. Here it's the curator—also called the privacy mechanism—that will add random noise to the result of queries.

This method has promise and we will return to it in chapter 5 on differential privacy. We note that if done naively the approach will fail [Dwo11]. To see this, suppose the noise has mean zero and that independent randomness is used in generating every response. In this case, if the same query is asked several times, the responses can be summed and averaged, and the true answer will eventually be known.

3.3 K-anonymity and Variants

K-anonymity is a non-interactive privacy mechanism that was invented by Sweeney [Swe02]. Recall from section 1.6 that in the non-interactive setting a sanitized version of the database is released to the public. At first k-anonymity was considered as a potential solution to the privacy problem, mainly because of its conceptual simplicity and due to efficient algorithms that guarantee k-anonymity. Unfortunately it was soon clear that it had shortcomings and is unable to protect the privacy of all individuals. We will begin with explaining k-anonymity. Then we consider several improvements of k-anonymity and explain the specific problem each one was designed to avoid.

Notation In the following sections we will let D denote a bag[1] of tuples, where each tuple corresponds to a single individual in the database. The anonymized version of D will be represented by R. In the remaining we will frequently use the terms tuple, row and individual interchangeably. Now let $A = A_1, A_2, \ldots, A_r$ be a collection of r attributes and t be a tuple in R. We will let the notation $t[A]$ be equivalent to $(t[A_1], \ldots, t[A_r])$ where each $t[A_i]$ denotes the value of attribute A_i in table R for t.

3.3.1 K-anonymity

Introduction

The idea behind K-anonymity is that, when given information about an individual from an external source, such as his name, birth date, ZIP code, gender, etc., it should be impossible to (accurately) find the row in the anonymized database corresponding to this individual. Intuitively this should result in being unable to gain information about this individual, since we are unable to locate his information in the database. This can be achieved by removing certain attributes in combination with modifying certain values before releasing the database.

When creating the anonymized database R, the first step is removing attributes that clearly identify individuals, called explicit attributes. These attributes are similar to primary keys: Knowing the value of an explicit attribute of an individuals allows you to find, with high probability, the row corresponding to this person. Examples of explicit attributes are address, name, social security number, phone number, and so on. However, there are also attributes that, when taken together, can potentially identify an individual. Such a collection of attributes is called a quasi-identifier.

[1]A bag is a generalization of a set that can contain duplicate items. A synonym is multiset.

Roughly speaking, a quasi-identifier contains those attributes that are likely to appear in other databases.

The precise definition of a quasi-identifier is based on the separation between sensitive and nonsensitive attributes. The value of a sensitive attribute should remain a secret, which means that in an anonymized database it should be impossible to link a sensitive value (i.e., a row containing a sensitive value) to a specific individual. The reasoning is that it is then impossible to learn the sensitive value of an individual from the anonymized database and that privacy is thus preserved. We note that for this reason a sensitive attribute cannot be an explicit attribute. All attributes other than the sensitive attributes are called nonsensitive attributes. It's also assumed that the selected sensitive attributes do not appear in other databases. Therefore when knowing the value of a sensitive attribute one cannot find the individual corresponding to this value, which also means that a sensitive attribute is never included in a quasi-identifier. We conclude that a quasi-identifier consists only of nonsensitive attributes. The definition then becomes

Definition 3.1 (Quasi-Identifier). *A set of nonsensitive attributes $Q = \{Q_1, \ldots, Q_r\}$ is called a quasi-identifier for a database D if*

$$\exists t \in D \colon \neg \big(\forall t' \in D \colon t = t' \vee t[Q] \neq t'[Q]\big)$$

In other words, there is at least one individual in the original database who can be uniquely identified using only the values of the attributes in Q. We denote the set of all quasi-identifiers by $\mathcal{QI}$.

Good examples of quasi-identifiers are {birth date, ZIP code, gender} and {occupation, age, ZIP code}. Technically the explicit identifiers are also quasi-identifiers, so {Social Security Number} and {address} are also examples. However, the explicit identifiers are removed from the database before an anonymous version is created [LLV07]. In order to illustrate the relation between quasi-identifiers and keys, we recall the requirement of a key over the attributes Q for a database D:

$$\forall t \in D \colon \neg \big(\exists t' \in D \colon t \neq t' \wedge t[Q] = t'[Q]\big)$$

Based on this we can define an *anti-key*, which although strictly speaking is not the opposite of a key, can be seen as such. The requirement of an anti-key is

$$\forall t \in D \colon \exists t' \in D \colon t \neq t' \wedge t[Q] = t'[Q]$$

We see that the opposite (negation) of an anti-key is a quasi-identifier. To talk about a group of individuals that have the same values for a set of quasi-identifiers we introduce the definition of an equivalence class[2]:

[2]Sometimes we will use the terminology "group of individuals" as synonym for an equivalence class

	Non-Sensitive			Sensitive
	Zip code	Age	Nationality	Condition
1	130**	< 30	*	AIDS
2	130**	< 30	*	Heart Disease
3	130**	< 30	*	Viral Infection
4	130**	< 30	*	Viral Infection
5	1485*	≥ 40	*	Cancer
6	1485*	≥ 40	*	Heart Disease
7	1485*	≥ 40	*	Viral Infection
8	1485*	≥ 40	*	Viral Infection
9	130**	3*	*	Cancer
10	130**	3*	*	Cancer
11	130**	3*	*	Cancer
12	130**	3*	*	Cancer

Table 3.1: Example of a 4-anonymous database. The only quasi-identifier is the set of all the nonsensitive attributes, namely {Zip code, Age, Nationality}. [MGK07]

Definition 3.2 (Equivalence Class [GKS08]). *An equivalence class for a table R with respect to attributes in A is the set of all tuples $t_1, t_2, \ldots, t_i \in R$ for which $t_1[A] = t_2[A] = \ldots = t_i[A]$. That is, the projection of each tuple onto attributes in A are the same.*

Take table 3.1 as an example. In the next section we will see that this represents a 4-anonymous database. This means that each equivalence class contains at least 4 individuals with respect to the single quasi-identifier. The equivalence classes are $\{1,2,3,4\}$, $\{5,6,7,8\}$ and $\{9,10,11,12\}$, as each class has the same values for Zip code, Age and Nationality.

K-anonymity

K-anonymity now works by assuring there are at least k other individuals (i.e., rows in the database) that have the same values for all quasi-identifiers. This should assure that if we are searching for a specific individual, we will always get at least k results and thus can't identify the row of the individual.

Definition 3.3 (k-anonymity [GKS08]). *A release R is k-anonymous if for every tuple $t \in R$, and for every quasi-identifier $A \in \mathcal{QI}$, there exist at least $k-1$ other tuples $t_1, t_2, \ldots, t_{k-1} \in R$ such that $t[A] = t_1[A] = \ldots = t_{k-1}[A]$.*

A technique to achieve k-anonymity is called generalization, which replaces quasi-identifier values with values that are less-specific but semantically consistent. As a result, more records will have the same set of quasi-identifier values [LLV07]. An example of a 4-anonymous database is given in table 3.1, where the only quasi-identifier is $\{$Zip code, Age, Nationality$\}$. In the table a "*" denotes a suppressed value so, for example, "age=3*" means the age is in the interval $[30, 39]$. The definition of k-anonymity assumes that the curator is able to accurately recognize the quasi-identifiers. From the attacks described in section 3.1 we can see that reasoning about the quasi-identifiers can be a very hard task and is easy to get wrong. Specifically, determining the separation between sensitive and nonsensitive attributes can be problematic. This is clearly one of the shortcomings of k-anonymity: The assumption that one is accurately able find all the quasi-identifiers seems to be too strong.

As mentioned previously, k-anonymity is a creation of Sweeney and was an answer to the HMO linkage attack [Swe02]. In her paper she noted that one must be careful with subsequent releases of the same database. If not done correctly, privacy of some, or possibly all individuals, can be lost. This is a second important downside, as it is in practice almost impossible to know which data was already released previously. We will return to this problem in section 3.4 on the intersection attack.

Achieving k-anonymity

There are two commonly used techniques to achieve k-anonymity: Generalization and suppression. For both cases attribute values are being modified in order to assure the requirement of k-anonymity. In the generalization technique a value for an attribute is transformed to a more general and possibly "abstract" value. For example, given a zipcode such as 02139 we can drop the last number and generalize it to the interval $[02130, 02139]$. The loss of information that occurs is a price we have to pay in order to achieve privacy. Another example can be to generalize the ages of individuals to intervals such as $[0, 10[$, $[10, 20[$, and so on.

The second method is suppression. Here a value is simply not released at all. An example is shown in table 3.1 where no value is given for Nationality. Suppression can be seen as a specific form of generalization where the new value is the most generic value possible. We could have for example changed the nationality of each person to *Human*, which essentially gives us to new information at all. Or we could replace the age of a person with the single interval $[0, 200]$, which again gives no new information about the individual in question.

One can now formulate the problem of achieving k-anonymity whilst minimizing the number of generalizations or suppressions applied. This would result in the optimal anonymized table, where optimal means the least amount of information has been distorted, and thus this it can be seen as the most useful anonymized database. Even if we limit the problem to only allowing suppression of values we can prove that the optimization problem is NP-Hard [MW04]. Hence only approximation algorithms are used in practice. Because we will show in section 3.4 that k-anonymity fails to protect against multiple, possibly independent, releases, we will not cover these approximation algorithms to achieve k-anonymity. The focus in this chapter is on explaining the shortcomings of previously suggested privacy mechanism, and not their precise implementation. Example approximation algorithms can be found in the paper by Aggarwal, Feder, Kenthapadi, Motwani, et al [AFK$^+$05]. We remark that each of these algorithms can have different properties and advantages.

3.3.2 L-diversity

An improvement of k-anonymity is ℓ-diversity. It was invented by Machanavajjhala, Gehrke and Kife [MGK07] and was a reaction to the following two attacks on a k-anonymous database:

Homogeneity Attack In the example database shown in table 3.1 we notice that individuals 9, 10, 11 and 12 all have Cancer. So if we know that a person is in the database, and his zip code starts with 130 and is between 30 and 40 years old, we can deduce the sensitive information. We observe that the identity of the person is protected (we don't know his row in the database), but the sensitive attribute value is *not* protected (because we know the value). In general, if there is no diversity in the value of the sensitive attributes, the privacy of the individual is violated. If the number of tuples heavily outnumber the possible values of the sensitive attributes, this can be a common situation [MGK07].

Background Knowledge Attack The adversary may also possess background knowledge on the distribution of the sensitive values. As an example we can have a person aged 21 with zip code 13023 and we know he is in the released database. Thus row 1, 2, 3 or 4 contains his information. We also know he is very active in sports and is thus unlikely to have a heart disease. From this we can conclude that, with high probability, he has a viral infection.

We can prevent the homogeneity attack by assuring there is enough diversity in the sensitive values. Protection against arbitrary background knowledge is nearly impossible. We can however make it more difficult for the adversary: Again, if there is greater diversity in the sensitive attributes, more background knowledge will be needed to retrieve the exact value of the sensitive attribute. The definition of ℓ-diversity then becomes:

Definition 3.4 (Entropy ℓ-diversity [GKS08])**.** *For an equivalence class E, let S denote the domain of the sensitive attributes, and* $\Pr[E, s]$ *the fraction of records in E that have sensitive value s, then E is ℓ-diverse if:*

	Non-Sensitive			Sensitive
	Zip code	Age	Nationality	Condition
1	1305*	≤ 40	*	Heart Disease
2	1305*	≤ 40	*	Viral Infection
3	1305*	≤ 40	*	Cancer
4	1305*	≤ 40	*	Cancer
5	1485*	> 40	*	Cancer
6	1485*	> 40	*	Heart Disease
7	1485*	> 40	*	Viral Infection
8	1485*	> 40	*	Viral Infection
9	1306*	≤ 40	*	Heart Disease
10	1306*	≤ 40	*	Viral Infection
11	1306*	≤ 40	*	Cancer
12	1306*	≤ 40	*	Cancer

Table 3.2: Example of a 3-diverse database. The quasi-identifiers are all the nonsensitive attributes, namely {Zip code, Age, Nationality}. [LLV07]

$$-\sum_{s\in S}\Pr[E,s]\log\left(\Pr[E,s]\right)\geq\log(\ell)$$

A table is ℓ-diverse if all its equivalence classes are ℓ-diverse.

There exists also a simplified definition that states that every equivalence class must have at least ℓ different values for the sensitive attribute. An example of a 3-diverse database—according to the simplified definition—is shown in table 3.2. Based on the true definition of ℓ-diversity, the table is actually 2.8-diverse. We note that the equation in the definition is almost the same as the shannon entropy, where the probabilities—and so also the supposed discrete random variable—are now given as fractions over the sensitive attributes. This definition addresses the homogeneity attacks, and makes the background knowledge attack harder. The higher ℓ is, the more background knowledge is needed to reveal the sensitive attribute of an individual.

The introduction of ℓ-diversity is an important improvement of k-anonymity. However, it still has several flaws [LLV07]:

Difficult to achieve

Say the original database has only one sensitive attribute, namely the test result for a particular virus. This is basically a boolean value that is either positive or negative. Now assume that 99% of the population doesn't have the virus, and the database contains the information of 1000000 individuals. If we want to achieve any form of t-closeness every equivalence class must contain both negative and positive results. This means that there can be at most $100000 * 1\% = 1000$ equivalence classes, resulting in a large information loss when generalization will be applied. We also note that because the entropy of the sensitive attribute in the overall table is very small, assuring ℓ-diversity can only be done if we pick a smal enough ℓ, as for high values of ℓ it would in fact become impossible to assure ℓ-diversity. Although this is not a flaw, we mention it to illustrate the potential difficulty of acheiving ℓ-diversity.

Skewness Attack

Consider again the example where the sensitive attribute is the test result of a virus and 99% of the population doesn't have the virus. The ℓ-diversity principle now allows there to be an equivalence class with an equal number of positive and negative records. For any person that falls into this group his or her privacy is greatly reduced, because they are now considered to have a 50% probability of having the virus, instead of the 1% a priori probability. Here it's also important to note that if an attacker incorrectly thinks an individual has a certain value of the sensitive attribute, and behaves according this incorrect perception, this can also harm the individual.

Similarity Attack

Another potential problem is if the sensitive attribute values are distinct but semantically similar. For example, one equivalence class may contain various types of cancers. In this case we know that everyone in the group has cancer. Another example could be the salary of a person. Every individual in a group may have a unique value, but the overall interval can still be small. Hence we could fairly accurately guess the salary of each person by taking the average.

3.3.3 T-closeness

An improvement of ℓ-diversity, that attempts to solve the discussed problems, is called t-closeness. Here we attempt to assure that the distribution of a sensitive attribute is the same as the distribution of the attribute in the whole population. This results in the following definition.

Definition 3.5 (t-closeness [GKS08])**.** *An equivalence class E is t-close if the distance between the distribution of a sensitive attribute in this class and distribution of the attribute in the whole table is no more than a threshold t. A table is t-close if all its equivalence classes are t-close.*

The precise distance measure that is used is of little importance in our discussion. Although t-closeness is a clear advancement, we will see in section 3.4 that it's still vulnerable to a *linkage attack*. This is because t-

closeness—and also the previous methods—focus on static data that doesn't change. They are thus restricted to one-time publication and do not support republication of a new version of the database. This causes problems, because there might be database maintained by a different organization that stores almost identical information. If that particular organization would also release an anonymized database, the one-time publication assumption is no longer valid. In other words, in the real world it's impossible to prevent multiple publications of the same database, and thus privacy mechanism assuming one-time publication can are considered flawed in practice.

3.4 Intersection Attack

The attack that defeats k-anonymity, ℓ-diversity and t-closeness all at once is called the intersection attack. It's a particular instance of a composition attack, where multiple releases of the database—or overlapping databases released by different entities—are combined. A synonym of composition attack is a linkage attack. The intersection attack depends on an important property that a certain privacy mechanism may posses, namely that of locatability.

Definition 3.6 (Locatability [GKS08]). *Let Q_{values} be the set of quasi-identifier values of an individual in the original database D. An anonymization mechanism M that produces the anonymized database R on input D satisfies the locatability property if one can identify the set of tuples $\{t_1, \ldots, t_K\}$ in R that correspond to Q.*

In short it states that, given the values of the quasi-identifiers of an individual, we can find the equivalence class of that person. As explained in section 3.3 on k-anonymity there exists several algorithms that produce an anonymous database when given the original database. Most of these mechanism satisfy the locatability property, but it does not necessarily hold for all algorithms. For algorithms that do not strictly satisfy locatability, experiments suggested that simple heuristics can still locate a person's equivalence group with good probability [GKS08]. In the intersection attack we will assume the mechanism used to generate the anonymized table satisfies the locatability property.

3.4.1 The Algorithm

The intersection attack is now described in algorithm 3.1. The function GetEquivalenceClass returns the equivalence class where the individual is located in. Since we assumed the privacy mechanism that generated the anonymized releases R_j satisfies the locatability property this function always exists. The function SensitiveValueSet returns the set of distinct sensitive values for the members in a given equivalence class.

Algorithm 3.1 Intersection Attack

```
 1: $R_1, \ldots, R_n \leftarrow n$ independent anonymized releases
 2: $P \leftarrow$ a set of individuals common to all $n$ releases
 3: for each individual $i$ in $P$ do
 4:     for $j = 1$ to $n$ do
 5:         $e_{ij} \leftarrow \text{GetEquivalenceClass}(R_j, i)$
 6:         $s_{ij} \leftarrow \text{SensitiveValueSet}(e_{ij})$
 7:     end for
 8:     $S_i \leftarrow s_{i1} \cap s_{i2} \cap \ldots \cap s_{in}$
 9: end for
10: return $S_1, S_2, \ldots, S_{|P|}$
```

In essence the attack is straightforward. We are given set of individuals P of whom we know the value for the quasi-identifiers, and we know that their information is included in all the releases R_j. The function GetEquivalanceClass is based on the locatability property to retrieve the equivalence class of a given person based on the values for his or her quasi-identifiers. From every anonymized release we now deduce the possible set of values for the sensitive attribute. We then take the intersection of all these sets. If we end up with only one value, we know the sensitive attribute. For example, say that from an anonymous database A we know that Alice has either a heart disease or cancer. From a second anonymous database B we learn that Alice has either cancer or the flu. We can conclude:

$$S_{Alice} = \{\text{Heart Disease}, \text{Cancer}\} \cap \{\text{The Flu}, \text{Cancer}\} = \{\text{Cancer}\}$$

Clearly the privacy of Alice has been breached. We call this a perfect breach because we can deduce the exact sensitive value of an individual. In other words the adversary has a confidence level of 100% about the individual's sensitive data. Another possibility is that a partial breach occurred. This happens when the adversary is able to boil down the possible sensitive values to only a few values which itself could reveal a lot of information. Take a medical database for example, if we can boil down the sensitive values of the diagnosis to a few values such as Flu, Fever and Cold, we can conclude that the individual is suffering from a viral infection. The confidence level of the adversary in this example is $1/3 = 33\%$.

3.4.2 Empirical Results

In their paper on the composition attack, Ganta, Kasiviswanathan and Smith have tested the intersection attack on two databases [GKS08]. They are both from the census databases of the UCI Machine Learning repository. The first one is called the Adult database and has been used extensively in several k-anonymity experiments. The second is the IPUMS database which contains information from the 1997 census studies. From these inputs they generated k-anonymous, ℓ-diverse and t-close anonymized databases.

Without unnecessarily delving into the details, they noticed the privacy mechanisms failed to protect the privacy of all the individuals. It was also observed that ℓ-diversity and t-closeness performed better than the original k-anonymity definition, but still lead to considerable privacy breaches. Another observation is that for ℓ-diversity and t-closeness large equivalence classes had to be created, resulting in a heavy information loss.

Their experimental results also indicated that the severity of the intersection attack increases with the increase in entropy of sensitive attribute domain size [GKS08]. This means that if more possible values for the sensitive attribute are used in practice, the more severe the intersection attack becomes. The severity also increases as more independent databases are released to the public.

3.5 Conclusion

After discussing all the previous suggestions and examples it's clear that privacy is a difficult problem. When reasoning about it there are many pitfalls we must avoid. Nevertheless, each of these attempts is bringing us closer to a strong and usable privacy mechanism.

We also noticed that both in the real world examples, and for the theoretical privacy mechanisms such as k-anonymity and its variants, the background information that an adversary may possess is detrimental to the supposed privacy guarantees. These may be overlapping database, as was the case for the Netflix and IMDb databases. Or these may be different versions of the same databases, which is possible in an intersection attack. In particular it's very hard to specify the auxiliary information an adversary has access to. Ideally we need a privacy mechanism that can withstand all forms of auxiliary information.

Chapter 4

Impossibility Result

In 1977 Dalenius expressed a desideratum for statistical databases: nothing about an individual should be learnable from the database that cannot be learned without access to the database [Dal77]. This intuitively captures the essence of privacy: you will learn nothing new about the individual. Unfortunately this is impossible and the privacy of someone not even in the database can be compromised. We will require that our database and privacy mechanism is *useful*, and we will prove the impossibility of the desideratum in a general setting.

4.1 Motivation

As we have seen in the previous chapters, many ad hoc privacy mechanisms are flawed and can be broken. In order to guarantee adequate privacy for individuals we need a strong definition that can also be achieved in practice. Inspired by the semantically flavored definitions that are also used in cryptography, we turn our attention to the desideratum of Tore Dalenius: *Access to a statistical database should not enable one to learn anything about an individual that could not be learned without access.*

This is similar to the definition of a semantically secure cryptosystem, where it's stated that an adversary does not gain anything by looking at the ciphertext[Gol01]. Thus, whatever information an attacker can extract from the ciphertext, he could also extract from the information available a priori (without knowing the cipher text). This definition can be achieved in practice when we limit the computational power of the attacker[1].

Unfortunately the desideratum of Dalenius cannot be achieved in general. The problem is that of auxiliary information : Information that the

[1]In cryptography it's common to limit the computational power of the adversary to probabilistic polynomial-time Turing machines.

adversary already knows without access to the database (say from newspapers, medical studies, other databases, etc.). In any “reasonable” setting there is some piece of information that, together with access to the database, will compromise the privacy of an individual. Note that we have yet to give a proper definition of what privacy means, and in particular when a privacy breach occurs. This will be discussed in the next section.

Before beginning with the formal proof, we can capture the idea behind it with the following parable. An insurance company wants to minimize its losses, and to do so wants to refuse people that have an increased cancer risk. The company also knows that Alice, whom is asking for insurance, is a smoker. With only this auxiliary information about Alice the company has no valid reason to deny her insurance (i.e., we assume that it’s currently unknown if smoking causes cancer). Now consider the event that the company has access to a database that contains medical information about people. From this database they learn that if a person smokes, their risk on getting cancer greatly increases. Combining this newly learned information with the auxiliary information yields the conclusion that Alice has an increased cancer risk, which in this situation can be seen as a privacy breach.

The proof, including several parts of this chapter, is based on the works of Dwork and Naor [Dwo06, DN10].

4.2 The Setting

We start with a high level description of the proof. Assume the privacy mechanism provides some useful information to the analyst that can only be learned by accessing this privacy mechanism. Then consider a sensitive piece of information unknown to the analyst. With this we can construct an auxiliary information generator that uses the useful information provided by the privacy mechanism as a “password” to encrypt the sensitive information. A simple XOR cipher will be used to encrypt and decrypt the sensitive information. An adversary is then able to utilize the useful information to decrypt the auxiliary information and expose the sensitive information, while someone without access to the database will be unable to decrypt the auxiliary information.

4.2.1 Preliminaries

As we have seen in section 1.6 there are two natural models for privacy mechanisms: interactive and non-interactive. The proof is applicable to both settings.

As an approach to take into account any information one may have about the database prior to seeing any auxiliary information—in the proof this will mean before one sees the output of the auxiliary information generator—is to define a probability distribution $\mathcal{D}$ on databases. For example, this distribution can represent the fact that we know what *type* of information the database contains: It may contain the ages of people, so we know that the data type of an attribute is an integer, and we also know that the ages must lie between 0 and the age of the oldest known human.

In a sense $\mathcal{D}$ can also be seen as the schema of the database.

4.2.2 Assumptions

To prove the impossibility of Dalenius' desideratum we first need to define when a privacy breach occurs, what auxiliary information is available to the adversary, when a privacy mechanism can be considered useful and how we will model the adversary. We will try to be as abstract as possible in these definitions. This means the proof holds for a wide array of situations, so that in general the assumptions made in our proof will satisfy those in the real world.

Utility

As already mentioned in the description of this chapter, our proof needs some notion of utility. This means the privacy mechanism needs to output *useful* results. For example, whilst a mechanism that always outputs the same result or just returns a completely random answer each time preserves privacy, it's clearly useless.

To model usefulness, we require that there is a vector of questions whose answers should be learnable by the analyst. These answers shouldn't be known in advance (at least most of them), otherwise there is still nothing useful about them. We will therefore define the utility vector w that represents the answers to these questions. Without loss of generality[2] we assume that this is a binary vector of some fixed length κ. Intuitively we can think of this vector as answers to questions about the data. We assume that a large part of w can be computed from the raw data via the sanitization mechanism[3]. This excludes the case where the answers are given completely randomly, otherwise w cannot be *computed* from the raw data.

Since we don't know most of the answers in advance the utility vector can be seen as a random variable. To specify that we don't know all the answers in advance, we will assume that w is sufficiently hard to guess. This notion of "hard to guess" is formalized using the *min-entropy* measure that was introduced in section 2.2. This will—among other things—exclude the trivial case where the same answer is given to each question. The distribution $\mathcal{D}$ on databases induces a distribution on the utility vectors.

[2]In this case this means we can easily extend the proof where one considers larger domains

[3]We use the terms sanitization mechanism and privacy mechanism as synonyms

Privacy Breach

It's difficult to give a clear definition of when a privacy breach has occurred because because it's such a broad term and can mean different things to different people. So instead of quantifying what a privacy breach exactly is, we will use a more abstract idea and "describe" a privacy breach by a Turing machine $\mathcal{C}$ that simply tells us *whether or not a breach occurred.* It takes as input a distribution $\mathcal{D}$ on databases, a database DB drawn according to this distribution, and a string s—the supposed privacy breach—and outputs a single bit denoting whether the string is a privacy breach or not. We require that $\mathcal{C}$ always halt and we will use the convention that if s is a privacy breach, $\mathcal{C}$ will accept.

We say that an adversary *wins* with respect to $\mathcal{C}$ and for a given $(\mathcal{D}, DB)$ pair, if it produces a string s such that $\mathcal{C}(\mathcal{D}, DB, s)$ accepts. Henceforth "with respect to $\mathcal{C}$" will be implicit. It's important to remark that the string s contains the information that is a privacy breach for some individual. The requirements of the breach decider $\mathcal{C}$ are as follows:

1. It should be hard to find a privacy breach without access to the database and auxiliary information. In other words, if we only know $\mathcal{D}$ it should be unlikely to find a privacy breach. Otherwise there is no privacy to protect, since there is none to give up anyway. This is formalized in Assumption 4.1 part 2 (see page 58) where *even with* some form of access to the database it is hard to find a breach.

2. If the raw data would be released, it should be easy to find a privacy breach. Otherwise it's again trivial to preserve the privacy: even releasing *all the data* doesn't breach privacy. This is formalized in part 1b of Assumption 4.1.

Auxiliary Information

In the proof we will define a Turing machine that will generate the auxiliary information denoted by the string z. It takes as input a description of the distribution $\mathcal{D}$ from which the data is drawn as well as the database DB itself. In the proof we will give a precise description of how the string z is computed. The auxiliary information z will be given to both the adversary and a simulator. The simulator has no access of any kind to the database, whilst the adversary has access to the database via the privacy mechanism. The auxiliary information generator, including both its inputs, will be represented using the symbol $\mathcal{X}(\mathcal{D}, DB)$.

4.3 The Proof

The proof consists of two parts. In the first part it's assumed that the complete utility vector w can be learned, and in the second part it's assumed that it can only be partially learned. Although the second case captures the first one, the proof when w can be completely learned is simpler and easier to comprehend. We will therefore give a detailed proof when w can be completely learned, and only informally describe the second case.

We will let $San()$ denote the privacy mechanism. It stands for *sanitizer* and it answers queries in a way that provides some amount of privacy, i.e., it is hard to find a privacy breach using *only* the results of $San()$. We can view it as the interface to our database that the analyst must use in order to access the database. One can also see $San()$ as the trusted data curator that will attempt to provide privacy. Our results can be summarized using the following meta-theorem [DN10]:

Theorem 4.1 (Meta-Theorem [DN10])**.** *For any privacy mechanism $San()$ and privacy breach decider $\mathcal{C}$ there is an auxiliary information generator $\mathcal{X}$ and an adversary $\mathcal{A}$ such that for all distributions $\mathcal{D}$ where $\mathcal{D}$, $\mathcal{C}$ and $San()$ satisfy certain assumptions, and for all adversary simulators $\mathcal{A}^*$*

$$\Pr\left[\mathcal{A}(\mathcal{D}, San(\mathcal{D}, DB), \mathcal{X}(\mathcal{D}, DB)) \text{ wins}\right] - \Pr\left[\mathcal{A}^*(\mathcal{D}, \mathcal{X}(\mathcal{D}, DB)) \text{ wins}\right] \geq \Delta$$

where Δ is a suitably chosen (large) constant. The probability space is over choice of $DB \in_R \mathcal{D}$ and the coin flips of San, $\mathcal{D}$, $\mathcal{X}$, $\mathcal{A}$ and $\mathcal{A}^*$.

Here the adversary is represented by a probabilistic Turing machine. The *adversary simulator* $\mathcal{A}^*$ is the same as a normal adversary, but without access to $San()$. The probability space consists of the coin flips of all probabilistic Turing machines as defined in section 2.5 combined with the probability space of $\mathcal{D}$. Roughly the theorem states that—in a reasonable setting—an adversary with access to the database will likely find a privacy breach, while an adversary without access to the database much less likely to find a privacy breach. In particular there is a piece of auxiliary information z, so that z alone is useless to someone trying to find a privacy breach, whilst z in combination with access to the data through the privacy mechanism permits the adversary to win with probability arbitrarily close to 1.

4.4 Utility vector w is fully learned

As already stated several times, our proof requires some notion of utility as explained in section 4.2.2 and in general a "reasonable" (realistic) setting. Assumption 4.1 formalizes these requirements:

Assumption 4.1. *We assume a reasonable setting, namely that:*

1. We assume there exists $\ell \in \mathbb{Z}^+$, $\Upsilon \in \mathbb{R}^+$ and $0 \leq \gamma < 1$, such that both the following conditions hold:
    - (a) The distribution W on utility vectors has min-entropy at least $3\ell + \Upsilon$, that is $H_\infty(W) \geq 3\ell + \Upsilon$.
    - (b) Let $DB \in_R \mathcal{D}$. Then with probability at least $1 - \gamma$ over $\mathcal{D}$, DB has a privacy breach of length ℓ. Or in a more mathematical notation $\Pr\left[\exists y : |y| = \ell \wedge \mathcal{C}(\mathcal{D}, DB, y) = 1\right] \geq 1 - \gamma$.
2. For all interactive Turing machines $\mathcal{B}$, where μ is a suitably small constant that depends on $\mathcal{D}$ and $San()$, it must hold that

$$\Pr\left[\mathcal{B}(\mathcal{D}, San(\mathcal{D}, DB)) \text{ wins}\right] \leq \mu$$

   Where the probability is taken over the coin flips of $\mathcal{B}$ and the privacy mechanism $San()$, as well as the choice of $DB \in_R \mathcal{D}$.

Part 1a states that the utility vector contains enough *randomness.* This also makes sure the utility vector is useful, since this randomness means we don't know the answers in advance, and hence the answers can be consider useful. It also ensures that we can extract ℓ bits of randomness from the utility vector—see claim 4.2—which can be used as a one-time pad[4] to hide any privacy breach of the same length. Part 1b assures that with high probability there is some piece of information that can be considered a privacy breach. Finally, part 2 is a nontriviality constraint saying that the sanitization mechanism provides some form of privacy, at least against adversaries that have no access to auxiliary information.

We begin with showing that our assumption implies that, conditioned on most privacy breaches of length ℓ, the min-entropy of the utility vector is at least $\ell + \Upsilon$. This is needed so that the auxiliary information generator, given a privacy breach y of length ℓ, can extract enough randomness from w to encrypt y. Before giving the proof we define the following notation:

Definition 4.1 (Conditional Entropy)**.** *Given two discrete random variables A and B, the conditional min-entropy of A given the event $B = b$ is defined as*

$$H_\infty(A \mid b) \overset{\text{def}}{=} -\log\left(\max_{a \in A} \Pr\left[A = a \mid B = b\right]\right)$$

[4]In our scenario a one-time pad can be seen as a secret password used to encrypt and decrypt a message

We note that, as proven in section 2.1.1, the function $\Pr[A = a \mid B = b]$ (where b is given) defines a new probability space, and thus our definition of conditional entropy is consistent. Using this new notation we can proof the following claim.

Claim 4.2. *Let $\mathcal{D}$ be as in Assumption 4.1 part 1a. Consider any deterministic procedure $\mathcal{P}$ for finding a privacy breach given the database DB, so with probability $1 - \gamma$ (see part 1b of the assumptions) over $\mathcal{D}$ we have that $y = \mathcal{P}(DB)$ is a privacy breach. Let $\mathcal{P}(DB) = 0^\ell$ if there is no breach of length ℓ. Consider the distribution Y on y's obtained by sampling $DB \in \mathcal{D}$ and then computing $\mathcal{P}(DB)$. Then $\Pr[H_\infty(W \mid y) \geq \ell + \Upsilon] \geq 1 - 2^{-\ell}$, where the probability is taken over the choice $y \in Y$.*

Proof. It's given that $H_\infty(W) \geq 3\ell + \Upsilon$, which can be rewritten to

$$-\log\left(\max_{w \in W} \Pr[W = w]\right) \geq 3\ell + \Upsilon \tag{4.1}$$

$$\max_{w \in W} \Pr[W = w] \leq 2^{-(3\ell+\Upsilon)} \tag{4.2}$$

$$\forall w \in W : \Pr[W = w] \leq 2^{-(3\ell+\Upsilon)} \tag{4.3}$$

This followed straightforwardly from the definition of conditional entropy. We now note that $\Pr[W = w] = \sum_{y \in Y} \Pr[W = w \wedge Y = y]$. From this and equation 4.3 it follows that

$$\forall w \in W : \sum_{y \in Y} \Pr[W = w \wedge Y = y] \leq 2^{-(3\ell+\Upsilon)} \tag{4.4}$$

Specifically this means it's true that

$$\forall w \in W, \forall y \in Y : \Pr[W = w \wedge Y = y] \leq 2^{-(3\ell+\Upsilon)} \tag{4.5}$$

which—surprisingly enough—is the only thing we will need to complete the proof. We now continue by first by writing what we have to prove in a more readable and understandable form. To do this, let us consider the set G of y's that satisfy the event $H_\infty(W \mid y) \geq \ell + \Upsilon$

$$G = \{y \in Y \mid H_\infty(W \mid y) \geq \ell + \Upsilon\} \tag{4.6}$$

We can simplify the definition of G by noting that $H_\infty(W \mid y) \geq \ell + \Upsilon$ is equivalent to

$$\forall w \in W : \Pr[W = w \mid Y = y] \leq 2^{-(\ell+\Upsilon)} \tag{4.7}$$

From the definition of conditional probability we also have that

$$\Pr[W = w \mid Y = y] = \frac{\Pr[W = w \wedge Y = y]}{\Pr[Y = y]} \tag{4.8}$$

Using this we can further simplify the definition of set G to

$$G = \{y \mid \forall w \in W : \Pr[Y = y] \geq 2^{\ell+\Upsilon} \Pr[W = w \wedge Y = y]\} \tag{4.9}$$

Using this, the statement we now have to prove can be written as $\Pr[G] \leq 1 - 2^{-\ell}$ where the probability is taken over the complete set Y. This can equivalently be written as

$$\sum_{y \in G} \Pr[Y = y] \geq 1 - 2^{-\ell} \tag{4.10}$$

$$1 - \sum_{y \notin G} \Pr[Y = y] \geq 1 - 2^{-\ell} \tag{4.11}$$

$$\sum_{y \notin G} \Pr[Y = y] \leq 2^{-\ell} \tag{4.12}$$

The last statement is something we can indeed prove. We use the fact that $y \notin G$ means that there exists a w_y such that

$$\Pr[Y = y] < 2^{\ell+\Upsilon} \Pr[P = w_y \wedge Y = y] \tag{4.13}$$

We can now reason as follows

$$\sum_{y \notin G} \Pr[Y = y] < \sum_{y \notin G} 2^{\ell+\Upsilon} \Pr[W = w_y \wedge Y = y] \tag{4.14}$$

$$\leq 2^{\ell+\Upsilon} \sum_{y \notin G} 2^{-(3\ell+\Upsilon)} \tag{4.15}$$

$$\leq 2^{\ell+\Upsilon} 2^{\ell} 2^{-(3\ell+\Upsilon)} \tag{4.16}$$

$$= 2^{-\ell} \tag{4.17}$$

The first step follows from equation 4.13. The second step uses equation 4.5 that we derived from the given conditions. The third step is true because the number of y's is at most 2^{ℓ}. This finishes the proof. □

To describe the internal operation of the auxiliary generator we need one last element: A *strong randomness extractor*.

Definition 4.2 (Strong extractor [Sha02]). *A (k, ϵ)-strong extractor is a function*

$$Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

Such that for every distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$ the distribution $U_d \circ Ext(X, U_d)$ is ϵ-close to the uniform distribution on $\{0,1\}^{m+d}$. The two copies of U_d denote the same random variable[5]*.*

[5]Here $\circ$ stands for the concatenation of two strings.

The probability measure of the distribution $U_d \circ Ext(X, U_d)$ is defined as follows

Definition 4.3. *Let X be a discrete random variable and U_d the uniform distribution on $\{0,1\}^d$. The probability measure* Pr *corresponding to the probability space of the discrete random variable $P \overset{\text{def}}{=} U_d \circ Ext(X, U_d)$ is defined as*

$$\Pr[\{u \circ y\}] = \frac{1}{2^d} \sum_{x:Ext(x,u)=y} \Pr[X = x]$$

where y represents the result of the function $Ext(x, u)$. The sample space of P is equal to $\{0,1\}^{m+d}$. We now define the probability of the pairwise disjoint events $E = \{u_1 \circ y_1, \ldots, u_n \circ y_n\}$ as

$$\Pr[E] = \sum_{1 \le i \le n} \Pr[\{u_i \circ y_i\}] \tag{4.18}$$

Proof. We will now prove that this is indeed a valid probability measure. We can immediately see that the countable additivity property directly follows from equation 4.18 of our definition. The proof that $\Pr[\Omega] = 1$ is

$$\begin{aligned}
\Pr[\Omega] &= \sum_{u \in \{0,1\}^d} \sum_{y \in \{0,1\}^m} \Pr[u \circ y] && (4.19)\\
&= \sum_{u \in \{0,1\}^d} \sum_{y \in \{0,1\}^m} \frac{1}{2^d} \sum_{x:Ext(x,u)=y} \Pr[X = x] && (4.20)\\
&= \sum_{u \in \{0,1\}^d} \sum_{x} \frac{\Pr[X = x]}{2^d} && (4.21)\\
&= \sum_{u \in \{0,1\}^d} \frac{1}{2^d} && (4.22)\\
&= 1 && (4.23)
\end{aligned}$$

The first step holds because all the events $u \circ y$ are pairwise disjoint. In the second step the definition of $\Pr[u \circ y]$ is filled in. In the third step the last two summations are combined. Taken together they sum over all the possible values of x. For the fourth step we use the trivial property that $\sum_x \Pr[X = X] = 1$. Finally we get our desired result. The proof that the image of Pr is $[0,1]$ follows by noting that all terms in the calculation of Pr are non-negative and by the just proven property that $\Pr[\Omega] = 1$. This concludes the proof. □

A randomness extractor can be used to generate a random output that is shorter, but uniformly distributed, given a high-entropy source and a short random seed. In other words it can convert an input with "weak" randomness to an output with "high" uniform randomness. The additional requirement of a *strong* extractor is that the output, concatenated with the seed, should still yield a distribution that is close to uniform. There are several good constructions of strong extractors [NZ96, Sha02, DN10].

For this proof we can use an extractor based on the Leftover Hash Lemma:

Lemma 4.3 (Leftover Hash Lemma [ILL89])**.** *If* $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ *is a pairwise independent family where* $m = k - 2\log(\frac{1}{\epsilon})$*, then* $Ext(x,h) \overset{\text{def}}{=} h(x)$ *is a strong* (k,ϵ)*-extractor. Here we identify a function* h *with its index in* H*.*

For a proof, and more detailed explanation of this lemma, see the work of Impagliazzo et al. [ILL89]. The output of the extractor will be used to encrypt the auxiliary information z of length ℓ. Therefore we require a strong (k,ϵ)-extractor with $m \geq \ell$. From this requirement and Lemma 4.3 it follows that:

$$k - 2\log(\frac{1}{\epsilon}) \geq \ell \tag{4.24}$$

$$k \geq \ell + 2\log(\frac{1}{\epsilon}) \tag{4.25}$$

Hence, to extract enough randomness using the strong extractor from the Leftover Hash Lemma, the min-entropy of the utility vectors W conditioned on a given privacy breach y, must be at least $\ell + 2\log(\frac{1}{\epsilon})$. Combining inequality 4.25 with Claim 4.2, where it's stated that with high probability $H_\infty(W \mid y) \geq \ell + \Upsilon$, we can deduce that the choice of Υ and ϵ must satisfy $\Upsilon \geq 2\log(\frac{1}{\epsilon})$.

By utilizing this extractor we can describe how the auxiliary information generator $\mathcal{X}$ will produce the auxiliary information. We assume there is a (k,ϵ)-strong extractor with $m \geq \ell$ and $k \geq \ell + \Upsilon$. On input $(\mathcal{D}, DB \in_R \mathcal{D})$ it will do the following:

1. Find a privacy breach y for DB of length ℓ, which exists with probability at least $1-\gamma$ over $\mathcal{D}$. The efficiency of this operation will be discussed in section 4.6.

2. Compute the utility vector w using the sanitization mechanism. Remember that the utility vector w represents everything that can be learned from the database, and we assume that it can be fully learned using the interface $San()$. As described in section 4.2.2 the utility vector is represented by a binary vector of some fixed length κ.

3. Choose a random seed $s \in_R \{0,1\}^d$ and use the (k, ϵ) strong extractor to obtain from w an ℓ-bit almost uniformly distributed string r. In other words $r = Ext(w, s)$ where r will be ϵ-close to the uniform distribution.

4. The returned auxiliary information will be $z = (s, y \oplus r)$.

From Claim 4.2 and the definition of a strong extractor, the distribution of r is within statistical distance $\epsilon + 2^{-\ell}$ from the uniform distribution on $\{0,1\}^\ell$. This holds even if s and y are given.

The adversary $\mathcal{A}$ that has access to both the sanitization mechanism and the auxiliary information can decrypt the second element of the string z. To do this it first computes the utility vector w, and then from s obtains $r = Ext(w, s)$. The decryption step simply applies the XOR operation to retrieve the privacy breach $z \oplus r = (y \oplus r) \oplus r = y$. Given that a privacy breach occurs with probability at least $1 - \gamma$, the adversary $\mathcal{A}$ wins with probability at least $1 - \gamma$.

All that is left to do, is to investigate with which probability the adversary $\mathcal{A}^*$ can win. We start with part 2 of Assumption 4.1, that is to say $\Pr[\mathcal{B}(\mathcal{D}, San(\mathcal{D}, DB)) \text{ wins}] \leq \mu$, where the probability is taken over the coin flips of $\mathcal{B}$ and the privacy mechanism $San()$, as well as the choice of $DB \in_R \mathcal{D}$. If we remove access to San the probability that $\mathcal{B}$ wins cannot increase:

$$\Pr[\mathcal{B}(\mathcal{D}) \text{ wins}] \leq \mu \tag{4.26}$$

Where with $\mathcal{B}$ we denote the best machine, among all those with access to the given information, at producing a privacy breach. From this we can derive that the probability of correctly guessing a privacy breach y is bounded by μ. If we also give the adversary a randomly uniformly distributed string—here a better word for string would be noise—it gains no advantage. Specifically, the uniformly chosen bitstrings $s, u \in_R \{0,1\}^\ell$ will be given to $\mathcal{B}$, which cannot help it to win:

$$\Pr[\mathcal{B}(\mathcal{D}, s, u) \text{ wins}] \leq \mu \tag{4.27}$$

Let U_ℓ represent the uniform distribution on ℓ-bit strings. For any fixed string $y \in \{0,1\}^\ell$ we have $U_\ell = U_\ell \oplus y$. In particular this holds for all privacy breaches y of DB:

$$\Pr[\mathcal{B}(\mathcal{D}, s, u \oplus y) \text{ wins}] \leq \mu \tag{4.28}$$

We now want to replace the random uniformly chosen bitstring u with $r = Ext(w, s)$. However, it might be possible that r is correlated with y. If that is the case $y \oplus r$ might leak information about the privacy breach y. This is the reason we require Claim 4.2 which states that the min-entropy of

W, *conditioned on* y, must be at least $\ell + \Upsilon$. Now the output $r = Ext(w, s)$ will be close to uniform even if given y. That makes sure that $y \oplus r$ will never leak information about the privacy breach y.

As we have already seen, the statistical distance between the uniform distribution and the distribution of r is at most $\epsilon + 2^{-\ell}$. As we have proven in section 2.6.2 this means that when replacing $(s, u \oplus y)$ with $(s, r \oplus y)$ the acceptance probability of $\mathcal{B}$ differs at most by $\epsilon + 2^{-\ell}$. Hence we get

$$\Pr[\mathcal{B}(\mathcal{D}, s, r \oplus y) \text{ wins}] \leq \mu + \epsilon + 2^{-\ell} \tag{4.29}$$

Where $\mathcal{B}$ and its input is now equivalent with our simulator $\mathcal{A}^*$. We conclude that the probability that $\mathcal{A}^*$ wins is:

$$\Pr[\mathcal{A}^*(\mathcal{D}, \mathcal{X}(\mathcal{D}, DB)) \text{ wins}] \leq \mu + \epsilon + 2^{-\ell} \tag{4.30}$$

Finally we can calculate an upper bound between the chances that the adversary wins, and that the simulator wins. This gap is at least:

$$\Delta = (1 - \gamma) - (\mu + \epsilon + 2^{\ell}) = 1 - (\gamma + \mu + \epsilon + 2^{-\ell}) \tag{4.31}$$

This concludes our proof. We have shown that an entity with access to the database can—with high probability and in combination with auxiliary information—breach the privacy of an individual. The entity accesses the database through the privacy mechanism $San()$ which provides some amount of privacy. We have assumed that we can learn the full utility vector w by using $San()$. While $San()$ already provided some form of privacy, in combination with auxiliary information privacy of individuals can still be violated. The main result of our proof is that there always exists a piece of auxiliary information, that together with access to the database, can cause a privacy breach. However, without access to the database, this piece of auxiliary information is useless. All this is formalized in the following theorem.

Theorem 4.4 (Full Recovery of Utility Vector [DN10]). *Let Ext be a (k, ϵ)-strong extractor. For any privacy mechanism $San()$ and privacy breach decider $\mathcal{C}$ there is an auxiliary information generator $\mathcal{X}$ and an adversary $\mathcal{A}$ such that for all distributions $\mathcal{D}$ where $\mathcal{D}$, $\mathcal{C}$ and $San()$ satisfy Assumption 4.1 with $k \geq \ell + \Upsilon$ and for all adversary simulators $\mathcal{A}^*$*

$$\Pr[\mathcal{A}(\mathcal{D}, San(\mathcal{D}, DB), \mathcal{X}(\mathcal{D}, DB)) \text{ wins}] - \Pr[\mathcal{A}^*(\mathcal{D}, \mathcal{X}(\mathcal{D}, DB)) \text{ wins}] \geq \Delta$$

where $\Delta = 1 - (\gamma + \mu + \epsilon + 2^{-\ell})$. The probability space is over the choice of $DB \in_R \mathcal{D}$ and the coin flips of $San()$, $\mathcal{X}$, $\mathcal{A}$ and $\mathcal{A}^*$.

As already mentioned previously, because we used strong randomness extractors based on the Leftover Hash Lemma, it suffices that $\Upsilon \geq 2\log(\frac{1}{\epsilon})$.

4.5 Utility vector w is only partly learned

The proof can be extended to the more realistic case where the utility vector can only be partially learned. This is for example possible if the sanitization mechanism is randomized, and may add noise to the real output. A consequence is that one can only assume that the analyst will retrieve a vector w' within Hamming distance of the real utility vector. The reason a separate proof is needed is because the auxiliary information generator cannot know which utility vector w' is seen by the adversary.

The underlying idea of the proof remains the same: The returned utility vector will be used as a secret that can decrypt the auxiliary information. By decrypting the auxiliary information the adversary learns the privacy breach. For this reason we will not explain the details of the proof, as it only adds technical details for handling the fact that the utility vector can only be partially learned.

Without going into the details, the technicality that is used to solve our problem is called a *fuzzy extractor*. Originally, fuzzy extractors were created to derive cryptographic keys from biometric data [DORS08, DS05]. Here the original key (e.g., a fingerprint or iris scan) does not get reproduced precisely each time it is measured. By using fuzzy extractors however, we can derive a key that is always the same. This matches what we need: In our case the utility vector does not get learned precisely, yet we can still derive a key from it that is always the same. For the details of the proof see the paper by Dwork and Naor [DN10].

4.6 Discussion

Complexity

An objection to the proof might be that performing the attack is computationally infeasible to be executed in practice. However, this is only the case if finding a privacy breach for a given database is costly, since the complexity of all the other operations are low [DN10].

Statistical Distance Measure

Although we haven't given the proof if the utility vector is only partially learned, the actual proof of it focuses on the Hamming distance between two distributions. There exist also fuzzy extractors that are based on other distance measures, for example on the *edit distance* or *set distance*. Even so called general metric spaces can be used if the utility vector returned by the privacy mechanism is close to the original in those distance measures [DN10, DORS08].

It's also possible that the sanitization mechanism returns a utility vector w that must satisfy a certain relation (DB, w), although one may ask himself when such a thing would be useful in practice. In this case the result of the sanitization mechanism could vary a lot, and the (statistical) distance between two valid answers may be large. It's an open problem if the impossibility result also holds in this case [Nao10b]. In other words, for this scenario it's unknown if we can hide a privacy breach in the auxiliary information.

4.7 Conclusion

If the database is useful there is always some side information that, together with access to the database, will compromise the privacy of an individual. This means that absolute disclosure prevention is impossible: A secret will always be leaked, given that the adversary has enough auxiliary information. The surprising observation is that even the privacy of someone not in the database can be violated. In the parable mentioned in section 4.1 for example, we would learn that Alice had an increased cancer risk even if she wasn't in the medical database. This doesn't mean we should abandon the search for a proper definition that guarantees privacy. Instead we should move from the idea of assuring absolute disclosure prevention, to assuring relative disclosure prevention. This is done in the definition of differential privacy and is introduced in the next chapter.

Also note that we didn't restrict the computational power of the adversary, and perhaps more is realizable if we limit its power. We have also assumed that the returned utility vector is within a statistical distance from the real utility vector.

It may seem strange that Dalenius' goal cannot be achieved, while—when limiting the computational power of the adversary—semantic security for cryptosystems *can* be achieved. The reason behind this is that there is no difference between the adversary and a genuine user in our situation, while for cryptosystems they can be separated using secret keys. And since the database is designed to convey useful information, the adversary will inevitably also be able to access this information. Combined with auxiliary information the adversary is then able to find a privacy breach.

Chapter 5

Differential Privacy

In order to address the problems described in the previous chapters, we turn our attention to differential privacy. This definition takes into account any potential auxiliary information the adversary may have, and moves from absolute disclosure prevention to relative disclosure prevention. It has shown promise to be an adequate definition and is, at the time of writing, the only mechanism that can both handle auxiliary information and is *composable.* There are also abundant practical implementations that guarantee differential privacy.

5.1 Requirements

Before we give the definition of differential privacy, we will briefly look at two properties that a privacy mechanism must have.

Robustness to Side Information

It must take into account any side information (auxiliary information) that the adversary might have. In the definition we must avoid specifying exactly what the adversary may know, because this is very hard to quantify. Ideally it will assume the adversary knows *everything* except the individual information of a person.

Composability

If we would apply the sanitization mechanism several times on related databases, the adversary should not be able to combine both results to retrieve sensitive private information. As we have seen in Chapter 3 this is where a lot of previous suggestions have failed. For instance, k-anonymity has the problem that if it's done twice for slightly related databases, all privacy can be lost. For differential privacy general results have been proven about its composability [DRV10].

5.2 Definition

Since a privacy disclosure with respect to a given individual can happen even if that individual is not in the database, Dwork and McSherry propose the following idea: Any given disclosure (i.e., any output of the privacy mechanism) should be, within a small multiplicative factor, just as likely independent of whether the information if an individual is saved in the database or not. This means that including your personal information in a database doesn't significantly increase the chance of a disclosure happening. In other words, concealing your information gives only a nominal gain.

We begin by defining what a dataset is and when two datasets are adjacent.

Definition 5.1 (Dataset). *A dataset or database is finite collection of rows. Each row represents the information of one individual in the dataset.*

Definition 5.2 (Adjacency). *Two datasets are adjacent if one is a subset of the other, and the larger one contains at most one additional element. We also say that two adjacent datasets differ on at most one element.*

Adjacent databases are commonly also called *neighbours*. Since the definition of ϵ-differential privacy is based on randomized functions, we first have to rigorously define this notion.

Definition 5.3 (Randomized Function). *Let $\mathcal{F}$ be the set of all finite datasets and $\mathcal{V}$ be the set of all random variables with image V. A randomized V-valued function $\mathcal{K}$ is then defined as*

$$\begin{aligned}\mathcal{K} : \mathcal{F} &\to \mathcal{V},\\ D &\mapsto \mathcal{K}(D)\end{aligned}$$

where $\mathcal{K}(D)$ is a random variable defined separately for every $D \in \mathcal{F}$.

A lot of times the term *randomized algorithm* is used as a synonym of randomized function. Note that these randomized functions can actually be *continuous* random variables, and are thus different from probabilistic Turing machines. Strictly speaking $Range(\mathcal{K})$ should be $\mathcal{V}$. However, for ease of notation we will actually let $Range(\mathcal{K})$ be equal to V. We can now give the definition of differential privacy.

Definition 5.4 (Differential Privacy [DN10]). *A randomized function $\mathcal{K}$ gives ϵ-differential privacy if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq Range(\mathcal{K})$,*

$$\Pr\left[\mathcal{K}(D_1) \in S\right] \leq \exp(\epsilon) \times \Pr\left[\mathcal{K}(D_2) \in S\right]$$

It may seem unusual that we didn't require the probability distributions to be statistically close or that they be computationally indistinguishable, as is often done in the field of cryptography. The reason we don't use a distance measure such as ϵ-close is because with such a measure there might be an output where the difference in probability is relatively large, while the probability of all the other outputs could be equal, resulting in a small statistical distance. As an example consider the privacy mechanism where we randomly sample an individual and release all his data (i.e., return the row corresponding to the person). The probability of being picked when you participate in the database is $1/n$, where n is the size of the database, and 0 when you are not in the database. In this case the statistical distance would be small, yet clearly this does not preserve privacy [DMNS06]. The multiplicative factor $\exp(\epsilon)$ in our definition rules out this subsample-and-release paradigm, since it implies that an output whose probability is zero on a given database must have probability zero on any adjacent database, and by recursively applying the definition, also on any other database [Dwo11].

It's important to remark that a disclosure can still happen. However, the cause of the disclosure will not be the action of giving your individual information to the databases. Nor was there something you could do that would prevent the disclosure in the first place. This should address any concern the participant might have about the leakage of their personal information. After all, even if the participant removed her data from the database, no outputs—and hence possible privacy breaches—would become significantly more or less likely.

5.3 Variants

5.3.1 Discrete Range

To better understand the definition, we will focus on the case where the range of $\mathcal{K}$ is a finite or countable infinite set. In this case we can change the definition to:

Definition 5.5 (Differential Privacy (Discrete Sets))**.** *A randomized function $\mathcal{K}$ gives ϵ-differential privacy if for all data sets D_1 and D_2 differing on at most one element, where Range($\mathcal{K}$) is a* discrete set*,*

$$\forall x \in \mathit{Range}(\mathcal{K}) : \Pr\left[\mathcal{K}(D_1) = x\right] \leq \exp(\epsilon) \times \Pr\left[\mathcal{K}(D_2) = x\right]$$

If the set $\mathit{Range}(\mathcal{K})$ would be uncountable, the probability of the outcome $\mathcal{K}(D) = x$ would be zero, and the definition would trivially hold. It's clear that Definition 5.4 implies our new definition. On the other hand we can also show that, considering $Range(\mathcal{K})$ is finite or countable infinite, the new definition implies Definition 5.4.

Proof. Assume the privacy mechanism satisfies Definition 5.5. We have that $\forall S \subseteq Range(\mathcal{K})$ where $|S| = n$ and $S = \{s_1, s_2, \ldots, s_n\}$:

$$\begin{aligned}
\Pr\left[\mathcal{K}(D_1) \in S\right] &= \Pr\left[\bigcup_{i=1}^{n} s_i = \mathcal{K}(D_1)\right] && (5.1)\\
&= \sum_{i=1}^{n} \Pr\left[s_i = \mathcal{K}(D_1)\right] && (5.2)\\
&\leq \sum_{i=1}^{n} \exp(\epsilon)\Pr\left[s_i = \mathcal{K}(D_2)\right] && (5.3)\\
&= \exp(\epsilon)\sum_{i=1}^{n} \Pr\left[s_i = \mathcal{K}(D_2)\right] && (5.4)\\
&= \exp(\epsilon) \times \Pr\left[\mathcal{K}(D_2) \in S\right] && (5.5)
\end{aligned}$$

Where the first and last equality follow from the countable additivity property of the probability measure Pr. The inequality holds since we assumed that $\mathcal{K}$ satisfies Definition 5.5. We can conclude that Definition 5.4 is equivalent, for discrete ranges, to Definition 5.5. □

If the set $Range(\mathcal{K})$ were uncountable, the union in equation 5.1 would be over uncountable many events. Therefore the additivity property would no longer be applicable, and the proof does not hold. This gives us an insight as to way the definition of differential privacy uses the subset $S \subseteq Range(\mathcal{K})$, since using this makes the definition valid for both discrete *and* continuous variables.

5.3.2 Interpretation

From the definition of ϵ-differential privacy we notice that if the probability of an output—or a range of outputs for continuous variables—is zero for a specific data set, then this probability most be zero for *all* data sets. Practically this means the privacy mechanism $\mathcal{K}$ is never allowed to return this output, and that therefore the output is not contained in the Range of $\mathcal{K}$. Hence, without loss of generality, we can assume that the probability[1] is always nonzero.

Considering the definition must hold for data sets (D_1, D_2) as well as (D_2, D_1), where D_1 and D_2 differ in at most one element, we have:

[1]For discrete variables this means the pmf, for continuous variables we're actually talking about the cdf.

$$\Pr\left[\mathcal{K}(D_1) \in S\right] \leq \exp(\epsilon)\Pr\left[\mathcal{K}(D_2) \in S\right] \tag{5.6}$$
$$\Pr\left[\mathcal{K}(D_2) \in S\right] \leq \exp(\epsilon)\Pr\left[\mathcal{K}(D_1) \in S\right] \tag{5.7}$$

Since we can assume these probabilities are nonzero, we can rewrite this to:

$$\frac{\Pr\left[\mathcal{K}(D_1) \in S\right]}{\Pr\left[\mathcal{K}(D_2) \in S\right]} \leq \exp(\epsilon) \tag{5.8}$$
$$\frac{\Pr\left[\mathcal{K}(D_1) \in S\right]}{\Pr\left[\mathcal{K}(D_2) \in S\right]} \geq \exp(-\epsilon) \tag{5.9}$$

Combining these two inequality results in:

$$\exp(-\epsilon) \leq \frac{\Pr\left[\mathcal{K}(D_1) \in S\right]}{\Pr\left[\mathcal{K}(D_2) \in S\right]} \leq \exp(\epsilon) \tag{5.10}$$

This formulation of Definition 5.4 is easier to interpret. For small ϵ we observe that $\exp(\epsilon) \approx 1 + \epsilon$. Intuitively this means that both fractions should be close to each other, and consequently that the probabilities must be close to each other. This confirms our earlier statement that the output of the privacy mechanism is almost just as likely, independent of whether any individual opts in to, or opts out of, the database.

For completeness, note that inequality 5.10 is equivalent with the following inequality

$$\exp(-\epsilon) \leq \frac{\Pr\left[\mathcal{K}(D_2) \in S\right]}{\Pr\left[\mathcal{K}(D_1) \in S\right]} \leq \exp(\epsilon) \tag{5.11}$$

5.4 Choice of Epsilon

We have yet to define what the size of the parameter ϵ in the definition of differential privacy must be. Instead of letting ϵ be a fixed constant, we could've also defined it as a function whose image is $\mathbb{R}$. Taking inspiration from the field of cryptography, one may ask whether ϵ can be negligible in function of the database size.

Definition 5.6 (Negligible [Gol01])**.** *We call a function $\mu(x) : \mathbb{N} \to \mathbb{R}$ negligible if for every positive polynomial $p(.)$ there exists an N such that for all $n > N$,*

$$\mu(n) < \frac{1}{p(n)}$$

We will show that letting ϵ be negligible in function of the database size, denoted by n, the results of the privacy mechanism $\mathcal{K}$ become meaningless. Let D and D' be two data sets of both size n that differ in every element. Consider the sequence $D = D_1, D_2, D_3, \ldots, D_{2n} = D'$ where we get from D to D' by first removing an element, and then adding the corresponding modified element, until we have D'. For this we need at most $2n$ steps. All databases D_i and D_{i+1} differ in only one element and are thus adjacent. Let $\epsilon = \mu(n)$, where μ is a negligible function. For each of these adjacent databases we have

$$\Pr\left[\mathcal{K}(D_i) \in S\right] \leq \mathrm{e}^{\mu(n)} \Pr\left[\mathcal{K}(D_{i+1}) \in S\right] \tag{5.12}$$

Combining all these inequalities over the whole sequence gives us

$$\Pr\left[\mathcal{K}(D) \in S\right] \leq \mathrm{e}^{\mu(n)2n} \Pr\left[\mathcal{K}(D') \in S\right] \tag{5.13}$$

Because μ is a negligible function in n, the function $\mu'(n) = \mu(n)2n$ is also negligible. In other words, if we let ϵ be negligible in n, even two completely different data sets are treated as adjacent. Therefore the output of $\mathcal{K}$ should be similar for all databases, and thus these outputs would be useless.

Instead we let the parameter be a public value. Its choice will depend on how much privacy one wants to guarantee. The smaller the value, the more privacy. In most cases we will tend to think of ϵ as a small value such as 0.01, 0.1, ln(2) or ln(3). If the probability that some bad event will occur is very small, it might be tolerable to increase it by factors such as 2 or 3, while if the probability is already felt to be close to unacceptable, then even a small increase of e or $\mathrm{e}^{0.1}$ would be intolerable [Dwo08].

5.5 Achieving Differential Privacy

This section will show how one can achieve differential privacy. To get started we first investigate counting queries. Then we will see a more general property to assure differential privacy. Finally a few relatively more complex algorithms that achieve differential privacy will be explained.

5.5.1 Counting Queries

Assume we want to calculate how many rows satisfy a given predicate $P(x)$. What we will do is calculate the exact result and perturbate it with noise to guarantee differential privacy. Let D be a set of rows representing a database, f be the function that counts the true result, Y be a random variable used to add noise, and $\mathcal{K}$ be the resulting mechanism that will assure differential privacy. This can be written as

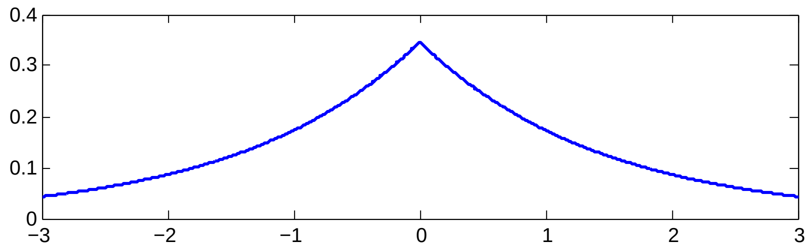


Figure 5.1: Pdf of noise added to counting queries using $\epsilon = \ln(2)$

$$\mathcal{K}(D) = f(D) + Y = \sum_{x \in D} P(x) + Y \tag{5.14}$$

If Y adds Laplace distributed noise, which has the probability density function $h(y) = \frac{1}{2b} \exp\left(-|y|/b\right)$, then we can show that $\mathcal{K}$ is differentially private.

Property 5.1. *Letting Y add Laplace distributed noise with parameter $b = \frac{1}{\epsilon}$ guarantees differential privacy.*

Proof. First observe that for any two database D_1 and D_2 differing in only one row, the sums $f(D_1)$ and $f(D_2)$ differs at most by one. Consider any $S \subseteq Range(\mathcal{K})$ and take $r \in S$. For $\mathcal{K}(D_1)$ to output r it must add $f(D_1)-r$ noise to the true answer, similarly $\mathcal{K}(D_2)$ must add $f(D_2) - r$ noise to the true answer for $\mathcal{K}$ to output the same r. Hence, the probability density that $\mathcal{K}(D_1)$ returns r is $h(f(D_1) - r)$, and for $\mathcal{K}(D_2)$ the probability density is $h(f(D_2) - r)$. We have

$$\frac{h(f(D_1) - r)}{h(f(D_2) - r)} \leq \exp\left(\frac{1}{b}|(f(D_1) - r) - (f(D_2) - r)|\right) \tag{5.15}$$

$$= \exp\left(\frac{1}{b}|(f(D_1) - f(D_2)|)\right) \tag{5.16}$$

$$\leq \exp\left(\frac{1}{b}\right) \tag{5.17}$$

The first inequality is proven in section 2.3 and the second inequality follows from the fact that $f(D_1)$ and $f(D_2)$ differ by at most one. Letting $b = \frac{1}{\epsilon}$, integrating over S with respect to r yields ϵ-differential privacy. $\square$

Figure 5.1 shows a graphical representation of how much noise is added to the true result. This is specifically for counting queries where one has chosen an ϵ values of $\ln(2)$. We observe that the noise is symmetrically centered around zero and that strictly speaking any value, no matter how small or large, is possible.

5.5.2 General Theorem to Achieve Privacy

An essential part of achieving differential privacy is knowing how much influence a single row has on the output of the query function. We will formally define this as the sensitivity of the query function.

Definition 5.7 (L_1 Sensitivity [Dwo11])**.** *The L_1 sensitivity of a function $f : \mathcal{D} \to \mathbb{R}^d$ is*

$$\begin{aligned}\Delta f &= \max_{D_1, D_2} ||f(D_1) - f(D_2)||_1 \\ &= \max_{D_1, D_2} \sum_{i=1}^{d} |f(D_1)_i - f(D_2)_i|\end{aligned}$$

for all $D_1, D_2 \in \mathcal{D}$ differing in at most one row.

As an example, the counting query in the previous section has a sensitivity of one. From the definition of sensitivity one can see that we also allow the function f to return a vector. To assure differential privacy we will add Laplacian noise to each element of this vector. If Y is a vector of d independent Laplace variables, the probability density of vector $y = (y_1, \ldots, y_d)$ is the multiplication of the probability density of each individual element:

$$\exp\left(\frac{-|y_1|}{b}\right) \times \ldots \times \exp\left(\frac{-|y_d|}{b}\right) = \exp\left(-\frac{|y_1| + \ldots + |y_d|}{b}\right) \tag{5.18}$$

$$= \exp\left(-\frac{||y||_1}{b}\right) \tag{5.19}$$

Using this we can prove the following theorem:

Theorem 5.2. *For $f : \mathcal{D} \to \mathbb{R}^d$, the mechanism $\mathcal{K}$ that adds independently generated noise with distribution $Lap(\Delta f/\epsilon)$ to each coordinate of the output assures ϵ-differential privacy.*

Proof. Analog to the proof of Claim 5.1, consider any subset $S \subseteq Range(\mathcal{K})$ and let $y \in S$. The pdf of $\mathcal{K}(D_1)$ returning y is $h(||f(D_1) - y||_1)$, and the pdf of $\mathcal{K}(D_2)$ returning the same y is $h(||f(D_2) - y||_1)$. We combine this as follows

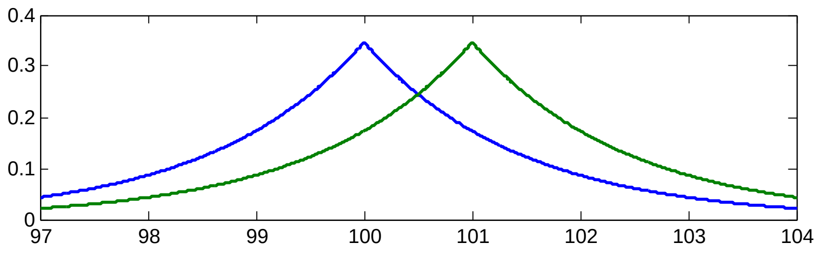


Figure 5.2: Comparison between the pdf's of two counting queries with $\Delta f = 1$ and $\epsilon = \ln(2)$. The true results of the queries are 100 and 101.

$$
\begin{aligned}
\frac{h(||f(D_1) - y||_1)}{h(||f(D_2) - y||_1)} &= \frac{\exp\left(-\frac{1}{b}||f(D_1) - y||_1\right)}{\exp\left(-\frac{1}{b}||f(D_2) - y||_1\right)} && (5.20)\\
&= \exp\left(\frac{1}{b}(||f(D_2) - y||_1 - ||f(D_1) - y||_1)\right) && (5.21)\\
&\leq \exp\left(\frac{1}{b}||f(D_2) - f(D_1)||_1\right) && (5.22)\\
&\leq \exp\left(\frac{\Delta f}{b}\right) && (5.23)
\end{aligned}
$$

Here the first inequality follows from the triangle inequality, and the second inequality from definition of Δf. Choosing $b = \Delta f/\epsilon$ and integrating over S yields ϵ-differential privacy. □

To visualize the effect of differential privacy on adjacent databases, figure 5.2 shows the situation for the case of a counting query where $\Delta f = 1$ and $\epsilon = \ln(2)$. The blue line show the probability density function when the true result is 100, and the green when the true result is 101. These two results could in practice come from two adjacent databases.

5.5.3 Adaptive Queries

In a real setting we want to execute more than one query. These queries may be independent of each other, or actually depend on previous results. For example, when we want to implement a more complex algorithm such as K-means we will perform multiple queries, where the current query depends on the results of the previous queries. This complicates the nature of the query function, as it is no longer a predetermined function, but rather a strategy for producing queries based on answers given thus far [DMNS06]. We will call these *adaptive queries*. The question is now how much noise must be added so the total algorithm—and thus the total sequence of queries—is ϵ-differentially private.

Theorem 5.3. *Let $f_1, f_2, \ldots, f_n$ be a sequence of queries where each query f_{i+1} depends on the output of $f_1, \ldots, f_i$. We define the total sensitivity as $\delta \overset{\text{def}}{=} \max(\Delta f_1 + \Delta f_2 + \ldots + \Delta f_n)$ where the maximum is taken over all possible outputs of $f_1, f_2, \ldots, f_n$. The mechanism $\mathcal{K}$ that adds independently generated noise with distribution $Lap(\delta/\epsilon)$ to the result of every function assures ϵ-differential privacy.*

Proof. For convenience we will let S denote the tuple of sets $(S_1, S_2, \ldots, S_n)$ such that $S_1 \subseteq \mathit{Range}(f_1), \ldots, S_n \subseteq \mathit{Range}(f_2)$. We will define the notation $(a_1, \ldots, a_n) \in S$ as equivalent to $a_1 \in S_1, \ldots, a_n \in S_n$. Let $f_1, f_2, \ldots, f_n$ be a sequence of queries where each query f_{i+1} depends on the output of $f_1, \ldots, f_i$. From this if follows that

$$\Pr\left[(f_1(D_1), \ldots, f_n(D_1)) \in S\right] = \prod_{i=1}^{n} \Pr\left[f_i(D_1) \in S_i | f_1(D_1), \ldots, f_{i-1}(D_1)\right] \tag{5.24}$$

Since for each of these conditional probabilities the results of the previous queries are given, the probability is only over the amount of noise added to the real result of f_i, which has a Laplace distribution. The true result of query f_i will be written as r_i. Recalling that we write $h(x)$ for the probability density function of the Laplace distribution $Lap(b)$, we have

$$\begin{aligned}
\frac{\prod_{i=1}^{n} h(||f_i(D_1) - r_i||_1)}{\prod_{i=1}^{n} h(||f_i(D_2) - r_i||_1)} &= \prod_{i=1}^{n} \frac{\exp\left(-\frac{1}{b}||f_i(D_1) - r_i||_1\right)}{\exp\left(-\frac{1}{b}||f_i(D_2) - r_i||_1\right)} \\
&= \prod_{i=1}^{n} \exp\left(\frac{1}{b}(||f_i(D_2) - r_i||_1 - ||f_i(D_1) - r_i||_1)\right) \\
&\leq \prod_{i=1}^{n} \exp\left(\frac{1}{b}||f_i(D_2) - f_i(D_1)||_1\right) \\
&= \exp\left(\frac{1}{b}\left(||f_1(D_2) - f_1(D_1)||_1 + \ldots + ||f_n(D_2) - f_n(D_1)||_1\right)\right) \\
&\leq \exp\left(\frac{\delta}{b}\right)
\end{aligned}$$

The first inequality is based on the triangle inequality. The second inequality follows from the definition of δ. Filling in δ/ϵ for b and integrating over all S_i with respect to every result r_i finishes our proof. □

In practice it can be difficult to calculate the true value of δ, as it's difficult to mathematically model all the possible interactions between the collection of adaptive queries that are being posed. The following inequality is used to avoid this problem

$$\delta = \max(\Delta f_1 + \ldots + \Delta f_n) \leq \max(\Delta f_1) + \ldots + \max(\Delta f_n) \tag{5.25}$$

So instead of calculating the global sensitivity of the set of adaptive queries, we calculate the sum of the sensitivity of each individual query. Whilst this may result in a larger value of δ, it's much easier to calculate. An important remark is that overestimating δ will result in a Laplace distribution with a higher value for its scale parameter. This in turn results in a higher "amount of noise", meaning the privacy of individuals is actually better protected if we happen to overestimate δ.

5.5.4 Histogram Queries

A compelling example of Theorem 5.2 are histogram queries. A histogram query is a function that assigns each row or element to one specific bucket, and returns the number of elements in each bucket. More formally, let the rows of the database be elements in the universe $\mathcal{D}$ and $\mathcal{D}_1, \ldots, \mathcal{D}_d$ be disjoint regions of $\mathcal{D}$, then the function $f : \mathcal{D} \to \mathbb{N}^d$ that returns the number of elements in each region is called a histogram query. As a concrete example the database may contain the ages of each individual, and the query f is a partitioning of the ages in to the ranges $\{[0, 10), [10, 20], \ldots, [90, 100), \geq 100\}$. In this example the function that would partition the database into disjoint regions is "hardcoded" into the query. A consequence is that a user would have to create a separate differentially private query for each possible partitioning, which is inconvenient. Instead we will require that the partitioning function is given as an argument to the histogram query, and that this query will calculate a result that is differentially private for the given value of epsilon. Hence the user only has to specify the partitioning function, and the rest is handled by the histogram query itself.

At first sight a histogram query may look like d independent counting queries, meaning one has to add noise distributed according to $Lap(d\Delta f/\epsilon) = Lap(d/\epsilon)$. This would significantly disturb the result when the buckets outnumber the rows in the database, and render the output meaningless. However, the entire query actually has sensitivity $\Delta f = 1$. To see this, note that if we remove one row from the database, then only one bucket in the histogram changes, and that bucket only changes by one.

Thus to achieve ϵ-differential privacy we must add independent random noise drawn from $Lap(1/\epsilon)$ to each coordinate of f (recall that f returns a vector). Most importantly, the noise added is independent of the number of buckets d. When comparing this to older methods such as the SuLQ Framework, where the noise added to each coordinate is $O(\sqrt{d}/\epsilon)$, this is a clear improvement [BDMN05, DMNS06].

5.6 K-Means Clustering

To illustrate the practical importance of adaptive queries, we will construct a privacy preserving alternative of the K-means clustering algorithm. This will be done by using existing queries that were shown to be differentially private. In other words we're going to use these existing queries as an "Application Programming Interface" to show that powerful algorithms can be constructed using only relatively simple differentially private queries.

5.6.1 Original Algorithm

Clustering algorithms attempt to divide data into several groups (clusters) that are meaningful, useful, or both. It has long played an important role in a wide variety of fields, and there have been many applications of cluster analysis to practical problems [TSK06].

The original K-means algorithm is relatively simple. First K initial centroids are randomly chosen, where K is a user-specified parameter signifying the desired number of clusters. Next each point is assigned to the closest centroid. Then all the points assigned to a specific centroid are taken together and are considered to be a cluster. The centroid of each cluster is calculated based upon these points assigned to the cluster. This is repeated until no more changes happen. The formal description is shown in Algorithm 5.1.

Algorithm 5.1 Generic K-means algorithm

```
1: Select $k$ points as initial centroids
2: **repeat**
3:     Form $K$ clusters by assigning each point to its closest centroid.
4:     Recompute the centroid of each cluster.
5: **until** Centroids do not change.
```

5.6.2 Private Algorithm

To construct a differentially private version we first make a few changes to the algorithm itself. While the original algorithm allowed a very abstract notion of "points", we will restrict ourselves to points in the d-dimensional unit cube $[0,1]^d$. The set of input points from this unit cube will be denoted by P. The resulting procedure is now described in Algorithm 5.2.

In the first for loop the points are partitioned into k clusters where each point is associated to the closest centroid μ_j. The second for loop calculates the new centroid of each cluster. This is repeated until a termination condition has been reached. Examples of termination conditions are that the centroids no longer change, or that a maximum number of allowed iterations has been reached.

Algorithm 5.2 Unit Cube K-means algorithm

```
 1: Initial centroids $\mu_1, \dots, \mu_k$ are chosen randomly from $[0,1]^d$.
 2: repeat
 3:     for $j = 1$ to $k$ do
 4:         $S_j := \{p \in P \mid \forall i \colon ||\mu_j - p|| \leq ||\mu_i - p||\}$
 5:     end for
 6:     for $j = 1$ to $k$ do
 7:         $\mu_j := \sum_{p \in S_j} p/|S_j|$
 8:     end for
 9: until Termination criterion reached.
10: return $\mu_1, \mu_2, \dots, \mu_k$
```

When attempting to construct a differentially private algorithm, step 3 of the algorithm would breach privacy if we needed to know the actual sets $S_1, \dots, S_k$. However, to compute the average among the points, knowing the size and sum of each set of points is already enough. Therefore the actual implementation of the algorithm doesn't need to expose these sets. The conclusion is that we can focus on step 5 and try to find a method to calculate the new value of μ_j while assuring differential privacy.

The first observation is that the denominators $|S_j|$ implicitly define a histogram query over all points. Therefore we will pose one single histogram query that returns a vector $(|S_1|, \dots, |S_j|)$ containing the size of each set. The partitioning function given as an argument to the histogram query separates the points into k regions as specified in the first for loop (i.e., the partitioning is based on the *current* values of μ_i and these represent the clusters). Posing a histogram query has the advantage that very low noise is added in each coordinate. Namely, the query has a sensitivity of only one. For the numerators we see that the sum of all the points in a set S_j has sensitivity at most d. This is because the points come from the unit cube $[0,1]^d$, and so the removal or addition of a point can affect the sum by at most $||(1, \dots, 1)|| = d$. More importantly, only one of the sums can be affected. For this reason we will use a variation of the histogram query where we still partition the points, but then return the *sum* of all the points instead of the number of points in a centroid. Thus the query will return a vector with the sum of the points for all the sets S_j, and it's sensitivity is at most d.

Each loop the query sequence has a total sensitivity of at most $d+1$. When running the algorithm for a fixed number of N iterations we can add noise distributed by $Lap((d+1)N/\epsilon)$ to achieve ϵ-differential privacy, which follows from theorem 5.3 about adaptive queries. This demonstrates the utility of the theorem: When combining multiple (possibly different) differentially private queries to construct a more complex algorithm, it specifies how much noise must be added to obtain ϵ-differential privacy. In case one doesn't know in advance how many iterations are needed, it's possible to increase the noise parameter as the computation proceeds. A possible strategy would be to start by adding noise distributed by $Lap((d+1)/(\epsilon/2))$ in the first iteration, $Lap((d+1)/(\epsilon/4))$ in the second iteration, and so on. In other words, each time half of the remaining "privacy budget" is being used. It can be proven this indeed assures ϵ-differential privacy by adjusting the proof of theorem 5.3.

5.7 Median Mechanism

The question can be asked if we can do better than independently adding noise to the result of every query. If we can design a mechanism were we need to add less noise, more queries can be answered while maintaining the same accuracy and privacy constraints. One strategy, called the median mechanism, can answer exponentially more queries than the previously discussed mechanism where we added independent Laplace distributed noise to each query result. We will briefly describe the idea behind this strategy.

The median mechanism outperforms the Laplace mechanism by exploiting correlations between different queries. It does this by storing the set of databases that are consistent with the answers given thus far. A query whose answer reduces this set by a constant factor (or more) is considered a "hard" query. All the other queries are classified as "easy". A key intuition here is that if a query is easy, then the user can actually generate the mechanism's answer on its own. These easy questions can be answered by adding very little noise, whilst hard queries are answered by adding Laplace distributed noise. Roth and Roughgarden showed in their paper that only a fraction of the possible queries are be hard [RR10]. The result is that most queries are considered easy and thus more queries can be answers while maintaining privacy and using only a fixed amount of noise.

5.8 Disadvantages

Although ϵ-differential privacy provides a strong privacy guarantee and has many practical implementations, it is not always applicable.

For example, if one wants to publish vague information about individuals in a database, differential privacy cannot be used. The reason is that it simply prohibits releasing *any* information about specific individuals. So say you want to calculate the Body Mass Index of every individual for medical research, and although it has been agreed that this doesn't violate privacy, differential privacy still can't be used. Its privacy guarantees are too strong for such an application.

Another disadvantage is that because differential privacy takes place in an interactive mode, the original database must be preserved while queries are being posed. If only one entity wants to query the database this is not a true problem, as we can simply delete the database once the queries have been executed. However, if one wants to remain the ability to pose queries in the future, the original database cannot be destroyed.

Part II

Privacy in New Settings

Chapter 6

Extensions of Differential Privacy

In the previous chapter we have shown how to assure differential privacy for relatively simple queries. By composing these simple queries one is able to build complex differentially private algorithms. In this chapter we will study new applications of differential privacy. In particular we will see how to handle cases where adding nose makes no sense and we consider the case where one wants to assure differential privacy when the internal state of the algorithm can be leaked. This is based on the work of McSherry and Talwar [MT07] and the work of Dwork, Naor, Pitassi, Rothblum and Yekhanin [DNP$^+$10].

6.1 When Noise Makes No Sense

Up until now our focus has been on functions that return a number or a vector of numbers. Moreover we have assumed that adding noise to the true output makes sense. This is not always the case. When the output of our query is non-numeric it's easy to comprehend that adding Laplacian distributed noise doesn't make sense. For example, if the query function computes a string, tree, or any other complex object, it's unclear how we can add noise in a meaningful way. Even when the output is numeric it's still not always the best choice to simply add Laplacian noise.

6.1.1 An Example

Assume we have a database containing points in $\mathbb{R}^d$ where each point has the label `A` or `B`. We want to make a simple classifier by dividing the d-dimensional space into two separate spaces. This is done by finding a d-ary vector describing a $(d-1)$-dimensional hyperplane that splits the space in such a way that:

- Points labeled with `A` have a nonnegative inner product with y.

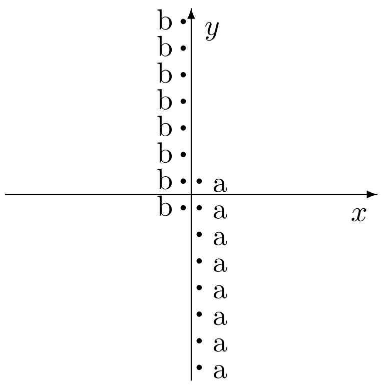


Figure 6.1: Example database of points in $\mathbb{R}^2$ having either label `A` or `B`.

- Points labeled with `B` have a negative inner product with y.

The returned vector is the one which maximizes the number of points classified correctly.

An example database for points in the space $\mathbb{R}^2$ is shown in figure 6.1.1. The optimal 1-dimensional hyperplane—which is just a line—classifying these points would be the line coinciding with the y-axis. This hyperplane is defined by the vector $(1, 0)$. Adding noise to each coordinate of the vector would change its direction, causing some points to be classified incorrectly. We notice that, if more than 3 points are misclassified by adding noise, the vector coinciding with the x-axis is a better candidate. After all, slightly changing its direction will have a lesser influence compared to slightly changing the direction of the vector coinciding with the y-axis. Surprisingly this vector is orthogonal to the optimal result, yet appears to be a better choice when noise is being added. We can capture such preferences by defining a *scoring function.*

6.1.2 The Scoring Function

We can specify the preference of results by defining a *scoring function.* It can be defined for any privacy mechanism that randomly maps a database $D \in \mathcal{D}$ to an output object $r \in \mathcal{R}$ without making any specific assumptions about the domains $\mathcal{D}$ and $\mathcal{R}$. In essence the scoring function gives a score to each possible output y for every database D.

Definition 6.1 (Scoring Function)**.** *Given the domain of databases $\mathcal{D}$ and the domain of possible results $\mathcal{R}$, the scoring function q is defined as*

$$q\colon \mathcal{D} \times \mathcal{R} \to \mathbb{R} \tag{6.1}$$

and assigns a score to any pair $(D, r) \in \mathcal{D} \times \mathcal{R}$. Higher scores denote a more appealing result.

The idea is now that, given a database D, the privacy mechanism returns an $r \in \mathcal{R}$ that maximizes the score $q(D, r)$ while guaranteeing differential privacy. In our example we can give preference to vectors that correctly classify the most points, i.e., the score of a result is the number of points classified correctly. We can define this particular scoring function as follows

$$\begin{aligned} q(D, y) = &|\{p \in D \mid L(p) = \mathrm{A} \wedge py \geq 0\}| \\ &+ |\{p \in D \mid L(p) = \mathrm{B} \wedge py < 0\}| \end{aligned}$$

where $L(p)$ denotes the label of point p. The first term counts the number of correctly classified points with label `A` and the second term the points with label `B`. An important remark is that although a score for a specific result can be zero or negative, this doesn't guarentee such a result will never be returned. It merely means that it's an output we don't prefer.

6.1.3 The Privacy Mechanism

We can now create a differentially private algorithm—called the exponential mechanism [MT07]—where noise is introduced in a more meaningful way by utilising the scoring function. But first we must change the definition of L1 sensitivity to be applicable for the scoring function. In order to be precise the definition to take into account the fact that the function has two parameters.

Definition 6.2. *The sensitivity of the scoring function $q \colon \mathcal{D} \times \mathcal{R} \to \mathbb{R}$ is*

$$\Delta q = \max_{r \in \mathcal{R}} \max_{D_1, D_2} |q(D_1, r) - q(D_2, r)|$$

for all adjacent databases $D_1, D_2 \in \mathcal{D}$.

The definition of the exponential mechanism now becomes

Definition 6.3 (Exponential Mechanism)**.** *An exponential mechanism is a randomized function $\mathcal{E} \colon \mathcal{D} \to \mathcal{R}$ that, when given the privacy parameter ϵ and database instance $D \in \mathcal{D}$, outputs a result $r \in \mathcal{R}$ according to the probability density function*

$$f_{\mathcal{E},D}(r) \propto \exp\left(\frac{\epsilon}{2\Delta q} q(D, r)\right) \tag{6.2}$$

where q can be any scoring function and Δq is the sensitivity of q as defined in 6.2.

Notice that in this definition the probability density function is over a generic set $\mathcal{R}$. While we have only given the formal definition of the pdf over the set $\mathbb{R}$ in section 2.1.3 on continuous variables, this is not a problem. It's possible to give definitions where the image of the random variable can be any generic set as long as the set is *measurable.* However these definitions are out of scope for this work, and the general idea behind them remains the same. The precise definition of $f_{\mathcal{E},D}$ such that it satisfies equation 2.9 on page 23 is

$$f_{\mathcal{E},D}(r) = \frac{\exp\left(\frac{\epsilon}{2\Delta q} q(D,r)\right)}{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D,r)\right) \mathrm{d}r} \tag{6.3}$$

We see that the mechanism assigns the highest probability to the best answer, and the probability assigned to any other answer decreases exponentially for lower scores given the current database, hence the name "exponential mechanism" [Dwo11]. We will prove this mechanism indeed assures differential privacy.

Theorem 6.1 (Differential Privacy)**.** *An exponential mechanism as defined in 6.3 gives ϵ-differential privacy.*

Proof. Given an exponential mechanism $\mathcal{E}$ where q is the scoring function, take $S \subseteq \mathcal{R}$ and $s \in S$. We get that for adjacent databases $D_1, D_2 \in \mathcal{D}$:

$$\frac{f_{\mathcal{E},D_1}(s)}{f_{\mathcal{E},D_2}(s)} = \frac{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D_2,r)\right) \mathrm{d}r}{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D_1,r)\right) \mathrm{d}r} \cdot \frac{\exp\left(\frac{\epsilon}{2\Delta q} q(D_1,s)\right)}{\exp\left(\frac{\epsilon}{2\Delta q} q(D_2,s)\right)} \tag{6.4}$$

The first quotient becomes

$$\begin{aligned}
\frac{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D_2,r)\right) \mathrm{d}r}{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D_1,r)\right) \mathrm{d}r} &\leq \frac{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} (q(D_1,r) + \Delta q)\right) \mathrm{d}r}{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D_1,r)\right) \mathrm{d}r} && (6.5)\\
&= \frac{\exp\left(\frac{\epsilon}{2}\right) \int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} (q(D_1,r))\right) \mathrm{d}r}{\int_{\mathcal{R}} \exp\left(\frac{\epsilon}{2\Delta q} q(D_1,r)\right) \mathrm{d}r} && (6.6)\\
&= \exp\left(\frac{\epsilon}{2}\right) && (6.7)
\end{aligned}$$

And for the second quotient one has

$$\begin{aligned}
\frac{\exp\left(\frac{\epsilon}{2\Delta q} q(D_1,s)\right)}{\exp\left(\frac{\epsilon}{2\Delta q} q(D_2,s)\right)} &= \exp\left(\frac{\epsilon}{2\Delta q} \left(q(D_1,s) - q(D_2,s)\right)\right) && (6.8)\\
&\leq \exp\left(\frac{\epsilon}{2\Delta q} \Delta q\right) && (6.9)\\
&= \exp\left(\frac{\epsilon}{2}\right) && (6.10)
\end{aligned}$$

For both quotients the inequalities hold because Δq is defined as the largest possible difference and is always positive. Combined this gives

$$\frac{f_{\mathcal{E},D_1}(s)}{f_{\mathcal{E},D_2}(s)} \leq \exp(\epsilon) \tag{6.11}$$

Integrating over S with respect to s yields the desired result. $\square$

6.1.4 Limitations and Future Directions

A problem that hasn't been addressed yet in this work is the issue of efficiently sampling from the distribution $f_{\mathcal{E},D}$ (defined in 6.3). This is needed when we actually want to implement the privacy mechanism and return a random value according to the given probability density function. The work of McSherry and Talwer [MT07] that introduced the exponential mechanism also leave this problem largely unanswered. They mentioned that if the scoring function q is simple, e.g., when it's piecewise linear or in the cases where the output space can be efficiently discretized, an efficient algorithm can be constructed [MT07]. Unfortunately for more complex scoring functions there are currently no efficient methods to sample from $f_{\mathcal{E},D}$.

Another problem is that computing the score $q(d,r)$ can be a computationally demanding task. If there are no efficient algorithms to calculate the score of a result, the exponential mechanism is also infeasible in practice. In such a situation we would like to use approximation algorithms to calculate the score $q(d,r)$. However these approximation algorithms ought to have a small sensitivity Δq, otherwise too much noise would have to be added. Unfortunately most approximation algorithms do not seem to have this property [MT07].

6.2 Pan Privacy

There are situations where a data curator is pressured to handover all the data it has collected or is still processing. This can be caused by a legal compulsion such as a subpoena[1]. To protect privacy in such a setting we will develop algorithms that retain their privacy properties even if their internal state becomes visible to an adversary.

The prefix *pan* comes from the greek πᾶν meaning "all" and "involving all members". So pan privacy signifies privacy of all parts, in this case even the internal state of the algorithm.

[1]Informally a *subpoena* in such a situation can be seen as a command to a witness to produce certain documents for the court.

6.2.1 Introduction

We will focus on *streaming algorithms* where the data is arriving element by element and therefore it's natural to discard the data that has already been processed. This is not a requirement per se, as batch algorithms working on a complete database may also be pan private. A batch algorithm could for example protect against an intruder that cannot directly access the data because it's stored on a different server [DNP$^+$10]. Pan privacy will *not* assume the existence of a private—possible small—internal memory state that can only be accessed by the analyst. Such an assumption would after all conflict with the goal of making the algorithm retain privacy when the internal state has been leaked. This means that we can't even store the last few records that have been analyzed, otherwise the privacy of the individuals corresponding to those last few records that are still in memory would be violated. It may seem that without such a secret state one cannot calculate accurate yet private results. However that is not the case, as we shall see many properties of a stream can be calculated whilst satisfying pan privacy within certain accuracy bounds.

The privacy guarantees will be based on differential privacy. For this to be possible we will first define when two data streams are adjacent. We can choose between two possible definitions that differ in their level of granularity. The first definition would hide all the elements in the stream of a user (user level pan privacy), whilst the second definition hides the existence of individual events (event level pan privacy).

6.2.2 Preliminary Definitions

As already mentioned we will study streaming algorithms where the input is a data stream. We assume the incoming data stream is of unbounded length and composed of elements in the universe X. As a motivating example you can imagine that each element describes a visit of a user—represented by his or her IP address—to a particular website [DNP$^+$10]. Pan privacy will then protect against the presence of absence of an IP address in the stream.

The definition of differential privacy is based on adjacent databases and is not directly applicable for streaming algorithms. We begin by defining what a data stream is and when two data streams are considered adjacent.

Data Streams

The precise definition of a data stream is

Definition 6.4 (Data Stream)**.** *A data stream over the universe X is a sequence of elements $x \in X$ that is possibly unbounded. A more sophisticated way of representing the abstract sequence is with the notation*

$$\{x_i\}_{i=1}^n$$

where n is the length of the sequence and $\forall i\colon x_i \in X$. If the sequence is unbounded we drop the superscript n from the notation.

From this definition we learn that a data stream can have an infinite number of elements. There are many occasions where we want to talk about data streams with a bounded number of elements. Such bounded data streams are frequently called *data stream prefixes* in other works, and we will also adopt this naming convention.

Definition 6.5 (Data Stream Prefix)**.** *A data stream prefix is a data stream of bounded length.*

Based on the definition of data streams there are now two general possibilities to interpret adjacent data streams. The first option, that would assure *event level pan privacy*, considers the presence or absence of only one single element $x \in X$. This means that any single event or action caused by a particular individual remains private, but the presence of absence of the individual herself is not protected. The second and more powerful definition, which will assure *user level pan privacy*, considers the presence or absence of *all* elements equal to any $x \in X$. Here the presence or absence of an individual is protected: Based on the output of the algorithm one cannot deduce whether the data stream contained information about a particular individual or not. In this chapter our focus will be on user level privacy. Hence we arrive at the following definition of adjacent data streams

Definition 6.6 (X-adjacent Data Streams [DNP$^+$10])**.** *The data streams S and T over universe X are X-adjacent if for all $x \in X$ the subsequences S' and T', obtained by removing all occurrences of the element x from respectively S and T, are identical.*

In other words data streams S and T are X-adjacent if they differ only in the presence or absence of *any number* of occurrences of a single element $x \in X$. If all occurrences of x are deleted from both streams, then the resulting subsequences ought to be identical. This definition permits strings of radically different lengths to be adjacent, which is seen as a positive property, as it provides the most general privacy guarantees. A relaxed definition of adjacent data streams—that is in line with user level pan privacy—has been proposed in the paper by Dwork, Naor, Pitassi and Rothblum [DNPR10]. In their work data streams S and S' are adjacent if there exist $x, x' \in X$ such that when changing some of the instances of x in S to instances of x' one obtains S'. In this work we will focus on definition 6.6.

Another common operation performed on data stream prefixes is concatenation. It's a straightforward operation but in order to avoid any ambiguity we will provide a definition of it.

Definition 6.7 (Data Stream Concatenation)**.** *Given the data streams prefixes $S = \{s_i\}_{i=1}^{n}$ and $T = \{t_i\}_{i=1}^{m}$ their concatenation, denoted by $S \circ T$, is defined as $\{c_i\}_{i=1}^{n+m}$ where*

$$c_i = \begin{cases} s_i & \text{if } 1 \leq i \leq n \\ t_{i-n} & \text{else} \end{cases} \tag{6.12}$$

We observe that the concatenation of two data stream prefixes is again a data stream prefix, i.e., the length of the result is always bounded.

6.2.3 Definition of Pan Privacy

We will model a randomized streaming algorithm using two randomized functions:

- **State**: This randomized function is called the *state mapping function* and maps data stream prefixes to an internal state of the algorithm. The returned state is the one the algorithm is in immediately after processing the prefix. For example, if the algorithm is a Turing machine, the state of the randomized function is the contents of its tape and the current state the Turing machine is in. We will abbreviate the function using the symbol $\mathfrak{S}$.
- **Out**: The randomized function **Out** is called the *output mapping function* and maps data stream prefixes to an output. This output is the one produced after processing the complete prefix. We will abbreviate the function using the symbol $\mathfrak{O}$.

Definition 6.8 formalizes these assumptions of a randomized streaming algorithm. It is based on the paper by Dwork et al. [DNP+10].

Definition 6.8 (Randomized Streaming Algorithm)**.** *A randomized streaming algorithm $\mathcal{M}$ is represented by a pair randomized functions*

$$\mathfrak{S}\colon \mathcal{S} \to I$$
$$\mathfrak{O}\colon \mathcal{S} \to \mathcal{R}$$

where $\mathcal{S}$ is the set of all possible data stream prefixes, I is the set of all possible internal states of $\mathcal{M}$ and $\mathcal{R}$ the set of all possible outputs.

Without loss of generality we will assume that the output is returned after updating the internal state of the algorithm. Note that the results of the functions $\mathfrak{S}$ and $\mathfrak{O}$ are allowed to be dependent on each other, as is mostly the case. We will use $\mathcal{M}(S)$ as a shorthand notation for $(\mathfrak{S}(S), \mathfrak{O}(S))$. Similarly the notation $\mathcal{M} = (\mathfrak{S}, \mathfrak{O})$ is used to specify the state and output mapping functions respectively. Additionally the algorithm doesn't know the length of the data stream prefix in advance. Conceptually this means it will keep processing input elements until it receives a special stop signal, after which the algorithm will produce an output.

A streaming algorithm can now be thought of as taking steps at discrete time intervals. A new step begins on receiving the next element in the stream, after which the algorithm updates its internal state based on this element. We assume receiving and updating the state happens in an atomic step, so that an adversary isn't able to see a received element when obtaining access to the internal state. In other words, an intrusion may only occur *between* atomic steps [DNPR10].

Pan Privacy

We are finally ready to define pan privacy. Remember that the definition is based on differential privacy.

Definition 6.9 (ϵ-differential Pan Privacy)**.** *Let $\mathcal{M}$ be a randomized streaming algorithm defined by a state mapping $\mathfrak{S}$ and output mapping $\mathfrak{O}$, and with possible internal states I and set of possible outputs $\mathcal{R}$. Then if for all $I' \subseteq I$ and $\mathcal{R}', \mathcal{R}'' \subseteq \mathcal{R}$ and for all X-adjacent data stream prefixes $S = U \circ V$ and $S' = U' \circ V'$ it holds that*

$$\begin{aligned}&\Pr\left[(\mathfrak{S}(U), \mathfrak{O}(U), \mathfrak{O}(S)) \in (I', \mathcal{R}', \mathcal{R}'')\right] \\ &\quad \leq e^{\epsilon} \Pr\left[\left(\mathfrak{S}(U'), \mathfrak{O}(U), \mathfrak{O}(S')\right) \in (I', \mathcal{R}', \mathcal{R}'')\right]\end{aligned}$$

algorithm $\mathcal{M}$ satisfies ϵ-differential pan privacy. The probability spaces are over the internal coin flips of $\mathfrak{S}$ and $\mathfrak{O}$.

Pan privacy thus protects against a single intrusion of its internal state. After the intrusion the algorithm can continue processing data and eventually output a result which still protects the privacy of the individuals. Figure 6.2.3 shows a timeline that illustrates these two events: The intrusion occurs after the prefix U has been processed, and the algorithm produces output once the complete prefix S has been processed.

6.2.4 Density Estimation

As an example application of pan privacy we will design a streaming algorithm that estimates the density of the stream while assuring ϵ differential pan privacy. The density of a data stream is the fraction of items in the universe X that appear at least once. We assume the universe X has finite size.

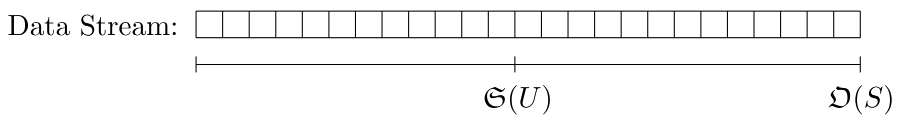


Figure 6.2: Illustration of when a possible intrusion occurs, and afterwards output is still produced. Each block represents an incoming element of the data stream. The intrusion happens after reading prefix U and the output is produced after reading prefix S.

Algorithm Description

The algorithm will have to store which items have already appeared in the stream so far. However this has to be done in a way that doesn't leak too much information, i.e., differential privacy must hold even if the internal state becomes visible to an adversary. The solution to this problem begins by defining two probability distributions $\mathcal{D}_0$ and $\mathcal{D}_1(\epsilon)$ on the set $\{0,1\}$. The distribution $\mathcal{D}_0$ returns both zero and one with probability $\frac{1}{2}$. Distribution $\mathcal{D}_1(\epsilon)$ returns one with probability $\frac{1}{2}+\frac{\epsilon}{4}$ and hence returns zero with probability $\frac{1}{2}-\frac{\epsilon}{4}$. Put differently $\mathcal{D}_1$ has a slight bias towards one whilst $\mathcal{D}_0$ assigns equal mass to both zero and one.

Algorithm 6.1 Density Estimation

```
   Initialization
1: Create a table of size $|X|$ with a single one-bit entry for each item in $X$
2: For every entry $x \in X$ initialize its entry in the table with an independent random draw from $\mathcal{D}_0$

   Processing
3: Upon receiving a value $x \in X$ from the data stream, update $x$'s entry in the table by drawing its new value from $\mathcal{D}_1(\epsilon)$

   Output
4: Let $\theta =$ fraction of entries in the table with value 1
5: Output the density value $f' = 4\left(\theta - \frac{1}{2}\right)/\epsilon + Lap(1/(\epsilon|X|))$
```

During the initialization step of the algorithm a table of size $|X|$ is created. Each entry of the table contains one single bit that is set to a draw from the distribution $\mathcal{D}_0$. When receiving an element $x \in X$ from the data stream, the entry corresponding to x is updated by drawing a value from the distribution $\mathcal{D}_1(\epsilon)$. Finally, at the end of the algorithm, the returned output is $4\left(\theta - \frac{1}{2}\right)/\epsilon + Lap(1/(\epsilon|X|))$ where θ is the fraction of entries in the table with the value 1. These steps are detailed in algorithm 6.1.

Step 3 in algorithm 6.1 is assumed to be an atomic step and cannot be interrupted. Therefore an intrusion cannot happen when an element x is *being* processed, hence its value can never be learned by an adversary.

Correctness of Privacy Guarantees

To prove that this algorithms satisfies 2ϵ-differential pan privacy we begin with the following claim.

Claim 6.2. *For $0 \leq \epsilon \leq \frac{3}{2}$ the distributions $\mathcal{D}_0$ and $\mathcal{D}_1(\epsilon)$ are ϵ-differential private. Meaning that for a discrete random variable X_0 with distribution $\mathcal{D}_0$ and X_1 with distribution $\mathcal{D}_1(\epsilon)$, and for all possible outputs $b \in \{0,1\}$ it holds that*

$$\exp(-\epsilon) \leq \frac{\Pr[X_1 = b]}{\Pr[X_0 = b]} \leq \exp(\epsilon)$$

Here we use the formula of differential privacy as explained in section 5.3.2 which is equivalent with the original definition. We will now prove the previous claim.

Proof. Let us begin by proving the statement for $b = 0$. We have that

$$\frac{\Pr[X_1 = 0]}{\Pr[X_0 = 0]} = \frac{\frac{1}{2} + \frac{\epsilon}{4}}{\frac{1}{2}} = 1 + \frac{\epsilon}{2} \tag{6.13}$$

So the inequalities we need to prove become

$$\exp(-\epsilon) \leq 1 + \frac{\epsilon}{2} \leq \exp(\epsilon) \tag{6.14}$$

For $\epsilon = 0$ the inequalities are true. Further observe that $\exp(-\epsilon)$ is a decreasing function and $1 + \frac{\epsilon}{2}$ is an creasing function. This proves that the first inequality is true for $\epsilon > 0$. To prove the second inequality we know that the function

$$\exp(\epsilon) - 1 - \frac{\epsilon}{2} \tag{6.15}$$

is increasing for $\epsilon > 0$ by taking the derivate and showing that it is non-negative. From this we know the second inequality is also true for non-negative ϵ, completing the case for $b = 0$.

The case for $b = 1$ again begins by calculating the fraction of the probabilities, resulting in the inequalities

$$\exp(-\epsilon) \leq 1 - \frac{\epsilon}{2} \leq \exp(\epsilon) \tag{6.16}$$

For non-negative ϵ the second inequality is trivially true as we already proved that $1 + \frac{\epsilon}{2} \leq \exp(\epsilon)$. The first inequality is more tricky and is equivalent to proving

$$f(\epsilon) = 1 - \frac{\epsilon}{2} - \exp(-\epsilon) \geq 0 \tag{6.17}$$

The two zero points of the function f are at 0 and at $1.593\ldots$ and the function is positive between these the points[2]. Given that $\frac{3}{2} < 1.593\ldots$ the proof is complete. □

Utilising this claim we can prove the privacy guarantee of our density estimation algorithm. The proof is based on the one given in [DNP+10].

Claim 6.3. *Algorithm 6.1 provides 2ϵ-differential pan privacy.*

Proof. The first part of the proof shows that the internal state of the algorithm is ϵ-differently private at the time of intrusion. The second part shows that the final output is also differentially private, even for an adversary who has previously seen the internal state during an intrusion.

Part one: Private internal state. For all items $x \in X$ in the table, its value either didn't appear in the data stream and its entry is drawn from $\mathcal{D}_0$, or it did appear in the data stream and its entry is drawn from $\mathcal{D}_1(\epsilon)$. Even if the item appeared multiple times its current entry is still drawn from $\mathcal{D}_1(\epsilon)$. By claim 6.2 these entries all satisfy ϵ-differential privacy. The users in X are thus guaranteed protection against an unannounced intrusion.

Part two: Private output. The query that would count the fraction of 1's the table at the end of the algorithms has sensitivity $1/|X|$. Therefore adding noise sampled from $Lap(1/(\epsilon|X|))$ guarantees ϵ-differential privacy. This is exactly what is done in the algorithm, thus proving the output on its own is ϵ-differentially private.

From the composability of differential privacy we infer that the total algorithm satisfies 2ϵ-differential privacy. This completes the proof. □

Accuracy

All that is left to do is to explain how the return value is calculated and prove that with high probability this result is accurate.

Let f denote the fraction of items in the universe X that appeared at least once. The expected fraction of entries in the table with a value of 1 is then

$$\mathrm{E}[\theta] = (1-f)\cdot\frac{1}{2} + f\cdot\left(\frac{1}{2}+\frac{\epsilon}{4}\right) = \frac{1}{2} + \frac{\epsilon}{4}f \tag{6.18}$$

Solving the equation for f gives

$$f = \frac{\mathrm{E}[\theta] - \frac{1}{2}}{4\epsilon} \tag{6.19}$$

[2]The actual value of the second zero point is $2 + \mathrm{LambertW}(-2\mathrm{e}^{-2})$ but such a calculation is out of scope for this work.

which is exactly what our algorithm is returning when substituting the expected value of θ with it's actual value. It's possible to prove that the result of the density estimation algorithm has good accuracy.

Claim 6.4. *For every $t > 0$ it holds that*

$$\Pr\left[|f - \mathrm{E}[f]| \geq t\right] \leq 2\exp\left(-\frac{|X|\, t^2\epsilon^2}{8}\right)$$

where f is a random variable defined as the output of algorithm 6.1 and $|X|$ the size of universe X.

Proof. This follows from the Hoeffding's inequality on the sum of random variables saved in the table. Let us first restate this inequality (for details see section 2.4.3).

Theorem 6.5 (Hoeffdings' Inequality). *If $X_1, X_2, \ldots, X_n$ are independent random variables with $\forall i\colon a_i \leq X_i \leq b_i$ and with*

$$f = \frac{X_1 + X_2 + \ldots + X_n}{n}$$

then for $t > 0$

$$\Pr\left[|f - \mathrm{E}[f]| \geq t\right] \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{6.20}$$

Before we can apply the bound we must rewrite how the random variable f is calculated. Letting $n = |X|$ we get

$$f = \frac{4}{\epsilon}\left(\theta - \frac{1}{2}\right) \tag{6.21}$$

$$= \frac{4}{\epsilon}\left(\frac{X_1 + X_2 + \ldots + X_n}{n} - \frac{1}{2}\right) \tag{6.22}$$

$$\tag{6.23}$$

The X_i denote the independent random variables representing all the entries in the table. We continue transforming this formula until we can apply Hoeffding's inequality.

$$f = \frac{4}{\epsilon}\left(\frac{\left(X_1 - \frac{1}{2}\right) + \left(X_2 - \frac{1}{2}\right) + \ldots + \left(X_n - \frac{1}{2}\right)}{n}\right) \tag{6.24}$$

$$f = \left(\frac{\frac{4}{\epsilon}\left(X_1 - \frac{1}{2}\right) + \frac{4}{\epsilon}\left(X_2 - \frac{1}{2}\right) + \ldots + \frac{4}{\epsilon}\left(X_n - \frac{1}{2}\right)}{n}\right) \tag{6.25}$$

$$f = \left(\frac{X_1' + X_2' + \ldots + X_n'}{n}\right) \tag{6.26}$$

$$\tag{6.27}$$

The last equation defines the new random variables $X_i' = \frac{4}{\epsilon}\left(X_i - \frac{1}{2}\right)$. Knowing that each variable X_i takes values in $\{0, 1\}$ it's straightforward to see that each X_i' takes values in $\{-\frac{2}{\epsilon}, \frac{2}{\epsilon}\}$. The denominator in the exponent of Hoeffding's inequality becomes

$$\sum_{i=1}^{n} \left(-\frac{2}{\epsilon} - \frac{2}{\epsilon}\right)^2 = n\frac{16}{\epsilon^2} \tag{6.28}$$

We can now directly apply Hoeffdings' inequality which results in

$$\Pr\left[|f - \mathrm{E}\left[f\right]| \geq t\right] \leq 2\exp\left(-\frac{nt^2\epsilon^2}{8}\right) \tag{6.29}$$

And since we choose $n = |X|$ the proof is finished. □

Claim 6.4 shows that the probability of a result decreases exponentially as it diverges away from the actual value. This substantially biases the distribution towards accurate results. It also learns us that as the size of the universe X grows linearly, the probability of the result diverging from the actual value decreases exponentially.

Finally we notice that as ϵ becomes larger, the results is more accurate and visa versa. This matches our intuition, since decreasing the "amount of privacy an individual has" increases the accuracy of a result.

Chapter 7

Handling Network Data

During the last decade there has been a growing interest in network data. Here network data is defined as any kind of data which can be meaningfully represented by a graph. In the previous chapters we have focused mainly on tabular data. While technically a graph can be stored in a tabular database, the existence of relationships between entities profoundly alters many aspects of the privacy problem. In short the analysis of network data is more complex and varied than the analyses of traditional tabular data, and involves different privacy risks. In this chapter we will formally introduce the graph model and several of it's properties. Specific privacy problems that arise when working with network data will also be addressed.

7.1 Motivation

The focus will be on social networks and their inherent privacy problems. As a demonstration of the importance of network data we will give a brief overview of several domains producing and analyzing such data. It also shows that both fully identified, and supposedly anonymized network data, are widely available and thus accessible by malicious individuals as well. This overview is partly based on that of Narayanan and Shmatikov [NS09].

Data Mining

There is a multitude of datasets which can be used for research. For example corporations like AT&T possess a database of nearly 2 trillion phone calls going back decades [Hay06], and they have in-house research facilities to perform analysis on these graphs. But smaller operators without such luxury must share their graphs with external researchers, which is not always done due to privacy concerns. Call graphs might be useful for social scientists and have also proven to be valuable to detect illicit activity such as calling fraud [Wil97]. Another purpose is improving national security, where communication graphs are being studied to determine if command-and-control structures of terrorist cells have a unique structure that can be automatically detected [Hay06].

Health-care professionals, epidemiologists and sociologists also collect various information about geographic, friendship, family and sexual networks to study disease propagation and risk. Potterat et al. [PPPM$^+$02] published a social network which shows a set of individuals related by sexual contacts and shared drug injection. Researchers had to weigh the benefit of public analysis against possible losses of privacy to the individuals involved, without clear knowledge of potential attacks. In another study the National Longitudinal Study of Adolescent Health (Add Health) collected the sexual-relationship network of students of an anonymous Midestern high school. The graph resulting from the Add Health study has been published in an anonymized form [BMS04].

Online Social Networks

Another common source are online social networks. Such data can be collected using a crawler that automatically visits public user profiles. Researchers on graph anonymization algorithms usually use these data sets to test the performance of their algorithms, or to show the feasibility of certain attacks [NSR11, BDK07, LT08]. Although this information is already publicly available, the generation of such graphs benefit adversaries who might not have the capabilities to perform large and automated crawling of these networks.

Advertising on online social networks is, most of the time, enhanced by using the information of the social graph. There is empirical evidence that using social network data increases the profit of advertising. As a result network operators are inclined to share the social graph with advertising partners. If we look at the facebook privacy policy we conclude that the complete social graph isn't handed over the advertiser, only information of users that give permission to hand over the data are being shared, which however might still be a large proportion of the users. The advertiser can specify to whom certain advertisements are shown, for example it's possible to target a category of user like a "moviegoer" or a "sci-fi fan" [Fac], but most of the data mining appears to be done by facebook itself. On the other hand—based on the testimony of Chris Kelly, the chief privacy officer at facebook—non-personally identifying information might be shared with advertisers [Uni08]. This raises the question if the shared data truly preserves privacy or actually resembles a form of naive anonymization, which doesn't assure privacy for all users.

Other Sources and Advantages

In network security it is often essential to get a global overview of the network if one wants to detect certain attacks. In particular it's difficult to defend against global phenomena such as botnets when operators can observe only the traffic in the local domains [RAW$^+$10]. Understandably network operators are hesitant in sharing information about the traffic in their networks. Mechanism that are able to create a global overview of the network while assuring privacy could provide a step in the right direction to solve these problems.

Another domain that can benefit from network data is that of face recognition, by exploiting the fact that users who appear together in photographs are likely to be neighbours in the social network[SZD08]. Since a large number of photos are published on social networks (e.g. flickr and facebook), having access to the anonymized graphs can greatly improve large-scale facial re-identification.

7.2 Graph Model

Evidently we will model network data by a graph $G = (V, E)$ where V is the set of nodes and E is a set of edges between the nodes.

It would also have been possible to save the nodes and edges in a tabular database and then apply the previous anonymization techniques such as k-anonymity. However such a mechanism would ruin the utility of the underlying network. In particular, if viewed as tabular database queries, many graph analysis require the use of joins on the edges of the graph, e.g., when calculating the diameter or transitivity of a graph. So the first disadvantage is that tabular anonymization doesn't work well if later on you want to use analysis which require many joins.

Another problem is that, if we look at k-anonymity for example, it would require that there are at least k entries representing "similar" edges (i.e., edges that are indistinguishable to the adversary), otherwise it wouldn't be indistinguishable with k other entries. Admittedly this depends on what you consider sensitive attributes when applying k-anonymization, but as we shall see the existence of an edge poses privacy risks. Such a modification to the network data would completely destroy any utility it had before.

7.2.1 Assuring Privacy

As always there are multiple ways one can retain privacy. For network data there are three main approaches if the goal is to release data while retaining privacy. These are depicted in figure 7.1. The first option is to release a so called naively anonymized graph, where identifiers are removed and attributes are coarsened or not released at all. The weaknesses of this

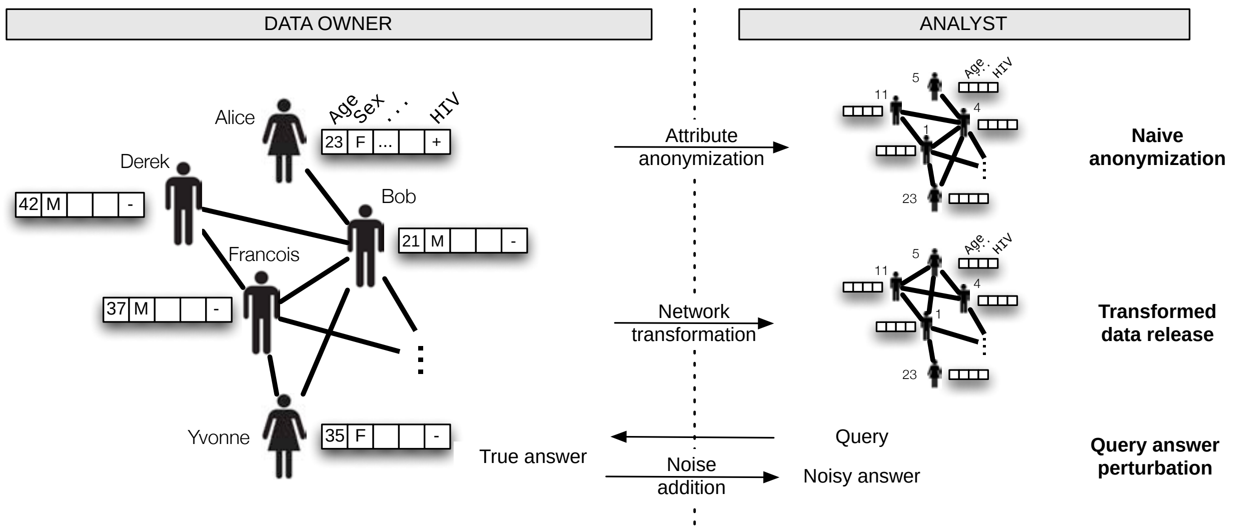


Figure 7.1: Illustration of the three possible approaches to protecting privacy in graph data. [Hay10]

paradigm are discussed in chapter 8. The second option is to transform the network so the actual structure of the graph is altered. Chapter 9 will show one method that can be used, but unfortunately doesn't retain privacy under all situations. Finally it's possible to withhold the network data to the analyst, and only allow him or her to pose queries and add noise to the result in order to retain privacy.

7.2.2 Risk Assessment

Let's consider the basic case where only the graph is released and all attributes have been removed. In effect every node has been given a completely random identifier, which on itself doesn't disclose any private information. The question is now what the privacy risk of releasing this graph would be.

The answer depends of course on the network data the graph represents. In some cases the privacy risk could be low, but in other cases there could be a high risk. Here we will focus on worst-case scenarios. Our example will be the the "HIV graph" of Colorado Springs where nodes correspond to individuals and the edges stand for sexual relationships or shared drug injection [PPPM$^+$02]. Such data contains sensitive information and can have a high privacy risk. We will discuss the two most important possible privacy violations when only the structure of the graph is released.

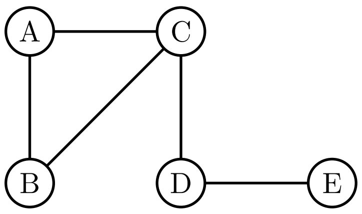


Figure 7.2: Preventing identity disclosure doesn't prevent edge disclosure. If an adversary knows an individual corresponds to either A, B or D, his or her identity is protected. However all three nodes have an edge to C, hence the existence of the edge is *not* protected.

Identity Disclosure

A possible attack is that an adversary learns the location of an individual in the released graph. This on itself could lead to a privacy violation. For example, assume that only people having HIV are included in the graph, then clearly identifying an individual in the graph means the adversary has now learned that he or she has HIV.

Identity disclosure inevitably also leads to degree disclosure. That is, when learning the location of an individual, an adversary will also learn the degree of the corresponding node. This can also have privacy implications. For example, the degree of a node could signify how many sexual relationships an individual has.

Edge Disclosure

Another risk is that of edge disclosure, where the adversary learns of the existence of an edge between two individuals. Again taking the HIV graph as an example, a malicious individual should not be able to learn whether two individual are in a sexual relationship or not.

Remark that if we prevent identity disclosure, this doesn't necessarily prevent edge disclosure. Figure 7.2 illustrates this point. Say the adversary wants to determine if there is an edge between Alice and Bob. He or she knows that Alice has degree 2 and that Bob has degree 3. Based on this information alone Alice can be node A, B or D, and Bob must be node C. In other words the identity of Alice is protected, as the adversary doesn't know the corresponding node. However notice that all three nodes have an edge to node C. So even though the adversary doesn't know the exact location of Alice, he or she does learn the existence of the edge to Bob.

Power of the Adversary

Our result of chapter 4 also applies for network data. That is, whenever useful information is released, there is a possibility that the privacy of an individual is compromised. This means that if we decide to release a—possibly modified—version of the graph, there will always remain a risk that the privacy of someone can be violated. So we must focus on *reducing* the chance of a privacy violation.

Commonly first the power of the adversary is specified in terms of access to auxiliary information and computational power, and then a privacy mechanism is designed to protect against such adversaries. A flexible model to describe how much information an adversary has access to is presented in the paper by Hay et al. [HMJ$^+$08].

7.3 Graph Theory

Graphs are a natural way to represent network data. We will give a brief overview of some basic graph definitions and properties that will be used in forthcoming chapters.

7.3.1 Basic Definitions

Evidently we begin with the definition of a graph.

Definition 7.1 (Graph)**.** *A graph is an ordered pair $G = (V, E)$ where V is a set called the nodes, and $E \subseteq V \times V$ is a set called the edges.*

Commonly we will also talk about *simple graphs.* These are graphs satisfying the following three constraints:

1. They have no loops, i.e., they have no edges starting and ending at the same node.
2. They are undirected, meaning if it has an edge (v_1, v_2) it must also contain the edge (v_2, v_2).
3. And finally they must be unweighted, which means edges are not mapped to a number.

Two graphs can have the same structure but their nodes can have different labels. This is formalized by the definition of an isomorphism.

Definition 7.2 (Isomorphism)**.** *Graphs $G = (V_g, E_g)$ and $H = (V_h, E_h)$ are isomorphic if there exists a bijection f between the set of nodes V_g and V_h such that*

$$\forall u, v \in V_g \colon (u, v) \in E_f \iff (f(u), f(v)) \in E_h$$

An isomorphism can be seen as a relabeling of the nodes of a graph so that both graphs become identical. An interesting property of an isomorphism is a *fixed point*, which will mainly be used in the proofs given in chapter 8.

Definition 7.3 (Fixed Point)**.** *A node v is a fixed point of an isomorphism $f\colon V \to V'$ if $v \in S \cap S'$ and $f(s) = s$. V and V' are the set of nodes of both graphs.*

Another common definition is that of a path. This is based on the definition of *incident* nodes and edges, and on the definition of a *walk*.

Definition 7.4 (Incident)**.** *Given a graph $G = (V, E)$, the nodes $u, v \in V$ are incident to the edge $e = \{u, v\}$. Similarly e is incident to both u and v.*

Definition 7.5 (Walk)**.** *A walk on a graph is an alternating sequence of nodes and edges, beginning and ending with a node, in which each edge is incident with the node immediately preceding it and the node immediately following it.*

Based on these definitions we can give the definition of a path.

Definition 7.6 (Path)**.** *A path is a walk in which all edges and nodes are distinct. As an exception the first and last nodes are allowed to be identical, and if this is the case we talk about a* closed *path, otherwise it's an* open *path.*

Often we will want to talk not about all the nodes and edges of a graph, but only about a selected region of the graph. To do this will we will use the notion of a subgraph.

Definition 7.7 (Subgraph)**.** *A graph $H = (V_h, E_h)$ is called a subgraph of $G = (V, E)$ if $V_h \subseteq V$ and $E_h \subseteq E$ such that $E_h \subseteq V_h \times V_h$.*

Additionally we will call G a *supergraph* of H if and only if H is a subgraph of G. Another interesting concept in the complement of a graph.

Definition 7.8 (Complement of a Graph)**.** *The complement of a simple graph $G = (V, E)$ is the simple graph $\overline{G} = (V, \overline{E})$ where*

$$\overline{E} = \{v_1, v_2 \in V \mid v_1 \neq v_2 \wedge (v_1, v_2) \notin E\}$$

7.3.2 Graph Properties

When evaluating the impact of anonymization on graph, we will analyze how certain properties of the graph change. For this reason we will first introduce several commonly used and important properties.

Density

The density of a graph is defined as the number of edges divided by the number of possible edges. Thus for a graph $G = (V, E)$ it is equal to

$$\text{density} = \frac{|E|}{\binom{|N|}{2}} = \frac{|E|}{|N|\,(|N| - 1)/2}$$

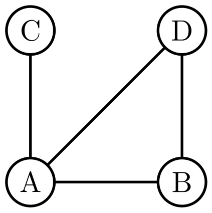


Figure 7.3: The degrees of A, B, C and D are 3, 2, 1 and 2 respectively.

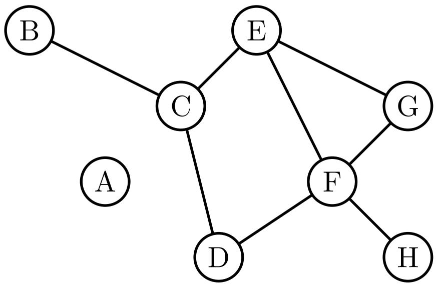


Figure 7.4: Graph with a degree sequence of $(4, 3, 3, 2, 2, 1, 1, 0)$.

Degree

The degree of a node in an undirected graph is the number connections it has to other nodes. In the graph displayed in figure 7.3, node A has a degree of 3, node B a degree of 2 and node C a degree of one.

Often we will let $d(v)$ stand for the degree of a node v. For directed graphs one can make a difference between the in-degree and the out-degree. As the names suggest, these correspond to the number of ingoing and outgoing edges respectively. In this work we will mainly focus on undirected graphs.

Degree Sequence

The degree sequence is arguably one of the most important properties of a graph. It influences both the structure of a graph and processes that operate on it.

It is defined as a monotonic increasing sequence of all the node degrees in a graph. For the example graph in figure 7.4 it is $(5, 3, 3, 2, 2, 1, 0)$. Some degree sequences uniquely correspond to only one possible graph (an extreme example are complete graphs). Other define a *world* of possible graphs.

Clustering Coefficient

The clustering coefficient measures the likelihood that two neighbours of a node are themselves connected. It's commonly defined as

$$\mathrm{CC}(G) = \frac{1}{|V|} \sum_{v \in V} \frac{\Delta(v)}{(d(v)(d(v)-1))/2} \tag{7.1}$$

for a graph $G = (V, E)$ where $\Delta(v)$ denotes the number of triangles (cliques of size 3) containing node v, and $d(v)$ is the degree of v. The denominator in the sum is the number of possible triangles node v can be part of, and the numerator is the actual number of triangles it is part of. Unfortunately this formula is not defined if there are nodes with a degree of 1 or 0 (division by zero). Most of the time the term in the sum is then treated as being equal to zero.

Another common variation of the formula is the following

$$\mathrm{CC}_1(G) = \frac{\sum_{v \in V} \Delta(v)}{\sum_{v \in V}(d(v)(d(v)-1))/2} \tag{7.2}$$

This avoids the devision by zero when some nodes have a degree of zero or one.

Both of these formula are not ideal, because values that are not defined (division by zero) should preferably have no influence on the result [Kai08]. To avoid this we introduce a variation of formula 7.1. Let S denote the set $\{v \in V \mid d(v) \geq 2\}$, then we have

$$\mathrm{CC}_2(G) = \frac{1}{|S|} \sum_{v \in S} \frac{\Delta(v)}{(d(v)(d(v)-1))/2} \tag{7.3}$$

Intuitively the clustering coefficient is a measure of neighbourhood connectivity. If a node has no neighbourhood it shouldn't contribute to the value of the clustering coefficient, making formula 7.3 a better option. However, when testing the effect certain graph modification algorithms have, we will report the clustering coefficient as calculated in equation 7.1. The reason behind this choice is that equation 7.1 is more frequently used, and to remain consistent we will adopt the same formula.

Average Path Length

First consider the shortest path between two nodes $v_1, v_2 \in V$ and denote this by $d(v_1, v_2)$. For unweighed graphs the distance is defined as the number of edges of the shortest path between v_1 and v_2. If there is no path between the nodes the convention will be that $d(v_1, v_2) = 0$. The average path length is then defined as

$$\mathrm{APL} = \frac{1}{n(n-1)} \sum_{i,j} d(v_i, v_j) \tag{7.4}$$

The abbreviation APL will commonly be used in tables and figures. It is a widely used property and can for example stand for the average number of people one will have to communicate through to contact a complete stranger.

Diameter

The diameter is the maximum distance between any pair of nodes. Written formally this becomes

$$\text{Diameter} = \max_{v_i, v_j \in V} d(v_i, v_j) \tag{7.5}$$

Power Law Estimation

The degree distribution of empirically obtained graphs often follow a power law distribution. Intuitively this means that there are a high number of nodes with a low degree and only few nodes with a high degree. A quantity x obeys a power law if it is drawn from a probability distribution

$$p(x) \propto x^{-\alpha} \tag{7.6}$$

where α is a parameter of the distribution known as the exponent or scaling parameter [CSN09]. Most of the time the degree distribution doesn't obey the power law for all values of x. Particularly it only applies for values greater than some minimum x_{min}. For most of our analysis we will pick $x_{\text{min}} = 3$ unless otherwise noted.

A nontrivial problem is how to know whether the degree distribution follows a power law, and if so what the value of α is. To estimate the scaling parameter α we will use the algorithm presented in the paper by Clauset, Shalizi and Newman [CSN09]. Their algorithm can estimate both x_{min} and α, but for our data we have noticed better results when manually setting x_{min} equal to 3. When testing the effect of anonymization we will measure how much the scaling parameter α changes.

7.4 Differential Privacy for Graphs

The original definition of differential privacy was given for tabular data. Hence, before being able to apply differential privacy to graphs, we must first define differential privacy for graphs. Note that we will only briefly cover this subject to inform the reader of its existence, and only the most important definitions are given.

7.4.1 Possible Definitions

The definition of differential privacy for graphs mainly depends on what we consider "neighbouring graphs". A first definition, resembling the opt-in or opt-out semantic of differently privacy, says that two graphs are neighbours if they differ by at most one node and all of its incident edges.

Definition 7.9 (Node neighbouring graphs)**.** *Two graphs $G = (V, E)$ and $G = (V', E')$ are node neighbours if and only if $|V \oplus V'| = 1$ and $|E \oplus E'| = \{(u, v)|u \in (V \oplus V') \text{ or } v \in (V \oplus V')\}$.*

It results in one of the strongest forms of privacy that can be defined, since results should be roughly the same whether or not an individual appears in the graph or not. For completeness, the definition of differential privacy for graphs becomes

Definition 7.10 (ϵ-differential privacy for graphs)**.** *A randomized function $\mathcal{K}$ gives node ϵ-differential privacy if for all neighbouring graphs $G = (V, E)$ and $G' = (V', E')$, and for all outputs $S \subseteq Range(\mathcal{K})$,*

$$\Pr\left[\mathcal{K}(G) \in S\right] \leq \exp\left(\epsilon\right) \Pr\left[\mathcal{K}(G') \in S\right]$$

The definition purposely doesn't specify that the graphs are *node* neighbouring. It allows us to specify other conditions which define when two graphs are neighbouring, resulting in different privacy guarantees. When assuming node neighbouring graphs, one is assuring node ϵ-differential privacy. Unfortunately node ϵ-differential privacy is too strong for many analysis because the amount of noise that must be added to assure privacy makes the final output meaningless [HLMJ09]. For example, the empty graph consisting of n isolated nodes is a neighbour of the star graph (one node connected to n nodes) for node differential privacy, indicating that certain properties can differ significantly for neighbouring graphs.

A more modest definition considers two graphs G and G' to be neighbours if one can produce G' from G by adding or removing one edge, or by adding or removing one isolated node [Hay10].

Definition 7.11 (Edge neighbouring graphs)**.** *Two graphs $G = (V, E)$ and $G = (V', E')$ are edge neighbours if and only if $|V \oplus V'| + |E \oplus E'| \leq 1$.*

Assuming edge neighbouring graphs in the definition of differential privacy for graphs results in edge ϵ-differential privacy. Using this definition less noise has to be added making more analysis possible while assuring privacy. However, it may not be a strong enough privacy guarantee in all situations. For example, the degree of a node is not well protected under this definition. When anonymizing graphs about sexual interaction or shared drug usage the degree of a node is sensitive information and must also be protected. The middle ground is captured by k-edge neighbouring graphs, resulting in k-edge differential privacy.

Definition 7.12 (k-edge neighbouring graphs)**.** *Two graphs $G = (V, E)$ and $G = (V', E')$ are k-edge neighbours if and only if $|V \oplus V'| + |E \oplus E'| \leq k$.*

For k-edge differential privacy two graphs are considered neighbours if they differ by at most k edges or nodes. Note that for nodes with a lower degree than k this is effectively the same as node differential privacy. Nodes with a higher degree face more exposure, but this is only natural, since they also have a higher influence on the structure of the graph. In other words high degree nodes are bound to have a higher probability of disclosing some information.

In complete analogy to differential privacy for tabular data, one can now also define the sensitivity of a function for neighbouring graphs. Then it can be proved that adding noise from $Lap(\Delta f/\epsilon)$, where Δf is the sensitivity of the function, assures privacy. For more details we refer the reader to [HLMJ09].

7.4.2 Degree Distribution Estimation

Estimating the degree distribution is a good example of the applicability of k-edge differential privacy. In fact it becomes almost trivial to calculate it under k-edge differential privacy. First the real degree distribution is computed. The sensitivity of it is $2k$ since adding or removing an edge changes the degree of two nodes by one. Therefore adding noise distributed by $Lap(2k/\epsilon)$ to each position in the real degree distribution will assure k-edge differential privacy.

7.5 Asymptotic Notation

Because at times asymptotic notation is abused, which can lead to ambiguity in certain cases, we will give precise definitions of the notations that are used in the remainder of this work and cited papers. Asymptotic notation is used when analyzing algorithms.

Definition 7.13 (Big-O Notation)**.** *We write $f(x) = O(g(x))$ iff*

$$\exists c > 0, \exists n_0 \geq 0, \forall n \geq n_0 \colon f(n) \leq cg(n)$$

and say that f is Big-O *of g.*

When proving that a function f is indeed Big-O of g the subsequent theorem can be used.

Theorem 7.1. *If $\lim_{x\to\infty} \frac{f(x)}{g(x)} \in \mathbb{R}$ then $f(x) = O(g(x))$.*

Let's first recall the definition of the limit to positive infinity of a real-valued function.

Definition 7.14 (Limit to Infinity)**.** *If we have that*

$$\forall \epsilon > 0, \exists S > 0, \forall x \colon x > S \implies |f(x) - L| < \epsilon$$

then we say that the limit of f as x approaches infinity is L. We will denote this with

$$\lim_{x\to\infty} f(x) = L$$

With this definition in mind the proof of theorem 7.1 becomes rather straightforward.

Proof. From the definition of the limit to infinity it follows that

$$\begin{aligned}
\forall\epsilon > 0, \exists S > 0, \forall x > S\colon\ & \left|\frac{f(x)}{g(x)} - L\right| < \epsilon \\
& \left|\frac{f(x)}{g(x)}\right| - |L| < \epsilon \\
& \left|\frac{f(x)}{g(x)}\right| < \epsilon + |L| \\
& |f(x)| < (\epsilon + |L|) \cdot |g(x)|
\end{aligned}$$

Let $\epsilon' = \epsilon + |L|$ then

$$\forall\epsilon' > |L|, \exists S > 0, \forall x > S\colon\ |f(x)| < \epsilon' |g(x)|$$

This surely implies

$$\exists\epsilon' > 0, \exists S > 0, \forall x > S\colon\ |f(x)| \leq \epsilon' |g(x)|$$

Apart from different variable names this precisely matches the definition of the Big-O notation. □

7.5.1 Special Notations

More important is how we deal with abuse of notation. Commonly the Big-O notation is used inside a formula. For example we can have something like

$$g(x) = h(x) \cdot 2^{O(f(x))}$$

which doesn't strictly follow the definition. In such cases we will define this to mean

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0\colon g(n) \leq h(n) \cdot 2^{cf(n)}$$

One can even have multiple occurrences of the Big-O notation in a formula. Say you have

$$g(x) = n^{O(x^2)} \cdot O(\log\log x)$$

which would translate to the following

$$\exists c_1, c_2 > 0, \exists n_0 > 0, \forall n \geq n_0\colon g(n) \leq c_1 n^2 \cdot c_2 \log\log n$$

In practice, when there are multiple occurrences of the Big-O notation inside a formula, we will commonly define c to be the maximum of all c_i and use that constant instead. So the previous formula would become

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0 \colon g(n) \leq cn^2 \cdot c \log \log n$$

where c is the maximum of c_1 and c_2.

Chapter 8

Naive Anonymization and an Active Attack

In this chapter we will show that publishing only the graph of a social network, i.e., only the nodes and edges, can already disclose sensitive information. We will describe an active attack where the adversary must be able to inserts nodes and edges in the social network before the graph is released.

8.1 Naive Anonymization

One possibility that was once thought to provide enough privacy is called *naive anonymization*. In this mechanism the nodes are renamed to a unique but random identifier and the structure of the graph is left unmodified. In other words only the graph of the social network is published. One might reason that because all the identifying attributes of a person are removed, the privacy of all the persons are now protected.

Unfortunately this is not the case. Nodes can be re-identified even if only the graph of a social network is published. This is possible because the all the edges among the nodes contain a large amount of information. Particularly the *neighborhood* around a node can be used to uniquely locate a node in the naively anonymized graph.

The exact definition of naive anonymization is

Definition 8.1 (Naive Anonymization [Hay10]). *Given a network modeled by an undirected graph $G = (V, E)$, the naive anonymization of G is an isomorphic graph $G_a = (V_a, E_a)$ defined by a random bijection $f : V \to V_a$.*

This prevents re-identification when the adversary posses information about the attributes of a node (e.g. their name, social security number, address, and son on). So in absence of any other information privacy would indeed be preserved. However the adversary may posses information about the structure of the graph itself. Such information can be obtained by either manipulating the network before it is released, or by having access to auxiliary knowledge about the social network structure. This corresponds to the active and passive attack respectively, as we will see.

8.2 Active Attack

The first attack found on a naively anonymized network was due to Backstrom et al. [BDK07]. They described an active attack where the adversary must be able to constructs additional new nodes and edges in the network *before* the anonymized version of the network is created and released. Although such an attack can be hard to execute in practice it's nevertheless a perfect example of how anonymity in graph data does not imply privacy. In their work they also suggested a passive attack which is fairly similar to the active attack, but less powerful.

8.3 Background

We will assume the social network is a graph $G = (V, E)$ where V is the set of nodes of size n and $E \subseteq V^2$ is the set of edges. Nodes commonly correspond to individual users but can also represent group of users, e.g., in many social networks such as facebook a user can choose to be part of certain public or private groups. An edge (u, v) signifies that u has a relationship with v. In this context there is a relationship between two individuals if they are friends on the particular network, they have communicated with each other or they have interacted multiple times using other means. A relationship between an individual and a group occurs when the individual is a member of the particular group.

As mentioned the adversary must be able to create new nodes and edges to the network. He or she can create new nodes by creating new user accounts on the social network. Creating an edge between two nodes can be done by communicating with the other node, befriending the individual, or by subscribing to the updates of the user, or any other possible action depending on the targeted network. Adding edges between two nodes that are under control of the adversary is easy, and creating directed edges between a node that is controlled by an adversary and another targeted node in the network is also trivial. However it can be hard to make the targeted node respond to messages of the adversary. We conclude that adding directed edges is easy to do, but creating an undirected edge (i.e., creating a directed edge from the targeted node back to our node) can be a nontrivial problem.

The attack becomes easier to carry out if the released anonymized graph data is directed [BDK07]. Because of this the attack will focus on the harder case of undirected graphs. In other words we assume the data curator released a graph where the direction of the edges are removed, and only undirected edges remain. At its core the attack consists of picking a set of targeted users $w_1, \ldots, w_b$. The goal is then to find these users in the anonymized graph. Once they have been found we can use the anonymized graph to learn whether there is an edge between any of the targeted users w_i and w_j. Apart from the existence of edges, the adversary will also learn the degree of the targeted users.

8.4 Idea Behind the Attack

Backstorm et al. suggest two active attacks called the *walk-based* and the *cut-based* attack. In the cut-based attack fewer nodes need to be created by the adversary. The disadvantage is that the attack is more complicated, easier to detect and it requires a computationally more demanding algorithm. Therefore we will only consider the walk-based attack in this chapter.

The attack begins by creating k new user accounts for a small value of k. Edges are created between these users resulting in a subgraph H. These newly created users are then connected to the targeted nodes $\{w_, \ldots, w_b\}$ (e.g. by sending messages to the user or by subscribing to the updates of the user) and possibly other nodes as well. This is done in such a way so that we can uniquely identify each targeted user w_i given the location of H, how this is achieved will be explained in the following section. When the data curator releases the anonymized graph G it will contain the subgraph H along with the edges to the targeted nodes. The attack proceeds by finding the constructed graph H. Based on this information the adversary is able to determine the location of the targeted nodes $w_1, \ldots, w_b$, and thus re-identity these users in the supposedly anonymized graph. The adversary will learn the degree of the targeted nodes and also any edges that exists between any of the targeted users. This of course compromises the privacy of the attacked individuals.

The difficulty of the attack lies in constructing a subgraph H that is unique within the global graph G. This is particularly difficult because the adversary doesn't know G at the time of constructing H. He or she must therefore attempt to construct a graph that is unique regardless of the structure of G. Secondly the adversary must be able to efficiently find the subgraph H which has been hidden within the anonymized graph G. In other words he or she must create an instance of the *subgraph isomorphism problem* that is tractable to solve, even for very large graphs G.

8.5 The Details

The main idea of the attack has already been discussed in the previous section, i.e., we construct a new subgraph which we will then attempt to locate in the anonymized graph. We will let X denote the set of nodes we will add to the network and $H = (X, E')$ the subgraph created by these nodes, where E' denotes the edges we will add within H itself. The graph G is the naively anonymized graph released by the data curator which contains the constructed subgraph H. To identity the original graph the notation $G - H$ is used. For a set of nodes S we will let $G[S]$ denote the subgraph of G induced by the nodes in S. Based on this notation we have $H = G[X]$.

8.5.1 Requirements for success

There are a number of technical requirements the subgraph must satisfy in order for the attack to be successful:

1. There exists no set of nodes $S \neq X$ such that $G[S]$ and H are isomorphic.
2. Given the anonymized graph G we must be able to efficiently locate the subgraph H.
3. The subgraph H must contain no non-trivial automorphisms.

An *automorphism* of a graph (V, E) is an automorphism from the set of nodes V to itself. Put differently the nodes of the edges are permutated while the structure of the graph is preserved.

If the first requirements holds we can locate the nodes of the subgraph we have constructed. The second requirements assures we can find these nodes within an acceptable amount of time. The third requirement is needed so we can correctly label the nodes as $x_1, \ldots, x_k$ and hence find the targeted nodes.

8.5.2 Construction of the Subgraph H

The construction of the subgraph happens in four steps:

(1) We begin by creating $k = (2 + \delta) \log n$ new nodes, and represent these using the set $X = \{x_i, \ldots, x_k\}$. Then for each node in $x_i \in X$ we pick an integer value $\Delta_i \in [d_0, d_1]$. The value Δ_i stands for the number of edges node x_i will have to nodes in $G - H$. The two constants $d_0 \leq d_1$ can be chosen arbitrary, but normally they are chosen in such a way to reduce the detectability of the subgraph H. Furthermore it's also possible to simply choose each Δ_i arbitrarily, but in experiments in works well to choose them at uniform from the interval $[d_0, d_1]$ [BDK07].

(2) Now pick the nodes $W = \{w_1, w_2, \ldots, w_b\}$ which we want to re-identity in the anonymized graph. Here $b = O(\log^2 n)$. For each targeted node w_i we choose a set $N_j \subseteq X$ such that all N_j are distinct and each x_i

appears in at most Δ_i of the sets N_j (note that this places an additional constraint on the possible values b can take). We add edges from each node w_j to all nodes in N_j. This allows us to find the targeted nodes once we located H in the anonymized release.

(3) Edges are added from H to $G - H$ so that each node $x_i \in X$ has exactly d_i edges to $G-H$. These edges must be added while preserving the unique linkage to the targeted nodes, i.e., for each targeted user w_j there must be no other nodes in $G - H$ that are also connected to precisely the nodes in N_j. Otherwise we would not be able to uniquely find the targeted users once we located H in the anonymized graph.

(4) As a last step we add all the internal edges of our subgraph H. We begin by including all the edges (x_i, x_{i+1}) for $1 \leq i \leq k-1$. This will allow us to uniquely label the nodes of H once we have located its position, provided that it has no non-trivial automorphisms. Finally we include each other edge with probability 1/2. The degree of a node $x_i \in X$ in the full graph is denoted by Δ'_i which is equal to Δ_i plus its number of edges to other nodes in X.

As already mentioned, in order for the attack to be successful, we must be sure that H contains no non-trivial automorphisms. Results from the domain of random graph theory imply that with high probability our constructed graph H has no non-trivial automorphisms [Bol01]. In the remaining of this work we will assume that H indeed has no non-trivial automorphisms.

An example of a subgraph that might be constructed by this algorithm is shown in figure 8.1. Note that the figure doesn't include any additional edges between the nodes x_i and g_i. These were excluded to prevent the figure from getting too cluttered.

8.5.3 Finding our Constructed Subgraph

We attempt to find H in G by searching for the path $x_1, x_2, \ldots, x_k$. This search begins by considering every node in G with degree Δ'_1. Then we try to add any node with the degree Δ'_2 continuing with nodes of degree Δ'_3 and so on. Apart from this degree test we also use a second edge test which verifies that the edges among the nodes currently in the path exactly correspond with the edges that exists in our constructed graph H. In other words words our search space is pruned by two constraints. The first one being the *degree test* where each possible candidate for node x_i must have the correct degree Δ_i. The second one is an *internal structure test* which verifies that each candidate for node x_i must have the correct subset of edges to the current path $x_i, x_2, \ldots, x_{i-1}$. All this is done by constructing a search tree $\mathcal{T}$ according to the the following details:

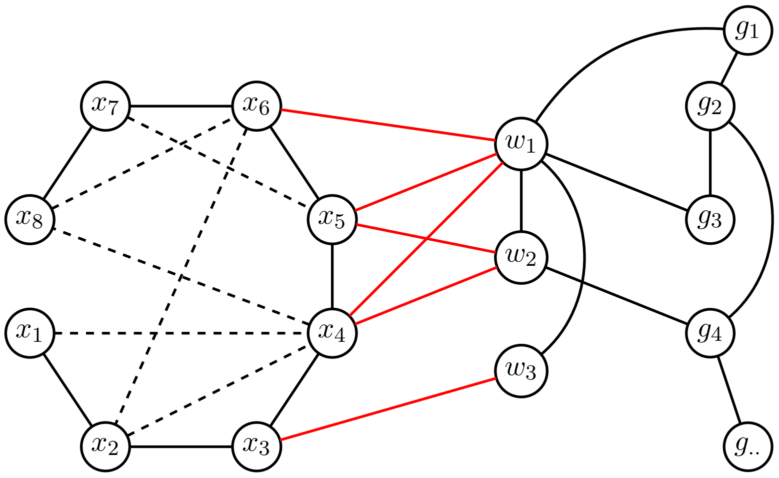


Figure 8.1: Example construction of a subgraph G with $k = 8$. The nodes x_1 to x_8 form the subgraph constructed by the adversary. Nodes w_1, w_2 and w_3 are the targeted nodes. The edges draw in red are the unique sets N_i to identify the targeted nodes. For completeness a few nodes g_i are also shown and represented the remained of the released graph G.

1. Every other node in the search tree corresponds with a node in G. Letting α be a node in $\mathcal{T}$, we will define $f(\alpha)$ as the node in G. It's possible that a node it represents in G appears multiple times in the search tree. The tree $\mathcal{T}$ is now constructed in such a way so that for every path of nodes $\alpha_1, \ldots, \alpha_\ell$ from the root to a leaf node, the corresponding nodes $f(\alpha_1), \ldots, f(\alpha_\ell)$ from a path in G with the same sequence of degrees and the same internal edges as $x_1, \ldots, x_\ell$. Additionally every such path in G corresponds to a distinct rooted path in $\mathcal{T}$ [BDK07].

2. The actual construction of $\mathcal{T}$ begins with a fixed dummy root node α^*. All the nodes in G having degree Δ'_1 are considered children of the root node. For further constructing $\mathcal{T}$ we consider every current leaf node α with its associated path $\alpha^* = \alpha_0, \alpha_1, \ldots, \alpha_\ell = \alpha$ to the dummy root node. We know that this path corresponds to a path $f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_\ell)$ in the graph G (notice that here we ignore the dummy root node $\alpha^* = \alpha_0$). The task is now to find all neighbours v of $f(\alpha_\ell)$ in G for which the degree of v is $\Delta'_{\ell+1}$ and has the correct subset of edges to the path, i.e.,

$$\forall \ell \colon 1 \le i \le \ell \colon (f(\alpha_i), v) \text{ is an edge iff } (x_i, x_{\ell+1}) \text{ is an edge.}$$

For every v satisfying both these test we create a new child β of α with $f(\beta) = v$.

3. Once the tree has been build and there is a unique rooted path of length k, then we know that it correspond to the nodes in H. Moreover we immediately learn the labels of these nodes since we associated each node α in $\mathcal{T}$ with a specific node $f(\alpha)$ in G, and we know that for the k-length path the node α_0 corresponds with x_0, α_1 with x_1, and so on.

Using H we can now re-identify the targeted nodes $w_1, w_2, \ldots, w_b$ by following the unique set of edges N_i to the target node w_i. Note that the running time of this algorithm is only slightly larger that the size of $\mathcal{T}$.

8.5.4 Privacy Breach

Having identified the targeted nodes we learn their node degree and whether there are edges between any of the targeted nodes. As explain in chapter 7 this can result in a privacy breach depending on the nature of the network that is being attacked.

8.6 Analysis

Both the correctness of our assumptions and the efficiency of the attack have to be proved. Our assumption is that with high probability the graph H will be a unique subgraph in G. To prove the efficiency we show that the size of the search tree $\mathcal{T}$ does not grow too large.

Uniqueness of H

Although describing the attack was not complicated, the analysis and proofs are complex and not easily comprehensible [BDK07]. Since the true focus of this work is to design robust anonymization techniques we will omit these long and intricate proofs. Instead we will focus on a simplified version of the theorem which states that H is unique with high probability. This proof is also included in the paper by Backstrom, Dwork and Kleinberg [BDK07] as a motivating example, and provides the technical intuition and idea behind the more general statement of the theorem. Let us first state the general theorem.

Theorem 8.1. *Let $k \geq (2+\delta)\log n$ for an arbitrary positive constant $\delta > 0$, and suppose we use the following process to construct an n-node graph G:*

1. *We start with an arbitrary graph G' on $n-k$ nodes and we attach new nodes $X = \{x_1, \ldots, x_k\}$ and arbitrarily include some edges to nodes in G'.*

2. *We build a random subgraph H on X by including each edge (x_i, x_{i+1}) for $i = 1, \ldots, k-1$, and including each other edge independently with probability* $1/2$.

Then with high probability there is no subset of nodes $S \neq X$ in G such that $G[S]$ is isomorphic to $H = G[X]$.

We will prove a more restricted statement where the subset of nodes S need not be merely unequal to X, but also disjoint from X. This is easier to prove while still capturing the idea behind the more general proof.

Claim 8.2. *Let $\mathcal{F}_0$ be the event that there is no subset of nodes S disjoint from X such that $G[S]$ is isomorphic to H. Then with high probability the event $\mathcal{F}_0$ holds.*

Proof. We pick an arbitrary graph G yet consider it to be a fixed graph during the proof. Pick a specific ordered sequence $S = (s_1, s_2, \dots, s_k)$ of k nodes in $G - H$ having edges between the consecutive nodes s_i and s_{i+1}. Define $\mathcal{E}_S$ as the event that the function $f \colon S \to X$ given by $f(s_i) = x_i$ is an isomorphism for H and $G[S]$. Since graph G is fixed, all but $k-1$ of the edges in H are chosen independently with probability 1/2, and since S is disjoint from X, we have that

$$\Pr\left[\mathcal{E}_S\right] = \left(\frac{1}{2}\right)^{\binom{k}{2}-(k-1)} \tag{8.1}$$

where the probability is over all possible graphs H. Simplifying the exponent using basic operations gives

$$\begin{aligned}
\binom{k}{2} - (k-1) &= \frac{k(k-1)}{2} - \frac{2(k-1)}{2} &(8.2)\\
&= \frac{k^2 - 3k + 2}{2} &(8.3)\\
&= \frac{(k-1)(k-2)}{2} &(8.4)\\
&= \binom{k-1}{2} &(8.5)
\end{aligned}$$

And hence we get

$$\Pr\left[\mathcal{E}_S\right] = \left(\frac{1}{2}\right)^{\binom{k-2}{2}} = 2^{-\binom{k-2}{2}} \tag{8.6}$$

where the probability is again for all possible graphs H. We now use the equation $\mathcal{F}_0 = \cup_S \mathcal{E}_S$ where the union is over all possible sequences S of k nodes in $G - H$. Recalling that G has n nodes, the number of different sequences S is bounded by n^k. As we picked $k \geq (2+\delta)\log n$ when constructing the subgraph H we have that $n \leq 2^{k/(2+\delta)}$. This gives

$$
\begin{aligned}
\Pr[\mathcal{F}_0] &= \Pr[\cup_S \mathcal{E}_S] &(8.7)\\
&\leq n^k \cdot 2^{-\binom{k-1}{2}} &(8.8)\\
&\leq 2^{\frac{k^2}{2+\delta}} \cdot 2^{-\frac{k^2}{2}} \cdot 2^{1+\frac{3k}{2}} &(8.9)\\
&= 2^{\frac{-\delta k^2}{2(2+\delta)}+\frac{3k}{2}+1} &(8.10)
\end{aligned}
$$

where inequality 8.7 follows from a union bound as explained in section 2.4.1. We notice that the probability of event $\mathcal{F}_0$ goes to zero exponentially quick in k. This completes the proof of claim 8.2. $\square$

For the complete proof we refer the reader to the paper by Backstrom et al. [BDK07]. An important part of their complete proof is claim 8.3 as stated below.

Claim 8.3. *With high probability, for any constant $c \geq 15$, the following holds. Let A, B and Y be disjoint sets of nodes in G with $A, B \subseteq X$, and let $f\colon A \cup Y \to B \cup Y$ be an isomorphism. Then the set Y contains at most $c \log k$ nodes that are not fixed points of f.*

Although no proof is given of this claim here, we will however use this when proving that the size of the search tree $\mathcal{T}$ is relatively small.

Efficient Recovery

Finally it's possible to show that the search tree $\mathcal{T}$ doesn't grow too large in size.

Theorem 8.4. *For every $\epsilon > 0$, with high probability the size of $\mathcal{T}$ is $O(n^{1+\epsilon})$.*

Proof. Recall that k denotes the size of H, and let d be the maximum degree in G of a node in H. As a result from the construction of H both of these quantities are $O(\log n)$. Let Γ be a random variable equal to the number of nodes in the search tree $\mathcal{T}$. We decompose Γ into the sum of two simpler variables. The first one is Γ' and stands for the number of paths of $G - H$ corresponding to some node of $\mathcal{T}$. The second variable Γ'' is the number of paths in G that meet H and correspond to some node in $\mathcal{T}$. We have $\Gamma = \Gamma' + \Gamma''$. Our goal is now to find an upper bound on $\mathrm{E}[\Gamma]$. During the proof we will assume the events $\mathcal{F}_0$ and $\mathcal{F}_1$ hold.

Part One. We begin by finding an upper found for $\mathrm{E}[\Gamma']$. For any path P in $G - H$ we define the random variable Γ'_P as

$$\Gamma'_P = \begin{cases} 1 & \text{if } P \text{ corresponds to a node in } \mathcal{T} \\ 0 & \text{otherwise} \end{cases}$$

Further we call a path P *feasible* if the degree of every node in P is at most d. If P is not feasible then Γ'_P is almost surely (i.e., with probability one) equal to 0.

Now consider any feasible path P of length $j \leq k$. For P to be present in $\mathcal{T}$, i.e., that there exists a node α in $\mathcal{T}$ defining a path that corresponds to P, we need the edges among nodes in P to match the edges among $\{x_1, x_2, \ldots, x_j\}$. In the probability calculations we can imagine that the edges between nodes in $\{x_1, x_2, \ldots, x_j\}$ are being generated *after* P is chosen. The edges (x_i, x_{i+1}) for $1 \leq i \leq j-1$ resulting from the path $x_1, x_2, \ldots, x_j$ are of course already known to exist. We have, even for the special cases when the length j of the path is 1 or 2, that

$$\begin{aligned} \mathrm{E}\left[\Gamma'_P\right] &= 0 \cdot \Pr\left[\Gamma'_P = 0\right] + 1 \cdot \Pr\left[\Gamma'_P\right] && (8.11)\\ &= \Pr\left[\Gamma'_P\right] && (8.12)\\ &= \left(\frac{1}{2}\right)^{\binom{j}{2}-(j-1)} && (8.13)\\ &= 2^{-\binom{j-1}{2}} && (8.14) \end{aligned}$$

A feasible path of length j can start with any node in the graph having at most degree d. At worst these are n possibilities. For every consecutive node in the path we can choose between at most d neighbours of the current node, since the degree of all nodes in a feasible path is at most d. Thus the worst case total number of feasible paths of length j is nd^{j-1}. Hence we obtain

$$\begin{aligned} \mathrm{E}\left[\Gamma'\right] &\leq \sum_P \mathrm{E}\left[\Gamma'_P\right] && (8.15)\\ &\leq \sum_{j=1}^{k} nd^{j-1} 2^{-\binom{j-1}{2}} && (8.16)\\ &= n \sum_{j=1}^{k} \left(d 2^{-\frac{j-2}{2}}\right)^{j-1} && (8.17) \end{aligned}$$

To simplify the inequality we use the fact that $d = O(\log n)$. In other words there exists a $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$:

$$\mathrm{E}\left[\Gamma'\right] \leq n \sum_{j=1}^{k} \left(n \log(c) 2^{-\frac{j-2}{2}}\right)^{j-1} \qquad (8.18)$$

Let $s = 2 \log\log n + 2 \log c + 2$ and split the sum into two parts based on this value.

$$\mathrm{E}\left[\Gamma'\right] \leq n\sum_{j=1}^{s-1}\left(c\log(n)2^{-\frac{j-2}{2}}\right)^{j-1} + n\sum_{j=s}^{k}\left(c\log(n)2^{-\frac{j-2}{2}}\right)^{j-1} \tag{8.19}$$

Focusing on the second sum we notice that every j in it is larger or equal to s. Hence we obtain the following

$$\begin{aligned}
j \geq s &= 2\log\log n + 2\log c + 2 && (8.20)\\
\frac{j-2}{2} &\geq \log\log n + \log c && (8.21)\\
2^{-\frac{j-2}{2}} &\leq (\log(n)c)^{-1} && (8.22)\\
c\log(n)2^{-\frac{j-2}{2}} &\leq 1 && (8.23)\\
\left(c\log(n)2^{-\frac{j-2}{2}}\right)^{j-1} &\leq 1 && (8.24)\\
& && (8.25)
\end{aligned}$$

From this we deduce that each term in the second sum is smaller or equal to one. In other words, once j is $\Theta(\log\log n)$ the second term is smaller than 1. Since there are less than k such terms, the total sum is surely bounded by k. Hence we have found the following upper bound for the second sum

$$n\sum_{j=s}^{k}\left(c\log(n)2^{-\frac{j-2}{2}}\right)^{j-1} \leq nk \tag{8.26}$$

We continue with finding an upper bound for the first sum. Observe that for $j \geq 1$ the factor $2^{-j/2}$ can only decrease the sum. Hence we can drop the factor and introduce an inequality. Further working out this inequality gives us

$$\begin{aligned}
n\sum_{j=1}^{s-1}\left(c\log(n)2^{-\frac{j-2}{2}}\right)^{j-1} &= n\sum_{j=1}^{s-1}\left(2c\log(n)2^{-\frac{j}{2}}\right)^{j-1} && (8.27)\\
&\leq n\sum_{j=1}^{s-1}(2c\log(n))^{j-1} && (8.28)\\
&= n\sum_{j=0}^{s-2}(2c\log(n))^{j} && (8.29)\\
&= n\frac{(2c\log n)^{s-2}-1}{2c\log n - 1} && (8.30)
\end{aligned}$$

The last equality uses the formula of geometric series. Finally we can rewrite this back into asymptotic notation.

$$n \sum_{j=1}^{s-1} \left(c \log(n) 2^{-\frac{j-2}{2}}\right)^{j-1} = nO(\log n)^{O(\log\log n)} \tag{8.31}$$

After all this we end up with our upper bound

$$\mathrm{E}\left[\Gamma'\right] = n \cdot (O(\log n)^{O(\log\log n)} + O(\log n))$$

which we can further simplify. Let c be the maximum of the constant values used in all three Big-O notations (this is possible because all functions are monotonically increasing). With this we can rewrite the upper bound.

$$\begin{aligned}
\exists n_0 > 0, \forall n > n_0 \colon \mathrm{E}\left[\Gamma'\right] &\leq n\left((c\log n)^{c\log\log n} + c\log(n)\right) \\
&= n\left(2^{\log(c\log n)\cdot c\cdot\log\log n} + 2^{\log(c\log n)}\right) \\
&= n\left(2^{\log(c)\cdot c\cdot\log\log n + c\cdot(\log\log n)^2} + 2^{\log(c)+\log n}\right) \\
&= n2^{O((\log\log n)^2)}
\end{aligned}$$

At long last we have the upper bound $\mathrm{E}[\Gamma'] = n2^{O((\log\log n)^2)}$ and this concludes the first part of the proof.

Part Two. For the second part of the proof we will try to find an upper bound for $\mathrm{E}[\Gamma'']$. This is done by decomposing it into a separate random variables for each possible pattern in which a path could snake in and out of H. In order to describe such a pattern we will use the notion of a *template.* A template τ is a sequence of $\ell \leq k$ symbols $(\tau_1, \dots, \tau_\ell)$ where symbol $\tau_i \in X$ is either a distinct node of H or the special symbol $\perp$. We call the set of all nodes of H that appear in τ the *support* of τ, and denote it $s(\tau)$. For example the support of template

$$\tau = (x_2, x_6, \perp, \perp, x_1, \perp)$$

is $\{x_1, x_2, x_6\}$. We will say that a path P in G is *associated* with τ if the i^{th} node on P lies in $G - H$ for $\tau_i = \perp$, and otherwise is equal to $\tau_i \in X$. So for our example template, possible paths associated with it are

$$x_2, x_6, g_1, g_1, x_1, g_{33}$$
$$x_2, x_6, g_5, g_2, x_1, g_{12}$$
$$\cdots$$

where each g_i can be any node in $G - H$. Finally, we say that the *reduction* of τ, denoted $\overline{\tau}$, is the template for which $\overline{\tau}_i = \perp$ whenever $\tau_i = \perp$, and for which $\overline{\tau}_i = x_i$ otherwise. We will call such a template $\overline{\tau}$ a reduced template. The reduced template of our example is

$$(x_1, x_2, \perp, \perp, x_5, \perp)$$

Notice that the support of a reduced template need to be the same as the support of the original template.

Let Γ''_τ be a random variable equal to the number of paths P associated with τ that are represented in the search tree $\mathcal{T}$. The length ℓ of τ is also equal to the length of the path P. If at least one such path exists, then there is an isomorphism

$$f\colon s(\tau) \to s(\overline{\tau})$$

given by $f(x) = x_i$ when $\tau_i = x$. Why is this an isomorphism? Well, if a path P is represented in the search tree $\mathcal{T}$ then it passed the internal structure test, which verifies that each node in the path has the same subset of edges as are present our constructed subgraph. The isomorphism we defined is exactly this bijection between the nodes of the path P—which passed the internal structure test—and the corresponding subset of nodes in our constructed subgraph H.

We know from claim 8.3, taking $A = \emptyset$, $Y = s(\tau)$ and $B = s(\overline{\tau}) - s(\tau)$, that with high probability all but at most $O(\log k)$ nodes are fixed points of f. For the remainder of the proof we will assume this is indeed the case. Put differently we have that τ agrees with $\overline{\tau}$ on all but $O(\log k)$ positions.

From this one can deduce that the only templates τ for which Γ''_τ can be non-zero are those that differ in at most $O(\log k)$ positions from a reduced template. After all, if it differs in more than $O(\log k)$ positions then, following from the contraposition of claim 8.3, there wouldn't exist an isomorphism between $s(\tau)$ and $s(\overline{\tau})$, hence—again by contraposition—there would be no path P associated with τ that is represented in the search tree $\mathcal{T}$.

To continue with our search for an upper bound we will decompose Γ''_τ into a sum of random variables $\Gamma''_{\tau j}$ for every feasible path P associated with τ. If P is represented in $\mathcal{T}$ then $\Gamma''_{\tau P} = 1$ and otherwise $\Gamma''_{\tau P} = 0$. We now have that

$$\begin{aligned} \mathrm{E}\left[\Gamma''\right] &\leq \sum_{\tau,P} \mathrm{E}\left[\Gamma''_{\tau P}\right] \\ &= \sum_{\tau,P} \Pr\left[\Gamma''_{\tau P} = 1\right] \end{aligned}$$

To calculate this sum we will actually consider all reduced templates with exactly j $\perp$'s, and then sum over all possible values of j. First we want to know how many reduced templates there are with j $\perp$'s. Note that the template cannot contain only $\perp$'s since then any path associated with it wouldn't contain nodes of H, meaning the minimum length ℓ of a template containing j $\perp$'s is $j+1$. For a reduced template we only need to pick the location of the $\perp$'s, the other symbols are then determined by their location, i.e., location i will then contain x_i. We have that the total number of templates with j $\perp$'s is

$$\sum_{\ell=j+1}^{k} \binom{\ell}{j}$$

It is proven in section 8.11 that k^{j+1} is an upper bound of this sum. Hence there are at most k^{j+1} reduced templates with j $\perp$'s, and therefore at most $k^{O(\log k)} \cdot k^{j+1}$ templates τ with j $\perp$'s for which $\Gamma''_{\tau P}$ can be non-zero. For each such τ, there are at most d^j feasible paths P associated with τ. Each such path P as a probability of *at most* $2^{-\binom{j-1}{2}}$ of being represented in $\mathcal{T}$. When summing over all j's this results in

$$\begin{aligned} \mathrm{E}\left[\Gamma''\right] &\leq \sum_{j=1}^{k-1} k^{j+1} d^j k^{O(\log n)} 2^{-\binom{j-1}{2}} \\ &= k^{O(\log n)} \sum_{j=1}^{k-1} k^2 d \left(\frac{kd}{2^{(j-2)/2}}\right)^{j-1} \end{aligned}$$

Similarly to part one of the proof, once j is $\Theta(\log kd) = \Theta(\log\log n)$ each term is less than 1, so

$$\begin{aligned} \mathrm{E}\left[\Gamma''\right] &\leq k^{O(\log k)} O(\log\log n)(kd)^{O(\log\log n)} && (8.32) \\ &= 2^{O((\log\log n)^2)} && (8.33) \end{aligned}$$

Conclusion. Combining both upper bounds we can conclude that $\mathrm{E}\left[\Gamma\right] = n2^{O((\log\log n)^2)} + 2^{O((\log\log n)^2)} = n2^{O((\log\log n)^2)}$.

Lemma 8.5. $\forall\epsilon > 0$: $f(x) \in n2^{O((\log\log n)^2)} \implies f(x) \in O(n^{1+\epsilon})$.

Lemma 8.5 shows that this implies that $\mathrm{E}\left[\Gamma\right] = O(n^{1+\epsilon})$ for every $\epsilon > 0$. For a proof of lemma 8.5 see section 8.11. This completes the proof of theorem 8.4. $\square$

8.7 Experiments

In order the verify the attack a simulation was carried out on real social network data. The social graph used in the experiments was creating by crawling the blogging site LiveJournal. Each node in the graph corresponds

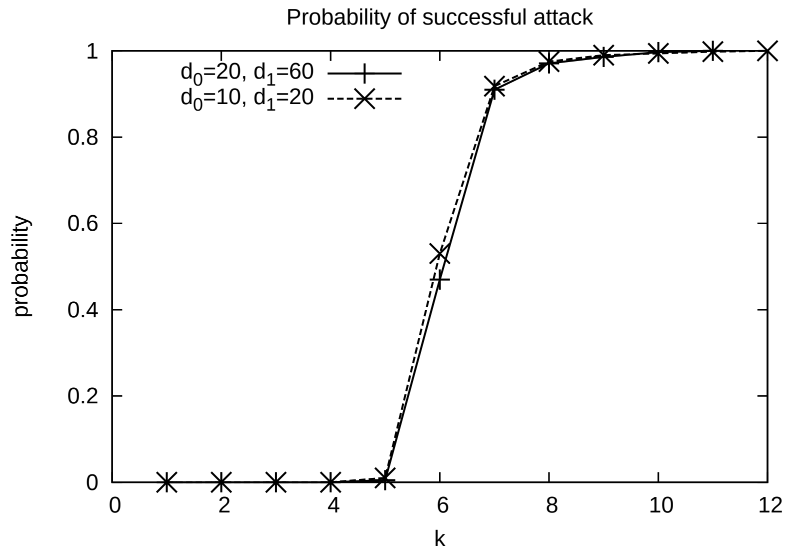


Figure 8.2: Success rate of the active attack for two given intervals $[d_0, d_1]$ on the LiveJournal graph. The x-axis show the size of the constructed subgraph H and the y-axis the probability of uniquely locating H. [BDK07]

to a user with a public blog. Users are able to mark other users as their friends which in turn creates a directional edge in the underlying social graph. Since our focus is on undirected edges we will convert all directed edges to undirected ones. Obviously all the nodes in their crawl are users who have chosen to publish their information publicly on the web, so a selection bias may be present. In total the graph has 4.4 million nodes and 77 million edges and has the same properties compared to other large online social networks [BDK07]. Anonymization has been simulated by removing all the attributes such as the username, location, age, etc. from the nodes.

The attack succeeds when the adversary can uniquely locate the constructed subgraph H within the LiveJournal social graph. To simulate the attack we randomly created several subgraphs over a range of possible sizes, and check if we can find a unique match in G. The construction is exactly as explained previously, i.e., during the simulation we will randomly select a few nodes to serve as the targeted nodes and we will further include edges to G so each node has the required amount of edges Δ_i it nodes in $G - H$. The probability of success is then defined as the as the number of unique matches divided by the number of attempts. Figure 8.2 shows the success probability for varying values of k, the size of G, and for two different choices of the interval $[d_0, d_1]$ [BDK07]. These two intervals fall well within the degrees typically found in G.

From figure 8.2 we notice that with $k \geq 7$ the attack has a high probability of success. The reason such a low value already results in a successful attack is because, apart from an internal structure test, we also utilize the degree test. In the LiveJournal graph each of the possible Hamiltonian graphs[1] on 7 nodes actually occurs. Hence the degree sequence of H within G is essential in finding a unique match. Note that in theorem 8.1 we base the probability of recovering the constructed subgraph merely on its internal structure. As $k = 7$ is much lower compared to the analyzed bound of $(2 + \delta) \log n$ there results are not conflicting.

Not only must the adversary be able to recover G, he or she must also be able to do this within an acceptable amount of time. During the analyses we briefly stated that the running time of the algorithm is only a small factor larger than the size of the search tree. Combined with theorem 8.4 this means that the proposed algorithm for recovering the subgraph H is efficient. And indeed, in practice the search tree $\mathcal{T}$ is not much larger than the number of nodes u such that $d(u) = d(x_1)$, making its recovery fast in practice [BDK07].

8.8 Detectability

We have purposely restricted the degree of the edges to the interval $[d_0, d_1]$ during the construction phase. If one chooses this interval to match the average degrees found in the social graph, then it shouldn't stand out based on its degree distribution. We have also assumed that we are working with undirected edges. Based on this one may conjecture that the attack is hard to detect.

In practical settings however, we commonly have to deal with directed edges. While this would make the active attack easier to execute, it also makes it easier to detect by the data curator. The reason is that the sybil[2] nodes, i.e., the nodes added by the adversary, will likely have very little incoming edges. This is because must users will have little to no reason to link back to the sybil nodes. In turn this results in the subgraph having many outgoing edges while having no incoming edges, which makes it easier to detect. Current research also shows that it's possible to detect sybil attacks on social networks [YGKX08].

[1]A Hamiltonian graph is a graph possessing a cycle visiting each node exactly once.

[2]This comes from the term *Sybil attack*, which is an attack in computer security where multiple identities in a system are forged with the goal of exploiting or subverting the system.

8.9 As a Passive Attack

As a final note we can convert the active attack to a less powerful passive attack. The idea is that instead of creating a subgraph, users currently participating in the network from a coalition. They then attempt to locate themselves in the released graph. If this is successful it might be possible to re-identify certain friends.

Such an attack is however of limited effect. A large obstacle is that we cannot target arbitrary users. Only users having edges to the graph formed by the coalition can potentially be re-identified. This greatly restricts the effectivity of the attack. On the other hand it has been found empirically that once a collation is moderately-sized, it can uniquely locate itself and compromise the privacy of at least some users [BDK07]. We also note that finding a coalition need to be hard. An adversary can pay other users for their information, partial information might be retrieved from other public networks, a badly secured account can be hacked, and so on.

8.10 Conclusion

The active attack shows that naive anonymization is flawed and does not assure privacy. But on the other hand the attack requires that the adversary is able to influence the network before any data is released. Moreover he or she has to able to create edges to the targeted nodes. This is not always an easy task. Given that in some networks an edge is only created when, say, the targeted node actually accepts a friend request, this could greatly reduce the effect of the active attack. After all, the targeted node will not always accept such a request. Even though the feasibility of the attack can be called into question, it does show that naive anonymization is *not* an option.

To provide privacy we must find a mathematically rigor definition of privacy that is applicable to graph data. For satisfying such a definition one will have to change the underlying graph structure, otherwise an active attack is possible. The search is now on for a definition that provides adequate privacy while also being achievable in practice. This will be our focus for the next chapter.

8.11 Proofs

8.11.1 Lemma 8.5

We first restate the lemma below.

Lemma 8.5. $\forall \epsilon > 0 \colon f(x) \in n2^{O((\log\log n)^2)} \implies f(x) \in O(n^{1+\epsilon})$.

Proof. The proof begins by expanding the left hand side

$$\exists c > 0, \exists n_0 > 0, \forall n > n_0 \colon f(n) \leq n 2^{c(\log\log n)^2} \tag{8.34}$$

Here the implication can be proven by showing that

$$\forall \epsilon > 0, \forall c > 0 \colon n 2^{c(\log\log n)^2} \in O(n^{1+\epsilon}) \tag{8.35}$$

This in turn can be proven by using theorem 7.1 and showing that for all positive ϵ and c it is true that

$$\lim_{n\to\infty} \frac{n 2^{c(\log\log n)^2}}{n^{1+\epsilon}} \in \mathbb{R} \tag{8.36}$$

Working out this limit can be done as follows

$$\lim_{n\to\infty} \left[\frac{n 2^{c(\log\log n)^2}}{n^{1+\epsilon}} \right] = \lim_{n\to\infty} \left[2^{c(\log\log n)^2) - \epsilon \log n} \right] \tag{8.37}$$

We continue by proving that the exponent in the limit goes to minus infinity.

$$\lim_{n\to\infty} \left[c(\log\log n)^2 - \epsilon \log n \right] = \lim_{n\to\infty} \left[\log n \left(c \frac{(\log\log n)^2}{\log n} - \epsilon \right) \right] \tag{8.38}$$

This can be further worked out by showing that

$$\lim_{n\to\infty} \left[c \frac{(\log\log n)^2}{\log n} \right] = c \lim_{n\to\infty} \left[2 \frac{\log\log n}{\log n} \right] \tag{8.39}$$

$$= c \lim_{n\to\infty} \left[2 \frac{1}{\log n} \right] \tag{8.40}$$

$$= c \cdot 0 = 0 \tag{8.41}$$

where the first two equations use l'Hôpital's rule. Plugging this result into equation 8.38 we have that, given a ϵ greater than zero, the exponent in the limit of equation 8.37 goes to minus infinity. In turn this proves that the limit we started with in equation 8.36 goes to zero, which is indeed an element of $\mathbb{R}$. □

8.11.2 Lemma 8.6

We prove the following result

Lemma 8.6. *For every k and j it holds that*

$$\sum_{\ell=j+1}^{k} \binom{\ell}{j} \leq k^{j+1}$$

with k and j are natural numbers.

Proof. The proof is by induction. As the base case we will prove the statement is true for all positive k and $j = 0$.

$$\sum_{\ell=1}^{k} \binom{\ell}{0} = \sum_{\ell=1}^{k} \frac{\ell!}{0!\ell!} \tag{8.42}$$

$$= \sum_{\ell=1}^{k} 1 \tag{8.43}$$

$$= k \tag{8.44}$$

$$\leq k^{j+1} \tag{8.45}$$

For the inductive step we will assume the statement holds for all k and $j = n$. We will then prove that is also holds for all k and $j = n + 1$. To begin we have

$$\sum_{\ell=n+2}^{k} \binom{\ell}{n+1} = \sum_{\ell=n+1}^{k} \binom{\ell}{n+1} - \binom{n+1}{n+1} \tag{8.46}$$

To reduce this to the induction hypothesis we exploit the following observation

$$\binom{\ell}{n+1} = \frac{\ell!}{(n+1)!(\ell-n-1)!} \tag{8.47}$$

$$= \frac{\ell - n}{(n+1)} \cdot \frac{\ell!}{n!(\ell-n)!} \tag{8.48}$$

$$= \frac{\ell - n}{(n+1)} \cdot \binom{\ell}{n} \tag{8.49}$$

Filling this into equation 8.46 gives

$$\sum_{\ell=n+2}^{k} \binom{\ell}{n+1} = \sum_{\ell=n+1}^{k} \frac{\ell-n}{(n+1)} \cdot \binom{\ell}{n} - 1 \tag{8.50}$$

$$\leq \sum_{\ell=n+1}^{k} \frac{k-n}{(n+1)} \cdot \binom{\ell}{n} \tag{8.51}$$

$$= \frac{k-n}{(n+1)} \cdot \sum_{\ell=n+1}^{k} \binom{\ell}{n} \tag{8.52}$$

$$\leq k \cdot k^{j+1} \tag{8.53}$$

$$= k^{j+2} \tag{8.54}$$

The first inequality is true because ℓ is always smaller or equal to k. The second inequality follows from the induction hypothesis and the fact that $n \geq 0$. This proofs the induction step, and hence it has now been proved by the principle of induction that the formula holds for all positive k and n. $\square$

Chapter 9

Degree Anonymization

In this chapter we will consider a privacy mechanism called degree anonymization, which attempts to transform the graph in order to guarantee privacy. Both the implementation and evaluation of several algorithms will be discussed in detail. The chapter is based on the paper by Liu and Terzi [LT08].

9.1 Introduction

We make the assumption that the adversary only knows the degree of a targeted node. This can happen when the adversary visits a public profile of the target, and he or she can see how many friends the target has. It must be noted that a more powerful adversary can possess more information which can be used to identify an individual. Nevertheless it is an interesting assumption against weak adversaries, or in cases where the network data doesn't contain highly sensitive information and protection against powerful adversaries is not deemed necessary. Admittedly this makes the applicability of degree anonymization rather low. However it provides a good insight in how anonymization can be done, and illustrates the importance of knowing how much auxiliary information the adversary has. Specifically, in chapter 10 we will argue that a powerful adversary can still defeat degree anonymization.

9.2 Anonymity Guarantees

The degree anonymization mechanism is similar to k-anonymity. The goal is to modify the input graph, with all attributes removed, so each node has the same degree with at least $k-1$ other nodes. We will call this privacy definition k-degree anonymity. The definition is based on k-anonymous vectors.

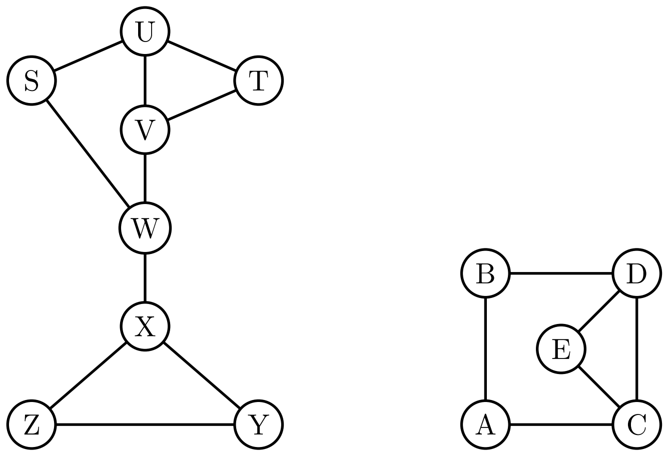


Figure 9.1: On the left is a 4-anonymous graph, and on the right a 2-anonymous graph.

Definition 9.1 (k-anonymous vector)**.** *A vector of integers is k-anonymous if every distinct value in it appears at least k times.*

For example the vector $[5, 5, 3, 3, 2, 2, 2]$ is 2-anonymous. The definition of k-degree anonymity now states that the degree sequence of G must be k-anonymous. By first defining k-anonymous vectors it will be easier for us to split the k-degree anonymization algorithm up in two parts. As we shall see, first we can create a k-anonymous degree sequence, and then construct a graph having this degree sequence that resembles the input graph.

Definition 9.2 (k-degree anonymity)**.** *A graph is k-degree anonymous if its degree sequence is k-anonymous.*

Assuming an adversary only knows the degree of individuals, k-degree anonymity prevents identity disclose but may not prevent edge disclosure in all cases (this is similar to the homogeneity attack on k-anonymity described in section 3.3.2). An example of a 4-degree anonymous graph and a 3-degree anonymous graph is shown in figure 9.1. Theoretically transforming the input graph to the complete graph would make the graph satisfy k-degree anonymity, but would render the graph useless for any study. Moreover, simply copying the graph $k-1$ times would also make it satisfy the privacy definition, but clearly provides zero additional privacy. So we want to find a graph satisfying k-degree anonymity without adding or removing nodes, and by using the minimum amount of edge-modifications on the input graph.

Definition 9.3 (k-degree anonymization)**.** *The k-degree anonymization of an input graph $G = (V, E)$ is a graph $\widehat{G} = (V, \widehat{E})$ that is k-degree anonymous, obtained by modifying the input graph G using the minimum amount of insertions or deletions of edges, i.e., it minimizes $|\widehat{E} - E| + |E - \widehat{E}|$.*

The algorithm we describe in this chapter will assume the graph is simple, i.e., the graph is undirected, unweighted and contains no self-loops or multiple edges. From the definition of k-anonymous vectors it's trivial to derive the following claim

Claim 9.1. *If a graph is k_1-degree anonymous, then it is also k_2-degree anonymous, for every $k_2 \leq k_1$.*

Based on these definition we will be able to create a graph anonymization algorithm which will assure k-degree anonymity. Unfortunately guaranteeing that only the minimum number of possible alterations are made to the input graph is a nontrivial problem, and we will have to fall back on approximations.

9.3 Problem Description

Let $G = (V, E)$ be the input graph we want to anonymize. For now we will restrict ourselves only to edge additions. Edge deletions can be simulated by the giving the complement of the input graph to the anonymization algorithm. Adding an edge in the complement of the graph is identical to removing edges in the original graph. In algorithm 9.1 the details are shown on how to turn an edge addition algorithm into an edge deletion algorithm. The function Anonymization_Edge_Addition is the anonymization algorithm using only edge additions and returns a k-degree anonymous graph.

Algorithm 9.1 Simulating edge deletions.

```
1: $\overline{G}$ = Complement of input graph $G$.
2: $\overline{G}_a$ = Anonymization_Edge_Addition($\overline{G}, k$).
3: **return** $G_a$ = Complement of $\overline{G}_a$
```

An algorithm capable of simultaneously utilizing edge additions and deletions will be considered at the end of this chapter. The goal of the anonymization algorithm we want to design first is formulated in problem 9.1.

Problem 9.1 (Graph Anonymization)**.** *Given a graph $G = (V, E)$ and an integer k, find a k-degree anonymous graph $\widehat{G} = (V, \widehat{E})$ with $E \subseteq \widehat{E}$ such that $|\widehat{E} - E| = |\widehat{E}| - |E|$ is minimized.*

This problem always has a solution: The complete graph always satisfies the anonymity constraint, given that a sensible k has been chosen, i.e., $k < |V|$. Since we need to a find a graph $\widehat{G}$ that minimizes $|\widehat{E}| - |E|$, it will also minimize the L_1 distance between the degree sequence of G and $\widehat{G}$ since we have the equality

$$|\widehat{E}| - |E| = \frac{1}{2} L_1(\widehat{d}, d) \tag{9.1}$$

The equality is true because adding one edge to the graph increases the degree of two nodes, hence the fraction $\frac{1}{2}$ in the formula. Recall that the L_1 distance is defined as follows

$$L_1(\widehat{d}, d) = \sum_i \left| \widehat{d}(i) - d(i) \right| \tag{9.2}$$

Here d stands for the degree sequence of the graph G, and $\widehat{d}$ for that of $\widehat{G}$. This is a very interesting observation that can be exploited when designing an algorithm. Instead of immediately trying to modify the graph by adding edges, we can first find the degree sequence $\widehat{d}$ that is k-anonymous and minimizes the L_1 distance. As a second step we can modify the input graph making sure it has the desired k-anonymous degree sequence. We will continue by formalizing these two subproblems.

Part 1: Degree Anonymization

First, starting from d, we construct a new degree sequence $\widehat{d}$ that is k-anonymous and minimizes $L_1(\widehat{d}, d)$. From equation 9.1 we know that minimizing this distance is equivalent to the requirement in problem 9.1 stating that we must use as few edge additions as possible.

Problem 9.2 (Degree Anonymization)**.** *Given d, the degree sequence of graph $G = (V, E)$, and an integer k, construct a k-anonymous sequence $\widehat{d}$ by only increasing the degree of nodes, such that $L_1(\widehat{d}, d)$ is minimized.*

Part 2: Graph Construction

Having the optimal k-anonymous degree sequence $\widehat{d}$, the next step consists of constructing a graph with this degree sequence that closely resembles the input graph. If all edges of the input graph are included in the constructed graph this gives an optimal solution for k-degree anonymization defined in problem 9.1.

Problem 9.3 (Graph Construction)**.** *Given a graph $G = (V, E)$ and a k-anonymous degree sequence $\widehat{d}$, construct the graph $\widehat{G} = (V, \widehat{E})$ such that its degree sequence is equal to $\widehat{d}$ and $E \subseteq \widehat{E}$.*

9.4 Degree Anonymization

Problem 9.2 can be efficiently solved by a dynamic-programming algorithm having a time complexity of $O(n^2)$. We can further optimize this algorithm to obtain a time complexity of $O(nk)$. Hence we can obtain the optimal results within a linear amount of time. First we outline the algorithm running in time $O(n^2)$ because it's easier to explain, and then we will describe how to improve the algorithm.

Since we have restricted ourselves to edge additions only, the degree of the nodes can only increase. Furthermore we make the following observation.

Claim 9.2. *Let d be the degree sequence of a graph. Then if $d(i) = d(j)$ with $i < j$, it follows that $d(i) = d(i+1) = \ldots = d(j)$.*

Proof. The proof is by contradiction. Assume that there is a position ℓ with $i \leq \ell < j$ such that $d(\ell) < d(\ell+1)$. Because a degree sequence is monotonically increasing this implies that $d(i) < d(j)$, which leads to a contradiction. Hence there is no position in the subsequence where the degree increases, so all degrees must be equal. $\square$

Let $D_A(d)$ be the cost of anonymizing the degree sequence d, which is equal to $L_1(d, \widehat{d})$ where $\widehat{d}$ is the optimal solution as defined in problem 9.2. We will call $D_A(d)$ the *anonymization cost* of a sequence d for a given number k. Furthermore we will let $d[i,j]$ denote a subsequence of d containing the elements $d(i), d(i+1), \ldots, d(j)$. Additionally $I(d[i,j])$ is defined as

$$I(d[i,j]) \overset{\text{def}}{=} \sum_{\ell=j}^{j} (d(i) - d(\ell)) \tag{9.3}$$

which can be seen as the anonymization cost when all nodes $i, i+1, \ldots, j$ are put in the same anonymization group[1] and all degrees are set to $d(i)$.

Using these notations we can derive the set of dynamic programming equation which can be used to solve the Graph Anonymization program.

For $i < 2k$:

$$D_A(d[1,i]) = I(d[1,i]) \tag{9.4}$$

For $2k \leq i \leq n$:

$$D_A(d[1,i]) = \min\left\{ I(d[1,i]), \min_{k \leq t \leq i-k} \{D_A(d[1,t] + I(d[t+1,i]))\} \right\} \tag{9.5}$$

The first equation is correct because, when $i < 2k$, one cannot construct two different anonymized groups each of size k. Therefore all nodes in the subsequence are put into the same anonymization group, and all the degrees are set to $d(1)$.

For i larger or equal to $2k$ either the complete sequence can be put into one big anonymization group, or the sequence can be split up into two

[1]In an *anonymization group* all nodes have the same degree.

subsequences that are anonymized. The equation is set up such that, when a sequence is split up, both subsequences are guaranteed to have a length of at least k. Then the minimum is found over all possible positions where the sequences can be split, and also checks if not splitting at all might be optimal.

Implementing the algorithm is done by first initializing $I(d[i,j])$ for every $1 \leq i \leq j \leq n$, which has a time complexity of $O(n^2)$. Then equation 9.4 is implemented and simply assigns the correct value to every $D_A(d[1,i])$ for $i < 2k$ having only a time complexity of $O(k)$. Given that $k < n$ these two preprocessing steps take time at most $O(n^2)$. For the recursive formula 9.5 the algorithm considers every i starting at $2k$ and ending at n. Every step it calculates the minimum over at most $i - 2k + 1$ splits. This results in a total running time of $O(n^2)$.

9.4.1 An Improvement

Improving the running time of the algorithm can be done by exploiting the following observation.

Claim 9.3. *An anonymization group with a size larger than $2k - 1$ can be split into two anonymization groups without increasing the overall anonymization cost.*

Proof. Say the original anonymization group would be anonymized so all nodes were given degree ℓ. Then after splitting it into two anonymization groups, one can always anonymize them so that in both groups all the node degrees are again anonymized to ℓ. This will not change the anonymization cost and thus completes the proof. $\square$

Based on this claim we can change the dynamic programming equations. The first equation stays the same, but for the second equation we now know that the outermost minimum function can be removed, so only the innermost minimum function remains. Furthermore when considering all possible splits, we know that the second subsequence should have a size of at most $2k - 1$. All this results in the following recursive formula for i's satisfying $2k \leq i \leq n$:

$$D_A(d[1,i]) = \min_{\max\{k, i-2k+1\} \leq t \leq i-k} \{D_A(d[1,t] + I(d[t+1,i]))\} \qquad (9.6)$$

Analyzing the time complexity again starts at the initialization steps. To precompute $I(d[i,j])$ we no longer have to consider all the possible pairs of i and j, but instead can focus on sequences having length at least k and at most $2k - 1$. This is done by considering, for every i, only j's such that $k \leq j - i + 1 \leq 2k$. We conclude that the running time for the initialization step drops to $O(nk)$. Secondly the recursion formula 9.6 needs to consider only $O(k)$ splits for every i, totaling in a time complexity of $O(nk)$. This proves the following theorem.

Theorem 9.4. *There exists an algorithm solving the Degree Anonymization problem in linear time.*

We conclude that this part of the algorithm is efficiently solvable. The algorithm can quickly find the optimal solution, and implementing it also isn't a complex task. In subsequent sections we will call this algorithm `DegreeAnonymization`(d, k) where d is a degree sequence and k the privacy parameter. The function returns the optimal k-anonymous degree sequence using only edge additions.

9.5 Graph Construction

9.5.1 Realizability and ConstructGraph

Having a k-anonymous degree sequence the goal is now to construct a graph that closely resembles the input graph with the given anonymous degree sequence. It is not clear that a graph having the given degree sequence must exist. After all, not all possible degree sequences have a corresponding *simple* graph. We will call a degree sequence that can be achieved by constructing a simple graph *realizable.*

Definition 9.4 (Realizable)**.** *A degree sequence is realizable if and only if there exists a simple graph having precisely this degree sequence.*

Thanks to Erdős and Gallai [Cho86] we can use the following theorem to test whether a degree sequence is realizable or not.

Theorem 9.5 (Erdős-Gallai Theorem)**.** *A degree sequence d is realizable if and only if $\sum_i d(i)$ is even and for every $1 \leq \ell \leq n-1$ it holds that*

$$\sum_{i=1}^{\ell} d(i) \leq \ell(\ell-1) + \sum_{i=\ell+1}^{n} \min\{\ell, d(i)\} \tag{9.7}$$

The first requirement is easy to understand, since the total sum of all the degrees is equal to $2 \cdot |E|$ and thus always even. The second requirement states that for every subset of the ℓ highest-degree nodes we can draw enough edges between these nodes themselves, and to the "remaining nodes", such that the node can obtain its given degree. The proof of this theorem is by induction and gives rise to a natural algorithm for constructing a graph with the specified degree sequence [Cho86, MV02]. Algorithm 9.2 called `ConstructGraph` shows the details of this algorithm.

First `ConstructGraph` initializes the graph to be constructed by creating n isolated nodes. During each iteration it will pick a random node and add edges such that the selected node has the required degree. This is done by storing the *residual degree* of every node during the executing of the algorithm. Basically the residual degree stands for the number of edges that still have to be added to the node in question for it to obtain the required degree. The randomly picked node is connected to the nodes with the highest residual degree, making sure that inequalities of the Erdős-Gallai Theorem hold. If it turns out the degree sequence is not realizable the algorithm will give some of the nodes a negative residual degree, and will then return "No". On the other hand, once all the nodes have a residual degree of zero, the graph has been constructed and is returned.

Algorithm 9.2 The `ConstructGraph` algorithm.

```
Input: A degree sequence $d$ of length $n$.
Output: A graph $G = (V, E)$ with the given degree sequence $d$ or "No"
if $d$ is not realizable.
 1: $V := \{1, \dots, n\}, E := \emptyset$
 2: if $\sum_i d(i)$ is odd then
 3:     return "No"
 4: while 1 do
 5:     if $\exists i\colon d(i) < 0$ then
 6:         return "No"
 7:     if $\forall i\colon d(i) = 0$ then
 8:         return $G = (V, E)$
 9:     Pick a random node $v$ with $d(v) > 0$
10:     Set $d(v) = 0$
11:     $W$ := the $d(v)$-highest entries in $d$ other than $v$
12:     for each node $w \in W$ do
13:         $E := E \cup (v, w)$
14:         $d(w) := d(w) - 1$
15:     end for
16: end while
```

Note that algorithm 9.2 is an oracle for the realizability property of a degree sequence. It will return "No" if and only if the degree sequence is not realizable.

To analyze the running time of `ConstructGraph` let n be the number of nodes and $d_{\max} = \max_i d(i)$. One can store all the residual degrees in a hash table with the key being the residual degree. When the node of a degree changes updating the hash table takes only constant time. Also note that checking if a residual degree is negative can be done while updating it, so we can avoid having to check all the values each iterations. Detecting if all residual degrees are zero can be done by saving the currently highest residual degree. By using the current maximum residual degree we can also efficiently find the current $d(v)$-highest residual degrees. Finally a node is at most updated $d_{\max}$ times, giving an overall time complexity of $O(nd_{\max})$.

A possible variation of the algorithm can pick the highest degree node at each iteration. This would result in a topology having a very dense core. On the other hand, starting with the lowest degree, would result in very sparse cores. The middle ground is obtained by picking a random node. All these variations would still have the same time complexity.

9.5.2 SuperGraph: Resemblance to the Input Graph

If one would use `ConstructGraph` to construct the graph of a k-anonymized degree sequence of the input graph, it could different significantly from the given input graph. This it not want we want. What we need is an algorithm that finds a solution to the Graph Construction problem. That is, the constructed graph $\widehat{G} = (V, \widehat{E})$ must satisfy the constraint $E \subseteq \widehat{E}$, where $G = (V, E)$ is the input graph. For this we first introduce a new definition.

Definition 9.5 (Realizable Subject to a Graph)**.** *Given an input graph $G = (V, E)$, a degree sequence $\widehat{d}$ is realizable subject to G if and only if there exists a simple graph $\widehat{G} = (V, \widehat{E})$ having precisely the degree sequence $\widehat{d}$ and with $E \subseteq \widehat{E}$.*

Given the above definition one has the following variation of the Erdős-Gallai Theorem [LT08].

Theorem 9.6. *Given a degree sequence $\widehat{d}$ and a graph $G = (V, E)$ with degree sequence d. Let*

1. *$a = \widehat{d} - d$*
2. *V_ℓ be the ordered set of ℓ nodes containing the ℓ largest $a(i)$ values sorted in decreasing order*
3. *$d^\ell(i)$ be 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*

we have that if *$\widehat{d}$ is realizable subject to G* then *$\sum_i a(i)$ is even and for every $1 \leq \ell \leq n-1$ it holds that*

$$\sum_{i \in V_\ell} a(i) \leq \sum_{i \in V_\ell} \left(\ell - 1 - d^\ell(i)\right) + \sum_{i \in V - V_\ell} \min\left\{\ell - d^\ell(i), a(i)\right\} \qquad (9.8)$$

Notice that for every pair of nodes (u, v) with $u \in V_\ell$ and $v \in V - V_\ell$ it's always true that $a(u) \geq a(v)$ and that $|V_\ell| = \ell$. Important is that the theorem only gives an implication. It gives no guarantees that if the formula is true, the degree sequence is realizable subject to the graph. In other words the theorem only gives a necessary condition, it might not be a sufficient one. The authors have not published a proof [LT08]. Nonetheless, to gain some insight in the theorem, it's helpful to convince yourself that the following two inequalities always hold:

$$\forall i\colon d^{\ell}(i) \leq \ell \tag{9.9}$$

$$\forall i \in V_{\ell}\colon d^{\ell}(i) \leq \ell - 1 \tag{9.10}$$

These make sure that the terms in the sum are never negative.

Even though theorem 9.6 gives no sufficient condition, we can still design an algorithm similar to `ConstructGraph` that will attempt to construct a graph that is a supergraph of the input graph and has the desired degree sequence. It can first check if inequality 9.8 holds for all appropriate ℓ. If it doesn't it returns "No" signaling that no such supergraph exists. Otherwise it will attempt to find the supergraph and return it, or return the value "Unkown" signaling that it didn't find a supergraph but one might nevertheless exist.

Algorithm 9.3 The `SuperGraph` algorithm.

```
Input: A degree sequence $\widehat{d}$ of length $n$ and a graph $G = (V, E)$.
Output: A graph $\widehat{G} = (V, \widehat{E})$ with degree sequence $\widehat{d}$ and $E \subseteq \widehat{E}$,
"Unknown" if no supergraph found, "No" if $\widehat{d}$ is not realizable s.t. to $G$.
1: $a := \widehat{d} - d$, with $d$ the degree sequence of $G$
2: if $\sum_i a(i)$ is odd or equation 9.8 doesn't hold then
3:     return "No"
4: $\widehat{E} := E$
5: while true do
6:     if $\exists i\colon a(i) < 0$ then
7:         return "Unknown"
8:     if $\forall i\colon a(i) = 0$ then
9:         return $\widehat{G} = (V, \widehat{E})$
10:    Pick a random node $v$ with $a(v) > 0$
11:    Set $a(v) = 0$
12:    $W :=$ the $a(v)$-highest entries in $\widehat{d}$ other than $v$, ignoring nodes that
       are already connected to $v$ in $\widehat{G}$
13:    for each node $w \in W$ do
14:        $\widehat{E} := \widehat{E} \cup (v, w)$
15:        $a(w) := a(w) - 1$
16:    end for
17: end while
```

Algorithm 9.3 gives the pseudo code of `SuperGraph`. The algorithm works on the *residual degrees* $a = \widehat{d} - d$. These are the degrees that must

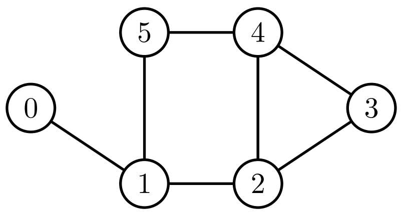


Figure 9.2: Example graph to be 2-degree anonymized.

be added in order to obtain the desired degree $\widehat{d}$. First it checks whether the conditions of theorem 9.6 are met, if not it outputs "No". Otherwise it attempts to add edges by first picking a random node v having $a(v) > 0$. It then adds edges to the $a(v)$-highest residual degree nodes while ignoring nodes that are already connected to v in G. If not enough nodes can be found satisfying this condition, the algorithm will return "Unknown". The remaining details of the algorithm are similar to `ConstructGraph`. If the algorithm returns "Unknown" it doesn't mean that a graph satisfying the required condition doesn't exists, it merely means that our algorithm couldn't find it.

Concerning the time complexity we can use the same datastructures as in `ConstructGraph`. With $a_{\max} = \max_i a(i)$ this gives a time complexity of $O(na_{\max})$. Here we assume that checking if an edge already exists can be done in constant time. This is possible when using appropriate data structures, e.g., a hash table containing the existing edges.

There is one subtlety we haven't addressed yet. This is best explained using an example. Say we want to 2-degree anonymize the graph shown in figure 9.2. The degree sequence d of the graph is $[3, 3, 3, 2, 2, 1]$. The 2-degree anonymization $\widehat{d}$ is $[3, 3, 3, 3, 2, 2]$. The vector of residual degrees a becomes $\widehat{d} - d = [0, 0, 0, 1, 0, 1]$. Adding an edge between both nodes having residual degree 1 could complete the anonymization (if this edge doesn't exists already). But we have a problem: We no longer know the node the residual degree belongs to! The two residual degrees of 1 are located at indices 3 and 5 in the vector a. However, these indices of the residual degrees do not correspond to nodes, as can be clearly seen in figure 9.2. Fortunately this can be easily solved by deviating from the formal definition of a degree sequence, and pairing each degree with its node. The vector a then contains pairs (b, v) where b is the residual degree and v represent the node. Slightly abusing notation, in our example a would be equal to

$$\begin{aligned}[3,3,3,3,2,2] - [(3,1),(3,2),(3,4),(2,3),(2,5),(1,0)] \\ = [(0,1),(0,2),(0,4),(1,3),(0,5),(1,0)]\end{aligned}$$

Now we do know the nodes where an edge could be added between. Based on the calculated value of a is would mean adding the edge $(0, 3)$.

And indeed, adding $(0, 3)$ makes the graph 2-degree anonymous. However adding this edge might come as a surprise. If you look at figure 9.2 visual inspection of the graph clearly shows that adding the edge $(0, 5)$ also makes the graph 2-degree anonymous. Thus it's possible there are multiple options that anonymize the graph and minimize the anonymization cost. Depending on how we order nodes with the same degree, different solutions can be found. If the nodes would be sorted differently, say in decreasing order:

$$\begin{aligned}[3, 3, 3, 3, 2, 2] - [(3, 4), (3, 2), (3, 1), (2, 5), (2, 3), (1, 0)] \\ = [(0, 4), (0, 2), (0, 1), (1, 5), (0, 3), (1, 0)]\end{aligned}$$

we see the other solution pops up which adds the edge $(0, 5)$.

A New Improvement?

In general the way we order nodes with the same degree can not only produce different solutions, sometimes it means the difference between finding and *not* finding a solution. For example, we can sort nodes with the same degree in increasing order of their node id, which could result in the following sequence

$$[(3, 4), (3, 2), (3, 1), (2, 5), (2, 3), (1, 0)]$$

Or we could sort them in decreasing order, giving rise the to sequence

$$[(3, 1), (3, 2), (3, 4), (2, 3), (2, 5), (1, 0)]$$

How we sort nodes having the same degree can impact the realizability of a degree sequence subject to the input graph. Therefore one might propose the following improvement: The order in which nodes with the same degree are put in the degree sequence should be randomized. The `ConstructrGraph` algorithm is then run several times so multiple orderings are tested. Unfortunately, after several tests on different graphs, we concluded that the performance increase of this modification is very low.

9.5.3 The Probing Scheme

The advantage of `SuperGraph` is that *if* it returns a graph, we know that the least amount of edge additions were made. The disadvantage is that the algorithm doesn't always return a graph (the same is true for `ConstructGraph`). This is not want we want. We want to return a k-degree anonymous graph that closely resembles the input graph. If this would require slightly more edge additions than calculated by `DegreeAnonymization`, this is a penalty we must pay. After all, releasing no graph whatsoever is not an option.

A simple solution, called the probing scheme, can be used. If no graph was found it slightly changes the degree sequence of the input graph G, recomputes the k-anonymous degree sequence $\widehat{d}$, and then runs the `SuperGraph` algorithm on the new degree sequence $\widehat{d}$. It continues doing this until a graph has been found. Although rather simple it has been claimed to work fairly well in practice. We will extensively study the effectivity of the probing scheme in section 9.7.2. The details are outlined in algorithm 9.4.

Algorithm 9.4 The `Probing` scheme.

```
    Input: A graph $G = (V, E)$ and integer $k$.
    Output: A $k$-degree anonymous graph $\widehat{G} = (V, \widehat{E})$ with $E \subseteq \widehat{E}$.
 1: $d :=$ the degree sequence of $G$
 2: $\widehat{d} :=$ DegreeAnonymization($d, k$)
 3: $\widehat{G} :=$ SuperGraph($\widehat{d}, G$)
 4: while $\widehat{G}$ is not a graph do
 5:     $d :=$ AddRandomNoise($d$)
 6:     $\widehat{d} :=$ DegreeAnonymization($d, k$)
 7:     $\widehat{G} :=$ SuperGraph($\widehat{d}, G$)
 8: end while
 9: return $\widehat{G}$
```

The most important step is the `AddRandomNoise` function. The precise implementation determines how well the probing scheme works, and whether it will work at all. First it must add random noise in such a manner so eventually a graph will always be found. Adding noise to a random location while avoiding that the degree doesn't exceed $|V| - 1$ achieves this requirement. Eventually the degree sequence d will converge to the complete graph which is always a solution to the anonymization problem. We can conclude that the probing scheme always halts.

Secondly the question is where to best add noise. A good heuristic is to increase the degree of low-degree nodes. The reasoning behind this is that during the degree anonymization of the degree sequence, nodes having a high degree are put into the same anonymization group. Rarely will these nodes have the same degree, forcing the anonymization step to increase the degree of the other nodes with a possibly large amount. In turn this potentially large increase must be compensated by increasing the degree of low-degree nodes, which the degree anonymization algorithm does not guarantee. Therefore the `AddRandomNoise` function should be baised towards increasing the degree of low-degree nodes.

Now, the degree sequence of most graphs have many low degree entries at the tail of the sequence. So when picking the node of which we want to increase the degree, simply picking a position uniform at random gives the desired bias towards increasing low degree nodes. The resulting implementation is shown in algorithm 9.5.

Deciding with how much to increase the selected degree is also a crucial step. Setting this value too low could greatly increase the running time of the algorith (in algorithm 9.5 this value is maxDelta). On the other hand,

Algorithm 9.5 The `AddRandomNoise` function.

```
    Input: A degree sequence $d$ of length $n$.
    Output: A slightly modified degree sequence.
1:  pos := pick a number from $[1, n]$ uniform at random
2:  delta := pick a number from $[1, \text{maxDelta}]$ uniform at random
3:  $d(\text{pos}) := \text{Min}(n - 1, d(\text{pos}) + \text{delta})$
4:  return Sort($d$)
```

picking a too big value could lead to unnecessary many edge additions, and would produce a result far from optimal. In our implementation we manually selected an appropriate value for each graph. It might be possible to automatically select the value by inspecting the highest, lowest and average degree.

It's regrettable that these details, which all influence the performance of the algorithm, are not discussed by Lui and Terzi in their paper.

9.5.4 Relaxed Graph Construction using GreedySwap

The `SuperGraph` algorithm creates a k-degree anonymous graph such that $E \subseteq \widehat{E}$. This can be a strong requirement, and we're actually looking for a graph that looks like the input graph, it doesn't *have* to be a supergraph of the input graph. So the real goal is to find a graph such that $\widehat{E} \approx E$. Since not all edges of E need to be present in the anonymous graph, this implicitly allows both edge additions and deletions. We will come back on simultaneous edge additions and deletions in the next section.

To avoid multiple modifications to the algorithm at once, we will still use the `DegreeAnonymization` algorithm designed only for edge additions. Later on we will modify it to also allow edge deletions. So assume that the algorithm `DegreeAnonymization` returns a k-anonymous degree sequence $\widehat{d}$ and that it's realizable such that `ConstructGraph` will return a graph $\widehat{G}_0$. The goal of the utility function called `GreedySwap` is, given the input graph G and a k-degree anonymous graph $\widehat{G}_0$, to transform the graph G_0 as close as possible to the input graph, without modifying its degree sequence. The result would be an anonymous graph that better resembles the input graph. Put differently `GreedySwap` will transform $\widehat{G}_0$ into $\widehat{G} = (V, \widehat{E})$ which is still k-anonymous and with $\widehat{E} \approx E$.

Because the degree sequence of $\widehat{G}_0$ is not allowed to change we can only apply transformations to edges that do not change the degree of any node in the graph. Such modifications are called *valid swaps.*

Definition 9.6 (Valid Swap)**.** *Given a graph $G_i = (V, E_i)$ a valid swap starts with the nodes $i, j, k, l \in V$ such that $(i, j) \in E_i$ and $(k, l) \in E_i$. Two valid swaps might be possible depending on the following conditions:*

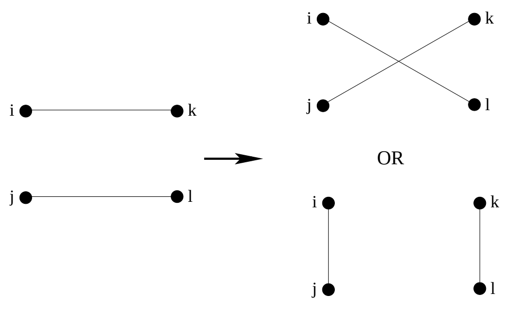


Figure 9.3: Illustration of valid swap operations [LT08]

If $(i,j) \notin E$ *and* $(k,l) \notin E$

$$E_{i+1} := E_i - \{(i,k),(j,l)\} \cup \{(i,j),(k,l)\} \tag{9.11}$$

If $(i,l) \notin E$ *and* $(j,k) \notin E$

$$E_{i+1} := E_i - \{(i,k),(j,l)\} \cup \{(i,l),(j,k)\} \tag{9.12}$$

The valid swaps given in the previous definition are illustrated in figure 9.3. Important is that the valid swaps can change the structure of the graph, but they do not change the degree sequence of the graph.

`GreedySwap` now finds valid swaps with the maximum increase in edge intersection, i.e., it finds a swap such that

$$|E \cap \widehat{E}_{i+1}| - |E \cap \widehat{E}_i|$$

is maximized, and continues until no more valid swaps are found. The utility function `FindMaxSwap` is used to find precisely these valid swaps. It returns the increase in edge intersection and the swap belonging to this increase. As shown in algorithm 9.6 `GreedySwap` will greedily keep swapping edges until there are no more valid swaps that can increase the size of the edge intersection.

Using `GreedySwap` it's now possible to implement our original idea: Given the input graph G we find the k-anonymous degree sequence, construct a graph with this degree sequence using `ConstructGraph`, and finally transform this graph making sure it resembles the input graph. In the case the degree sequence $\widehat{d}$ is not realizable, we can invoke a `Probing` scheme identical to the one we have described in algorithm 9.4. The final process is given in algorithm 9.7.

To calculate the time complexity we will ignore the probing scheme since it's hard to find usable upper bounds for it. However, remember that it converges to the complete graph and hence always terminates. A

Algorithm 9.6 The `GreedySwap` algorithm.

```
    Input: Input graph $G = (V, E)$ and initial graph $\widehat{G}_0 = (V, \widehat{E}_0)$.
    Output: A $k$-degree anonymous graph $\widehat{G} = (V, \widehat{E})$ with $\widehat{E} \approx E$.
1: $\widehat{G}(V, \widehat{E}) := \widehat{G}_0(V, \widehat{G}_0)$
2: $(c, (e_1, e_2, e'_1, e'_2)) :=$ FindMaxSwap($\widehat{G}$)
3: while $c > 0$ do
4:     $\widehat{E} := \widehat{E} - \{e_1, e_2\} \cup \{e'_1, e'_2\}$
5:     $(c, (e_1, e_2, e'_1, e'_2)) :=$ FindMaxSwap($\widehat{G}$)
6: end while
7: return $\widehat{G}$
```

Algorithm 9.7 The `RelaxedConstruction` algorithm.

```
    Input: A graph $G = (V, E)$ and integer $k$.
    Output: A $k$-degree anonymous graph $\widehat{G} = (V, \widehat{E})$ with $\widehat{E} \approx E$.
1: $d :=$ degree sequence of $G$
2: $\widehat{d} :=$ DegreeAnonymization($d, k$)
3: $\widehat{G}_0 :=$ ConstructGraph($\widehat{d}$)
4: while $\widehat{G}_0$ is not a graph do
5:     $d :=$ AddRandomNoise($d$)
6:     $\widehat{d} :=$ DegreeAnonymization($d, k$)
7:     $\widehat{G}_0 :=$ ConstructGraph($\widehat{d}$)
8: end while
9: return $\widehat{G} =$ GreedySwap($\widehat{G}_0, G$)
```

naive implementation of FindMaxSwap would result in a time complexity of $O(|E|^2) = O(n^4)$. Letting I be the number of iterations the Greedy Swap algorithm needs, the total time complexity would be $O(In^4)$. This is unacceptable for large input graphs. Instead we will sample only $O(\log|E|)$ edges in the FindMaxSwap utility function. As a result we are no longer guaranteed that it will find the real maximum swap, but it still gives good results in practice. Also note that the GreedySwap is by itself already an approximation algorithm. The running time becomes $O(I\log^2 n)$.

An important comment remains: Similar to the SuperGraph algorithm it's of importance how we decide which nodes in the graph returned by ConstructGraph correspond to which nodes in the input graph. Again this can be solved by slightly deviating from the formal definition of a degree sequence and pairing each degree with its node. This can then be used in the ConstructGraph algorithm to create the mapping as specified by this degree sequence.

9.5.5 SimultaneousSwap: Edge Additions and Deletions

Finally we move to our algorithm allowing both edge additions and deletions at all steps. First we must modify the DegreeAnonymization algorithm. Surprisingly it's very easy to include edge deletions and additions at the same time. All we need to update is $I(d[i,j])$ denoting the anonymization cost when all nodes $i, i+1, \ldots, j$ are put in the same anonymization group. It's now equal to

$$I(d[i,j]) \overset{\text{def}}{=} \sum_{\ell=i}^{j} |d^* - d(\ell)|$$

where d^* is

$$d^* \overset{\text{def}}{=} \arg\min_{d} \sum_{\ell=i}^{j} |d^* - d(\ell)|$$

The value of d^* has been proven to be equal to the median of the values $\{d(i), \ldots, d(j)\}$ [Lee95]. Thus calculating $I(d[i,j])$ can be done in linear time. The recursion equations stay exactly the same. So when using the improvement explained in section 9.4.1 the time complexity of calculating the optimal k-anonymous degree sequence is $O(nk)$. We will call this algorithm OptimalDegreeAnon.

Combined with a probing scheme, ConstructGraph and GreedySwap this results in algorithm 9.8. First it uses the probing scheme to construct a k-degree anonymous graph. Then it transform the graph to more closely resemble the input graph. We call it the SimultaneousSwap algorithm.

Algorithm 9.8 The `SimultaneousSwap` algorithm.

```
    Input: A graph $G = (V, E)$ and integer $k$.
    Output: A $k$-degree anonymous graph $\widehat{G} = (V, \widehat{E})$ with $\widehat{E} \approx E$.
1: $d :=$ degree sequence of $G$
2: $\widehat{d} :=$ OptimalDegreeAnon($d, k$)
3: $\widehat{G}_0 :=$ ConstructGraph($\widehat{d}$)
4: while $\widehat{G}_0$ is not a graph do
5:     $d :=$ AddRandomNoise($d$)
6:     $\widehat{d} :=$ OptimalDegreeAnon($d, k$)
7:     $\widehat{G}_0 :=$ ConstructGraph($\widehat{d}$)
8: end while
9: return $\widehat{G} =$ GreedySwap($\widehat{G}_0, G$)
```

9.6 Experiments and Implementation: Prologue

All algorithms are implemented in Java and accompanied with appropriate unit tests. This should assure that the algorithms function correctly. Although other papers on anonymization tend to use Python, we have chosen Java because of its speed advantages. This allows use to test the algorithms on large graphs within an acceptable amount of time.

9.6.1 Datasets

To test the anonymization algorithms several datasets have been used. We will briefly describe how we obtained them and how they were created.

Enron graph

During the Enron[2] scandal the Federal Energy Regulatory Commission released all the emails that had been send between employers of Enron. From these correspondences a social graph has been created. A link is introduced between two people only if both have sent emails to each other, and if in total they have send more than 5 messages to each other.

A cleaned MySql database created by Shetty and Abidi [SA04] was used in our experiments. From this we manually derived the social graph. At the time of writing the dataset was available at `http://www.isi.edu/~adibi/Enron/Enron.htm`.

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. [LT08] A visualization of the dataset is shown in figure 9.4.

[2]The Enron Corporation was an American energy company.

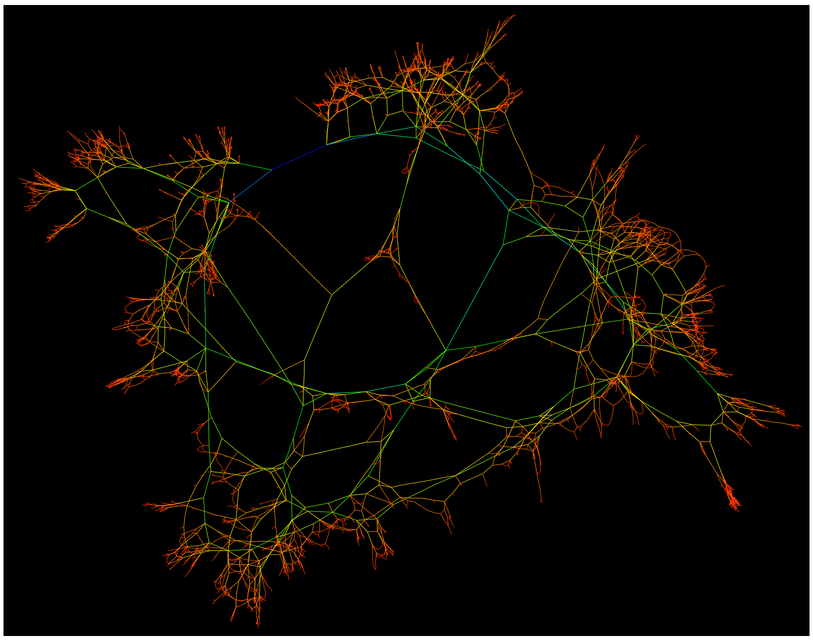


Figure 9.4: Powergrid graph visialization by AT&T Research.

At the time of writing the dataset was available at `http://www.cise.ufl.edu/research/sparse/matrices/Pajek/USpowerGrid.html` as a matrix saved in Matlab. A small utility function was written to convert the file into a format our own application could better handle.

Facebook100 dataset

In a study on the social structure of facebook networks, Traud, Mucha and Porter where given an anonymous dataset directly by Adam D'Angelo of Facebook [TMP11]. It contains the friendship graphs of 100 American colleges and universities as they existed in September 2005. Even if users didn't publish their friend list on facebook at the time, this dataset *does* contain all their friends! Thus it's a very accurate graph representing a real social network. Later on the Facebook Data Team requested that researchers should no longer distribute the dataset. However, copies of it still float around on the internet. Given that this graph contains invaluable information we still decided to use it. This also allows us to test the algorithm on *real* social networks, and not publicly scraped ones, or on randomly generated ones.

For our analysis we picked three graphs: The Texas graph, the Berkeley and the American graph. Later on the graphs Caltech and Bowdoin where also included. These are two smaller graphs that will be used when analyzing more computationally demanding algorithms. See table 9.1 for basic properties about these graphs. Looking at the amount of edges of the first three mentioned graphs, they are 10 to 100 times bigger compared to the largest graphs used in the tests by Lui and Terzi, and even though the Caltech and Bowdoin graphs are of a more moderate size, they're still 2 to 8 times bigger.

Co-authors graph

From the DBLP Computer Science Bibliography it's possible to derive the so called *co-authors graph*. In this graph each node represents an author, and an edge is drawn between authors that have co-authored a paper. This is a very large graph containing more than 1 million nodes and more than 3.6 million edges. This makes it too big to be used to evaluate more computationally demanding algorithms. However, when possible, a limited number of tests were also performed on this graph.

An XML file containing the dataset was downloaded from `http://www.informatik.uni-trier.de/~ley/db/`. The file is almost 1GB large and had to be parsed using a SAX parser to avoid running out of memory.

Graph Name	#Nodes	#Edges
Enron	151	502
Caltech	768	16,656
Bowdoin	2,251	84,387
Powergrid	4,939	6,594
American	6,386	217,662
Berkeley	22,937	852,444
Texas	36,371	1,590,655
Co-authors	1,048,435	3,663,990

Table 9.1: Basic graph statistics.

9.6.2 Testing Randomized Algorithms

It's very important to realize that most of our algorithms are *randomized*. For example, `ConstructGraph` picks a random node at every iteration. As another example, during the probing scheme a random amount of noise is added to a random location in the degree sequence. This complicates our analysis of the performances of the algorithms, since the output can change every run. Strangely Lui and Terzi don't mention how they dealt with this inherent randomness, raising questions how they performed their analysis. Either they reported the average value over an unknown number of runs, or only ran the algorithm once.

Only reporting the average value returned by a randomized algorithm actually gives us little information about the performance of it! [Sha] This is easily demonstrated by considering two randomized algorithms who have

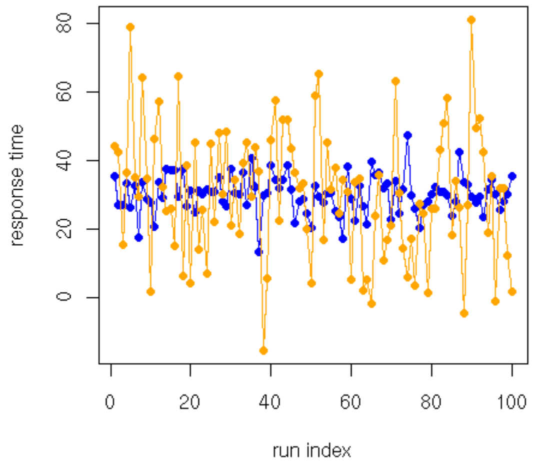


Figure 9.5: This example shows the fictive response time of two servers for 100 request. Both servers have the same average response time, but a different variance. This shows reporting only the average is not sufficient. [Sha]

the same average return value. The problem is that the *variance* of both algorithms can differ significantly. Here the algorithm with a lower variance performs better. A visual example is shown in figure 9.5. Both samples have the same average but a different variance. An algorithm with a lower variance is preferable as its results are more predictable.

During our tests we will report both the average and variance over a selected number of runs. The number of runs is chosen such that the total running time remains within practical limits. This gives a much better understand on the performance of the algorithm instead of just reporting the average value. On the other hand, if the variance is low, we will omit it in the graphs.

9.7 Individual Algorithm Evaluation

First we look at how the algorithms behave on their own. Particularly this evaluates the influence of the random steps that are taken in the algorithms, if they contain any at all.

9.7.1 Degree Anonymization

Implementing the Degree Anonymization algorithm was first done in Python and later on ported to Java. The reason for switching to java was to increase the speed of the algorithm, which made it possible to test it on larger and more realistic graphs within an acceptable amount of time.

Both `DegreeAnonymization` and `OptimalDegreeAnon` always return the optimal result. Nevertheless it is interesting to know what the anonymization cost is for certain graphs. Figure 9.6 shows the L1-distance between the original degree sequence and that obtained by `DegreeAnonymization` and `OptimalDegreeAnon`. We immediately see that allowing both edge deletions and additions greatly reduces the anonymization cost.

In figure 9.7.1 the anonymization cost when running the `OptimalDegreeAnon` algorithm are shown for the American and Powergrid graph. This illustrates that some graphs are easier to anonymize than others.

9.7.2 SuperGraph using Probing Scheme

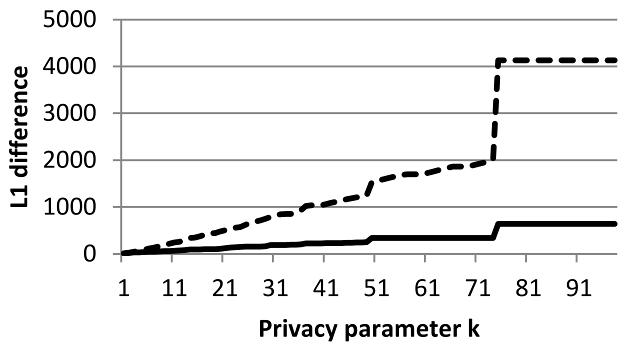
We continue by evaluating the performance of the `SuperGraph` algorithm using the probing scheme. The precise algorithm is shown in algorithm 9.4 with maxDelta set to 10. First the L1 distance between $\widehat{d}$ and $\widehat{d'}$ is measured, where $\widehat{d}$ is the degree sequence returned by `DegreeAnonymization` and $\widehat{d'}$ the degree sequence of the graph returned by algorithm 9.4. In effect this measures how much the probing scheme, and in particular the implementation of `AddRandomNoise`, have to modify the degree sequence in order for it to be realizable with respect to the input graph. For each value of k the algorithm was run 20 times on the input graph. Figure 9.8 shows the average L1 distance in a full line, and to visualize the variance the the dotted line shows the variance added to the average. A low variance then means both lines are close to each other, while a high variance increases the distance between both lines.

For the Enron and Powergrid graphs we notice the anonymization parameter k has little effect on the result. If we look at the graphs of Caltech and Bowdoin we see that that k does influence the result. The higher k, the higher the solution differs from the optimal one. In turn this means the probing scheme needs to make more and more iterations before it finds a solution. Lui and Terzi already warned that *in most cases* the probing scheme only needs to be invoked a few times. However, it appears that for real social networks, the current implementation of the probing scheme will not scale well to large graphs and values of k. For the even bigger Bowdoin graph it became too time consuming to test for all values of k, hence only the first few values are shown. Social networks can be much bigger than these example graphs, raising questions on the performance of the probing scheme in combination with the `SuperGraph` algorithm.

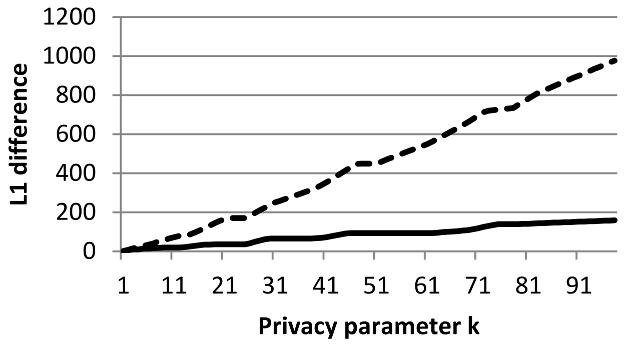
Probing Scheme 2

We attempted to improve the probing scheme by making the following modifications to the `AddRandomNoise` function:

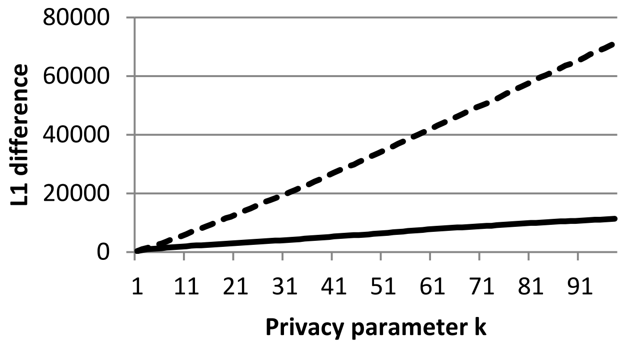
- If the sum of degrees is uneven, increase the degree of a random node by only one.



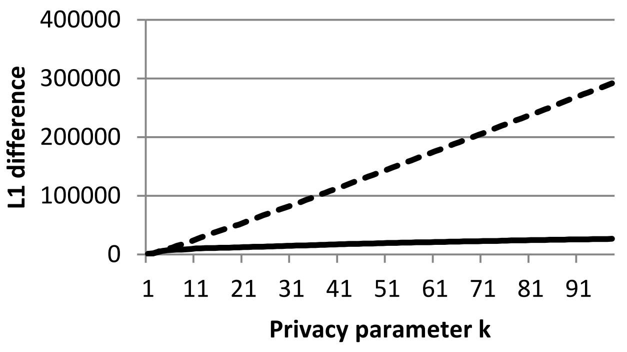
(a) Enron graph



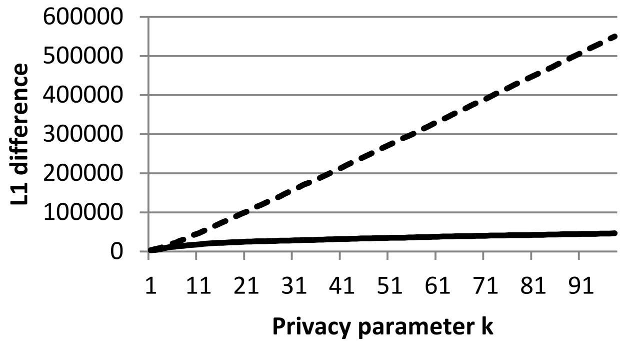
(b) Powergrid graph



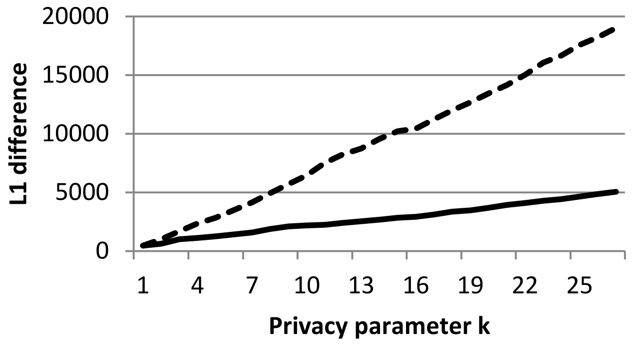
(c) American graph



(d) Berkeley graph



(e) Texas graph



(f) Co-authors graph

Figure 9.6: L1-distance of original degree sequence and that obtained by `DegreeAnonymization`, shown in a dashed line. And L1 distance between original degree sequence and that obtained by `OptimalDegreeAnon`, shown in a full line. X-axis stands for the privacy parameter k and the y-axis for the L1-distance.

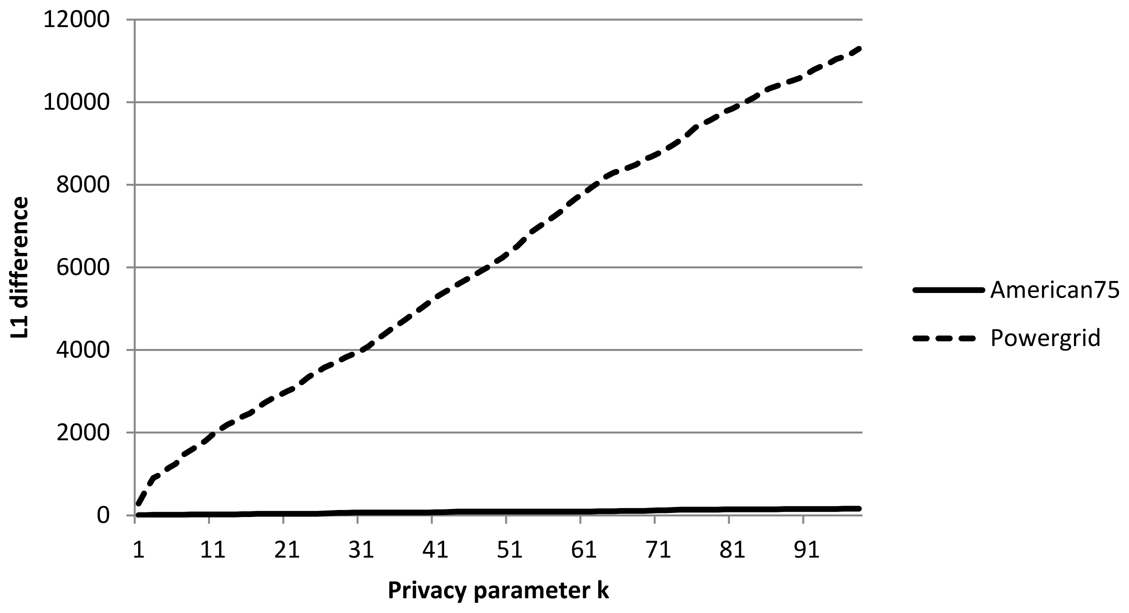
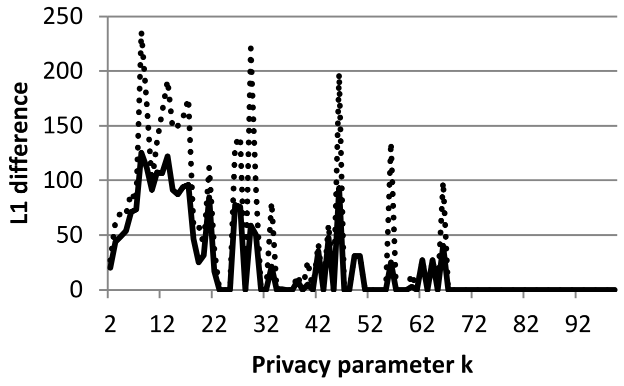
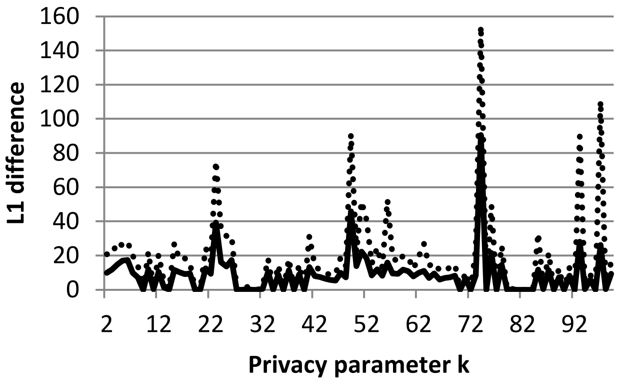


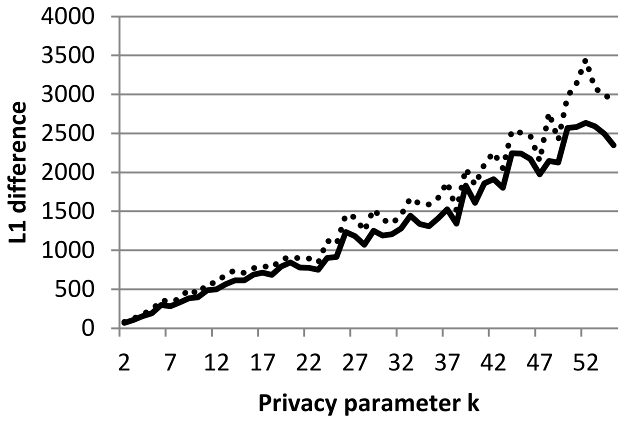
Figure 9.7: Comparison of L1-distance between original degree sequence and that obtained by `OptimalDegreeAnon` for two different graphs, showing that some graphs are easier to anonymize.



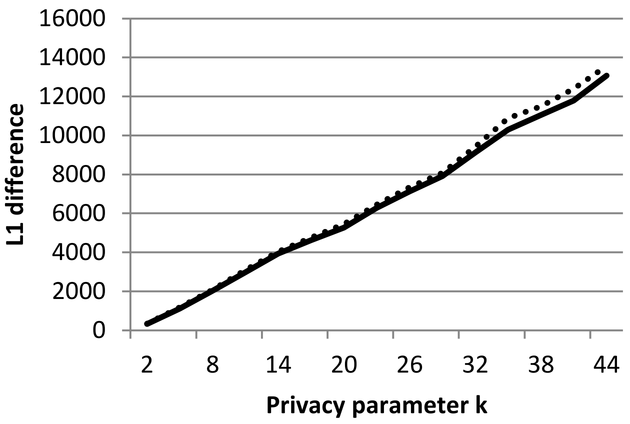
(a) Enron graph



(b) Powergrid graph



(c) Caltech graph



(d) Bowdoin graph

Figure 9.8: The full line is the average L1-distance between the degree sequence returned by `OptimalDegreeAnon` and that of the graph returned algorithm 9.4 over 20 runs. The dotted line is obtained by adding the variance to the full line.

- If the sum of degrees is even, increase the degree of a random node by an event amount.

In retrospect these might seem like obvious improvements. We will call this modification *probing scheme 2*, and the original version *probing scheme 1*. Figure 9.9 shows the results when using these modifications. We notice that not much has changed. Both the results and the variance are roughly the same. For low values of k such as 2 and 3 the L1 difference is actually much higher! A comparison between the old and new probing scheme is shown in figure 9.10 for the Powergrid and Caltech graph. Here we see that for some values of k probing scheme 2 is better, and for some values it's actually worse.

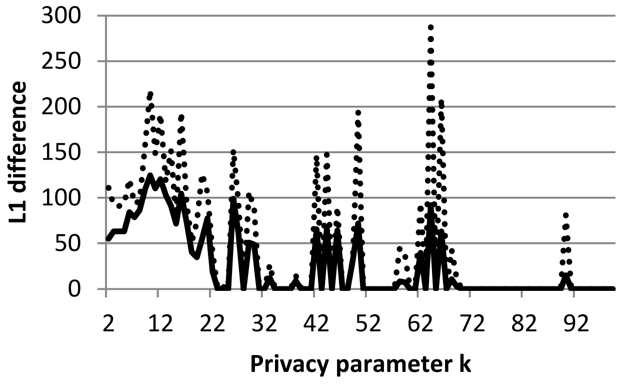
Probing Scheme 3

When running the tests a more drastic modification was also tried: We modified `AddRandomNoise` such that it increased the degree of the last node in the degree sequence by one. We will call it *probing scheme 3*. A predicted downside is that this will increase the running time of the algorithm. The first observation is that on all tested graphs the variance is much lower, making it impossible to visually see the dotted line, and therefore it's no longer included in any graphs dealing with probing scheme 3. A comparison of the L1 distance between the optimal degree sequence and that of the returned graph is shown in figure 9.11 with as input the Powergrid and Caltech graphs. We observe that when using probing scheme 3 on average less edge additions have to be used. A disadvantage is that because only a low amount of noise is added, on average the algorithms take a longer time to execute, as we already predicted. From the Caltech graph however, we observe something very interesting. For most values of k the result is indeed better, yet there are some outliers for which the returned result is actually worse!

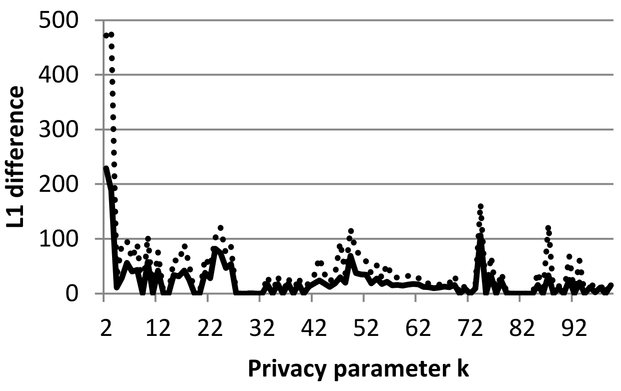
Probing Scheme 4

Our last attempt at improving the probing scheme is to increase the degree of a random node located in the *second half* of the degree sequence by one. The idea is that this will still reduce the variance and decrease the L1 distance as we have seen in probing scheme 3, but that it hopefully reduces the peaks where it performed worse than probing scheme 1 and 2. Figure 9.12 shows the results. The variance is higher than probing scheme 3, but lower than scheme 1 and 2. Figure 9.13 shows a comparison between the probing schemes for the Caltech graph. The accuracy is better than scheme 1 and 2 and has less outliers than scheme 3.

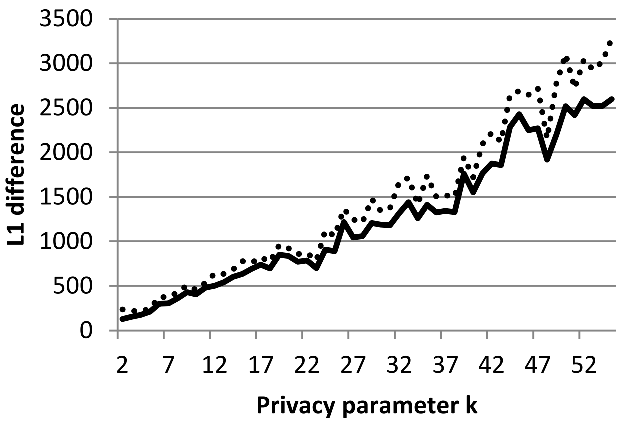
We conclude that improving the probing scheme appears to be a tedious task, and that for some graphs many iterations are needed before a result is



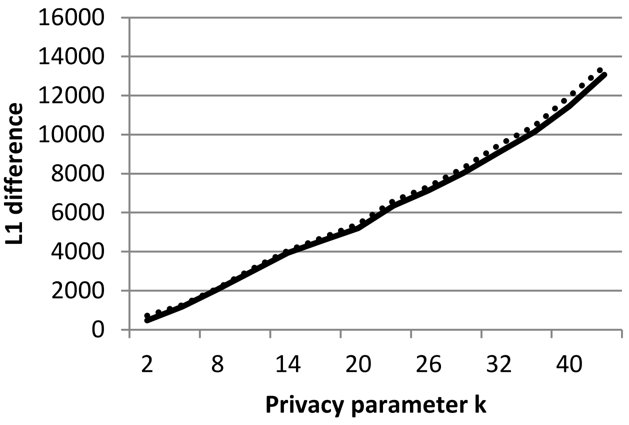
(a) Enron graph



(b) Powergrid graph

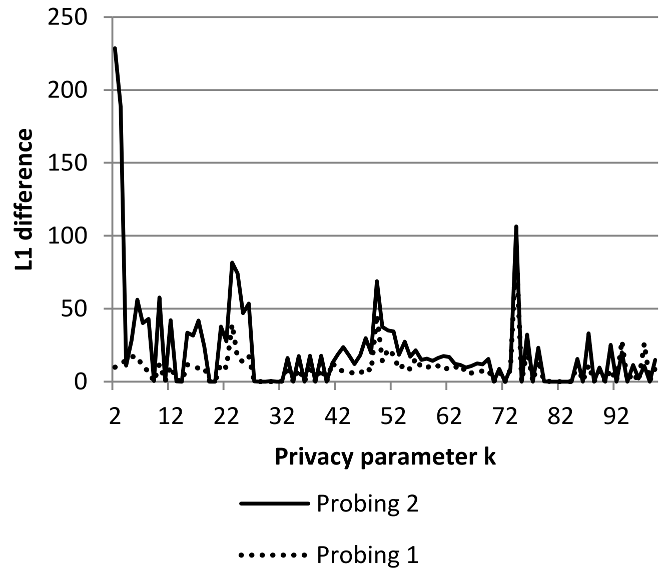


(c) Caltech graph

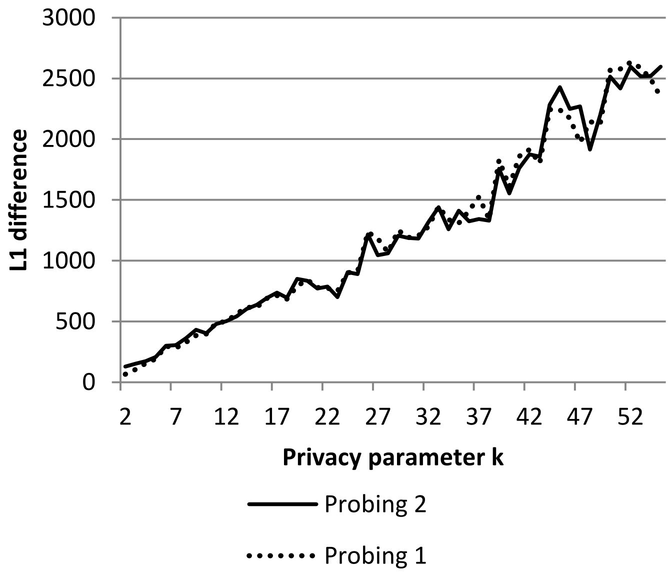


(d) Bowdoin graph

Figure 9.9: The full line is the average L1-distance between the degree sequence returned by `OptimalDegreeAnon` and that of the graph returned algorithm 9.4 using *probing scheme 2* over 20 runs. The dotted line is obtained by adding the variance to the full line.

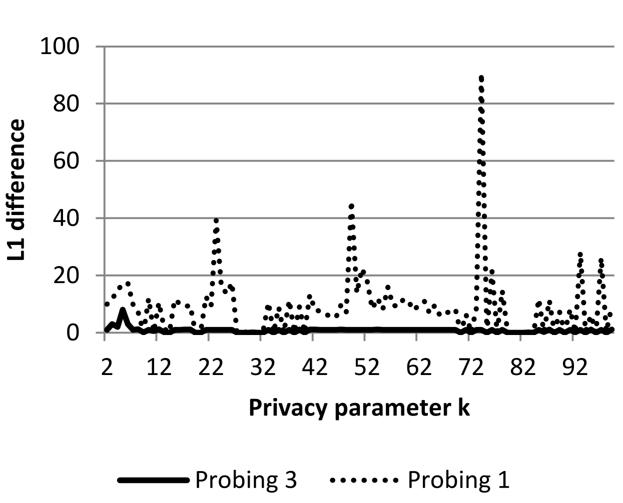


(a) Powergrid graph

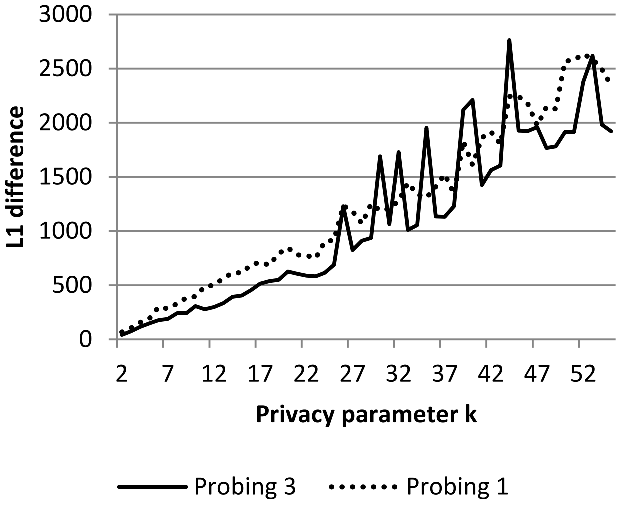


(b) Caltech graph

Figure 9.10: Comparison between average L1-distance of the degree sequence returned by `OptimalDegreeAnon` and that of the graph returned algorithm 9.4 using *probing scheme 1 and 2* over 20 runs.

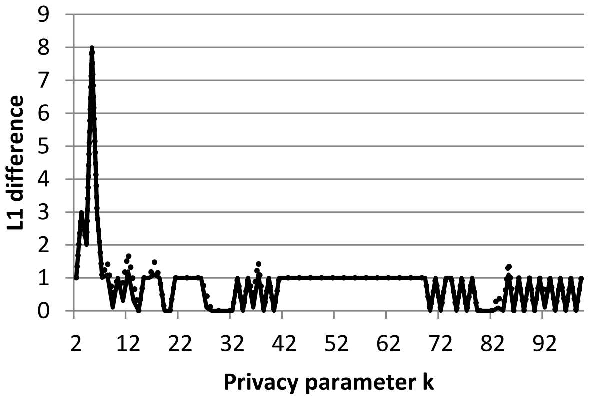


(a) Powergrid graph

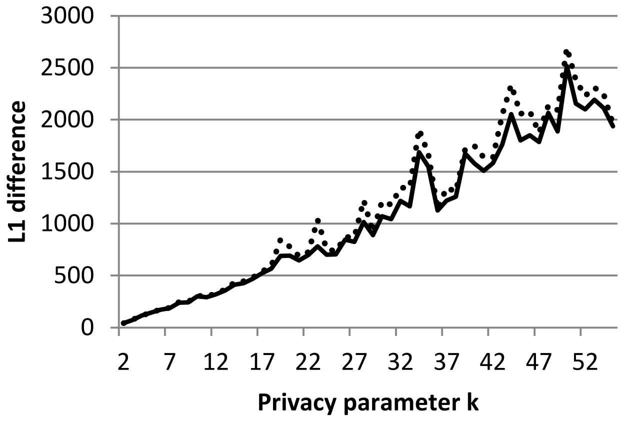


(b) Caltech graph

Figure 9.11: Comparison between average L1-distance of the degree sequence returned by `OptimalDegreeAnon` and that of the graph returned algorithm 9.4 using *probing scheme 1 and 3* over 20 runs.



(a) Powergrid graph



(b) Caltech graph

Figure 9.12: The full line is the average L1-distance between the degree sequence returned by `OptimalDegreeAnon` and that of the graph returned algorithm 9.4 using *probing scheme 4* over 20 runs. The dotted line is obtained by adding the variance to the full line.

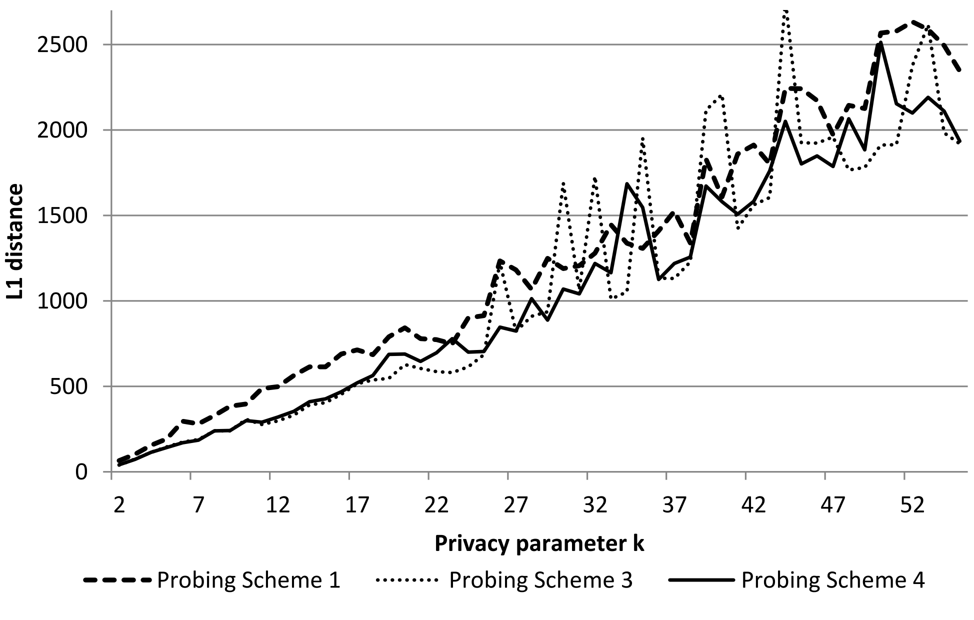


Figure 9.13: Comparison between average L1-distance of the degree sequence returned by `OptimalDegreeAnon` and that of the graph returned algorithm 9.4 using *probing scheme 1, 3 and 4* over 20 runs for the Caltech graph.

found. In the following algorithms we will always use probing scheme 4.

9.7.3 GreedySwap and SimultaneousSwap

Both `GreedySwap` and `SimultaneousSwap` use `ConstructGraph` to create the initial anonymous graph. Both algorithms will then swap edges in the created graph to make it resemble the input graph. The difference is that `GreedySwap` only uses edge additions when computing the k-anonymous degree sequence while `SimultaneousSwap` allows both edge deletions and additions during this step. When the degree sequence isn't realizable a probing scheme is used.

The probing scheme has already been extensively studied for the algorithm `SuperGraph`, and we will use probing scheme 4 for all algorithms. Furthermore the variance of the results are similar to that of `SuperGraph` for the input graphs we used. No particularly interesting observations were made when *individually* implementing and testing these algorithms. Our main interest is in comparing the results of the algorithms, which will be done in the next section.

9.8 Comparison Between Algorithms

Finally we can compare the performance of the anonymization algorithms against each other. Abbreviated algorithm names have been used to avoid the graphs from becoming too cluttered. We will test the following three:

- `SuperGraph`: Algorithm 9.4 on page 143
- `Relaxed`: Algorithm 9.7 on page 146
- `SimulSwap`: Algorithm 9.8 on page 148

The Enron, Powergrid, Caltech and Bowdoin graphs will be used to test the algorithms. Note that because the Enron graph has only 151 nodes, results deteriorate quickly for it in function of k, and once k is higher than 75 all nodes of the Enron graph are put in the same anonymization group. The other three graphs are large enough to avoid this problem. Again we will run the algorithms a number of times and then take the average of the computed properties. However, because of the size of the graphs, only a limited amount of runs for each value of k could be executed. To increase the speed not *all* values of k were tested and its value was increased with a predefined interval. An interval of, say 3, would mean the algorithm was tested for the values $k = 2, 5, 8, 11, \ldots, 98$. These parameters are summarized in table 9.2.

Unfortunately there are no special metrics designed to measure "how much two graphs differ in terms of structure" [Hay10]. Hence the effect of anonymization on the graphs will be measured by studying the change in several graph properties.

Graph Name	#Nodes	#Edges	#Runs	Interval Size
Enron	151	502	20	2
Powergrid	4,939	6,594	5	3
Caltech	768	16,656	4	3
Bowdoin	2,251	84,387	2	5

Table 9.2: Graph Statistics and Testing Parameters.

Diameter

The effect on the diameter of the graphs is shown in figure 9.14 on page 163. Although not shown, the variance of this property can be rather high. For the Enron and Caltech graph the variance is generally around 1, but for the Powergrid and Bowdoin graph using the `SimulSwap` algorithm it can be as large as 4. When using the `Supergraph` or `Relaxed` algorithms the variance is typically lower. That the variance of this property is large can be explained because it's the *maximum* distance between any pair of nodes. Adding or removing a specific edge can drastically influence this property.

Our tests indicate that `SimulSwap` best preserves the diameter of a graph. Remember that the results for the Enron graph deteriorate quickly because it only has few nodes, and therefore one should not focus too much on the results obtained from this graph.

Average Path Length (APL)

The effect on the average path length of the graphs is shown in figure 9.15 on page 164. The variance of this property is low. Interestingly `SuperGraph` seems to have better results for low values of k on the Powergrid graph, but `SimulSwap` is still better for higher values of k. For the other three graphs `SimulSwap` is clearly the better algorithm when considering the average path length.

Density

Figure 9.16 on page 165 shows the effect on the density of the graphs. Since only `SimulSwap` allows both edge additions and deletions using when constructing the k-anonymous degree sequence it clearly performs better. However we also see that `Relaxed` is slightly better than `SuperGraph`.

Powerlaw Estimation of the Degree Sequence

The effect on the powerlaw estimation of the degree sequence (as explained in section 7.3.2 on page 106) is shown in figure 9.17 on page 166. For the Powergrid graph the results of `Relaxed` and `SuperGraph` coincide, and for the Bowdoin graph the results of `SimulSwap` and `Relaxed` coincide.

Because the Enron graph contains only 151 nodes, estimating the powerlaw distribution becomes meaningless for higher values of k, since there will be only few anonymization groups left.

Clustering Coefficient

In figure 9.18 on page 167 the change in the clustering coefficient is show after anonymization. For the Powergrid graph `SuperGraph` clearly outperforms the other two algorithms. However, for the Caltech and Bowdoin graph it's less ideal and `SimulSwap` might be better for very large values of k.

Summary

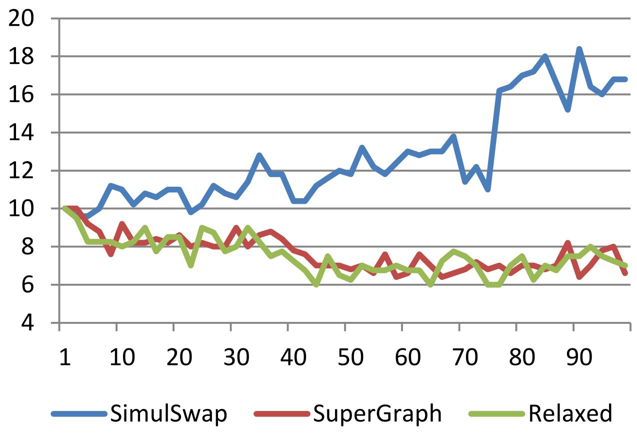
There is no superior algorithm. Depending on both the graph and the property being measured, some algorithms perform better than others. Thus selecting an anonymization algorithm can be a hard task, although commonly `SimulSwap` is chosen because it allows both edge additions and removals during all steps of the algorithm [Hay10].

9.9 Relation to Link Prediction

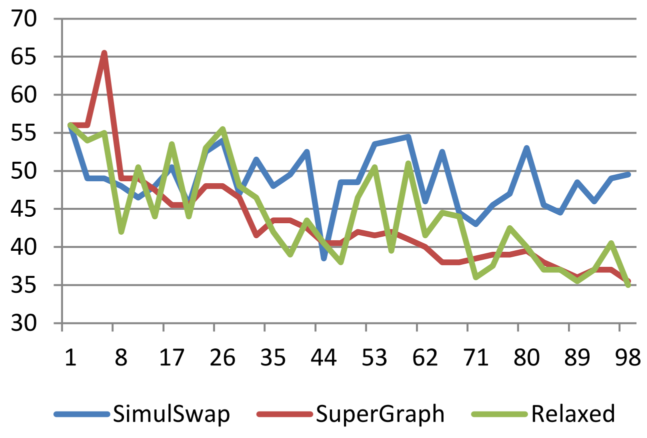
All the previous algorithms add or remove random edges. A possible problem is that the edges being added might actually be unlikely to exists in reality. Or that the removed edges are easy to predict. Therefore it might be possible to detect which edges where added during the anonymization on the graph, and that removed edges can be predicted. This boils down to the *link prediction problem*, where the question is asked which new interactions among its members are likely to occur in the near future [LNK07]. It might be possible to remove part of the noise introduced by the degree anonymization mechanism, and thus still breach the privacy of certain individuals. Unfortunately we did not have the time to explore this idea, but it could be an interesting direction for future work.

9.10 Conclusion

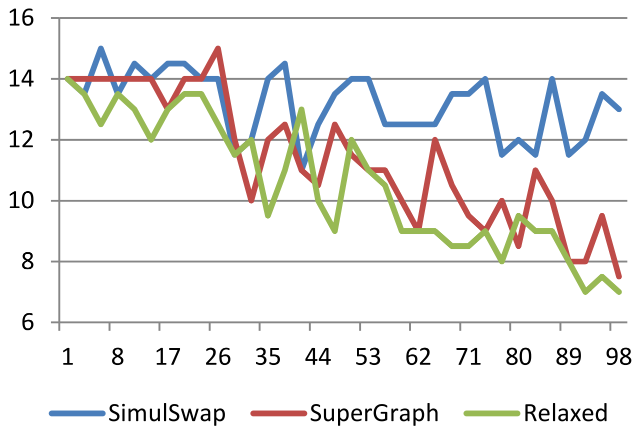
There are two obstacles that prevent degree anonymization from being useful in practice. First of all, assuming that the adversary only knows the degrees of targeted individuals is a strong assumption, an adversary likely also knows about the existence of certain edges. The second problem is that the current algorithms do not scale well to large graphs due to—among other things—the probing scheme.



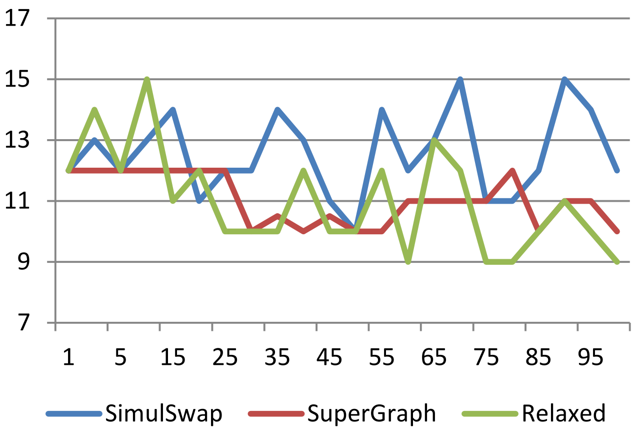
(a) Enron graph



(b) Powergrid graph

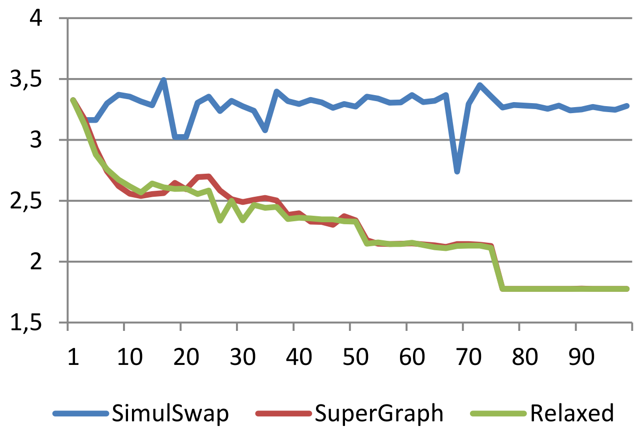


(c) Caltech graph

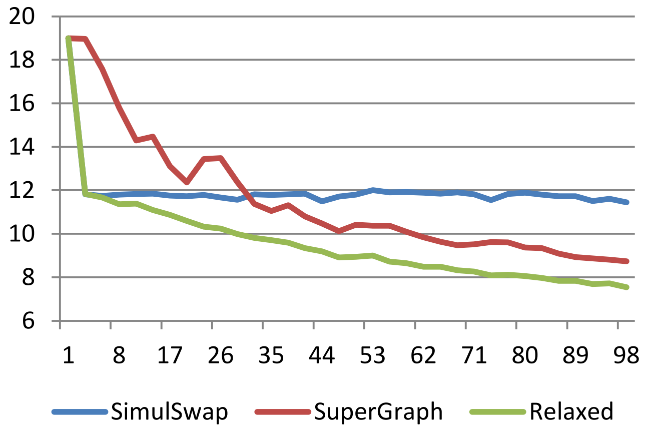


(d) Bowdoin graph

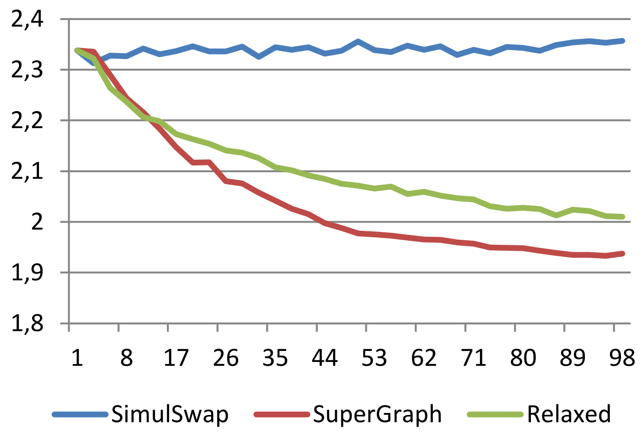
Figure 9.14: Diameter. The x-axis shows the privacy parameter k and the y-axis the diameter of the k-anonymous graph returned by the the various anonymization algorithms.



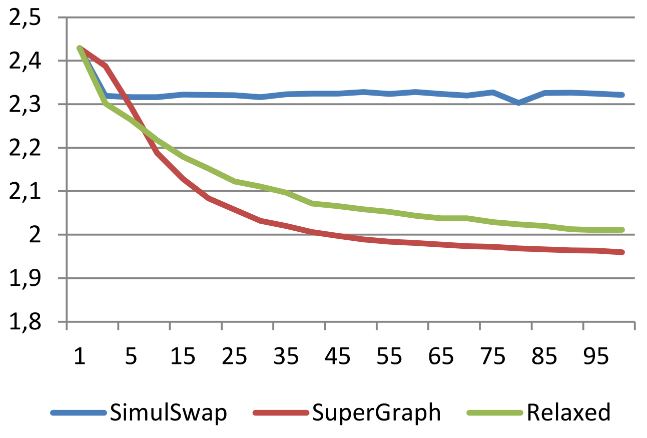
(a) Enron graph



(b) Powergrid graph

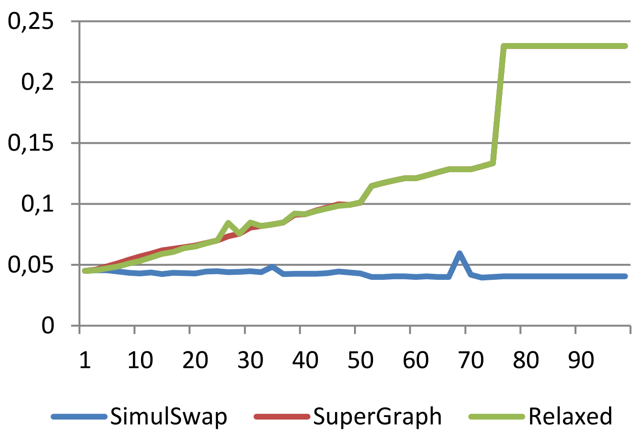


(c) Caltech graph

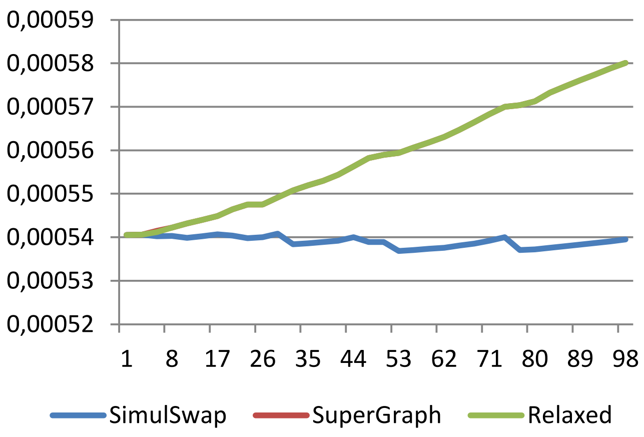


(d) Bowdoin graph

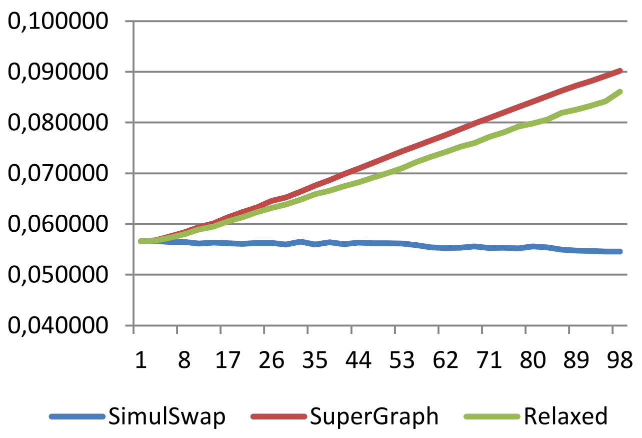
Figure 9.15: Average Path Length (APL). The x-axis shows the privacy parameter k and the y-axis the APL of the k-anonymous graph returned by the the various anonymization algorithms.



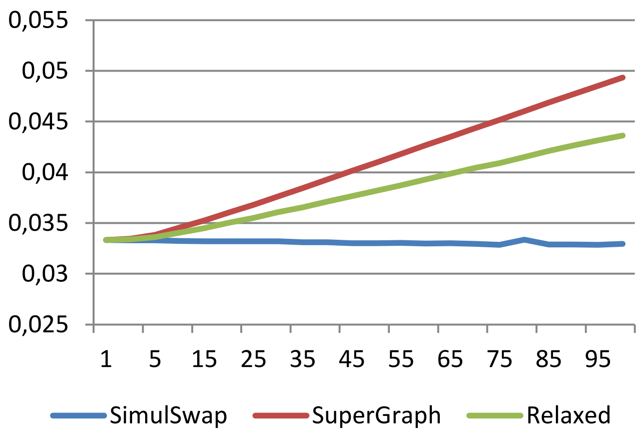
(a) Enron graph



(b) Powergrid graph

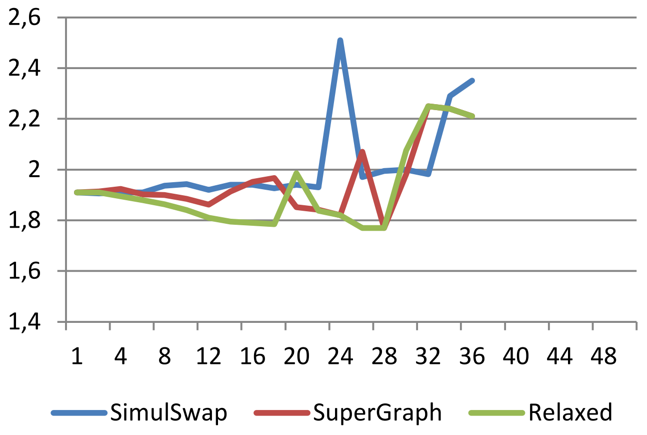


(c) Caltech graph

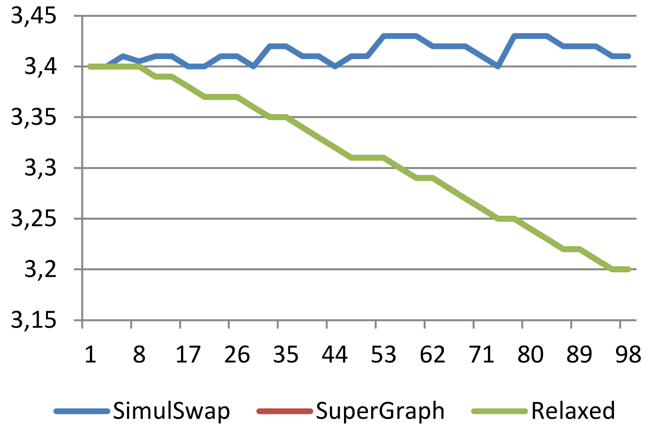


(d) Bowdoin graph

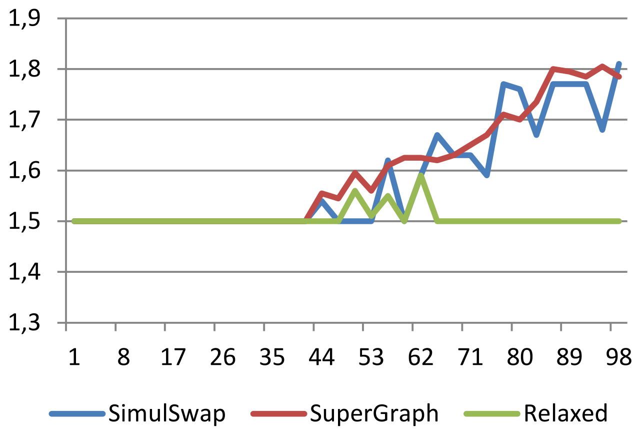
Figure 9.16: Density. The x-axis shows the privacy parameter k and the y-axis the density of the k-anonymous graph returned by the the various anonymization algorithms.



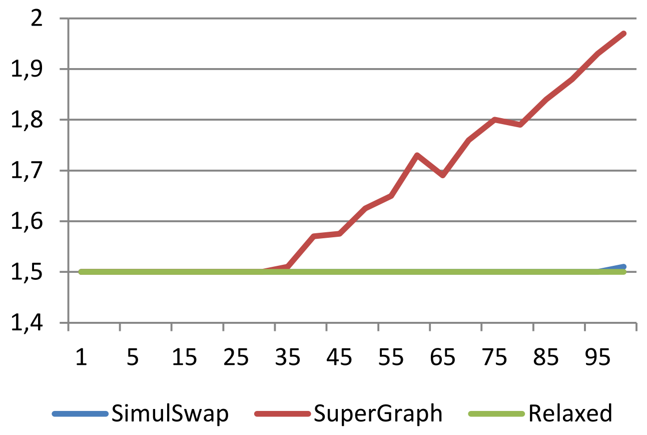
(a) Enron graph



(b) Powergrid graph

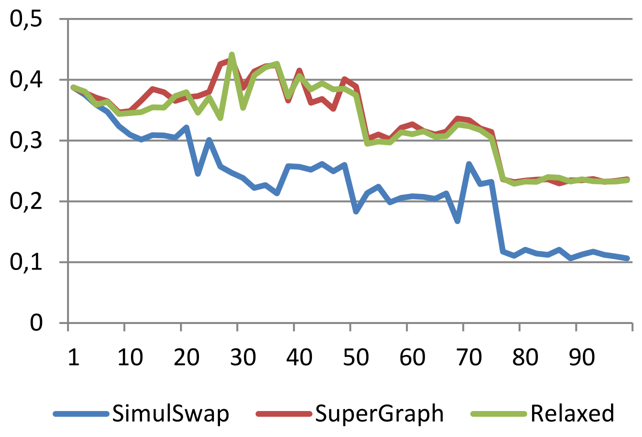


(c) Caltech graph

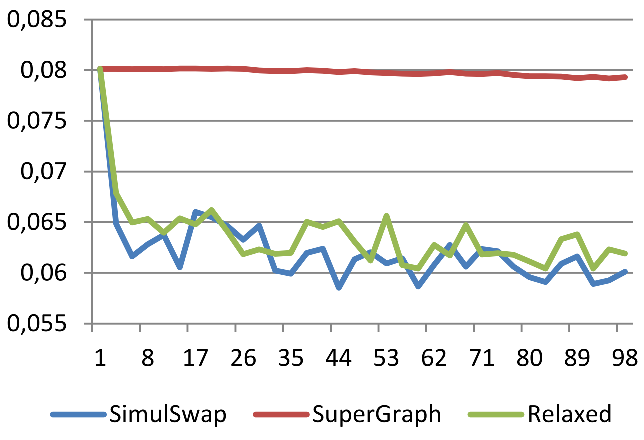


(d) Bowdoin graph

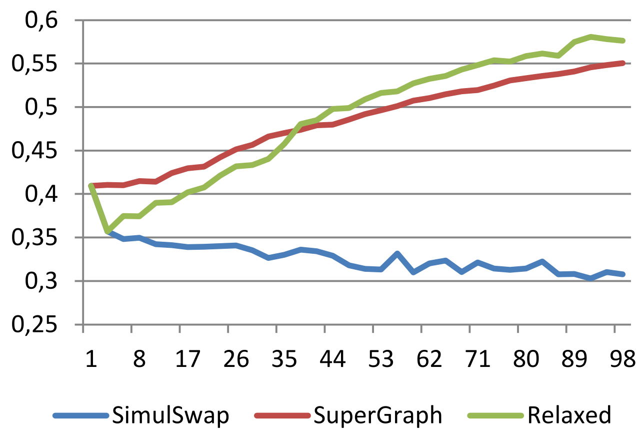
Figure 9.17: Powerlaw. The x-axis shows the privacy parameter k and the y-axis the powerlaw estimation of the degree sequence of the k-anonymous graph returned by the the various anonymization algorithms.



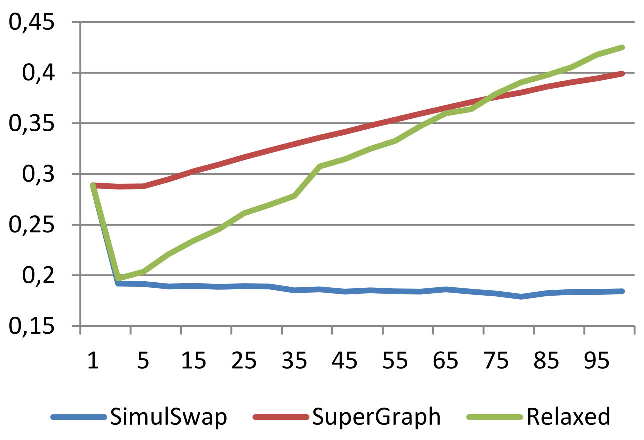
(a) Enron graph



(b) Powergrid graph



(c) Caltech graph



(d) Bowdoin graph

Figure 9.18: Clustering Coefficient. The x-axis shows the privacy parameter k and the y-axis the clustering coefficient of the k-anonymous graph returned by the the various anonymization algorithms.

Chapter 10

Passive Attack

The degree anonymization mechanism explained in chapter 9 protects only against limited adversaries, and is still vulnerable to attack. Although not causally related to degree anonymization, the passive attack will be presented as an example of a powerful attack an adversary can perform which can possibly defeat degree anonymization. In contrary to the active attack, the passive attack can be performed without adding new nodes in the target network. The chapter is based on two papers by Narayanan et al. [NSR11, NS09].

10.1 Background

When applying the degree anonymization mechanism explain in the previous chapter, we assumed the adversary only knows the degree of the user(s) he or she wants to target. We already mentioned that this is not always a realistic assumption, and in certain situations the adversary can posses more detailed information. As a result degree anonymization is not a very powerful anonymization technique that is usable in practice. However, it's an excellent example that illustrates how one can attempt to prevent privacy breaches, but if an adversary posses more auxiliary information than assumed, he or she will still able to re-identify individuals.

Originally the passive attack outlined in this chapter was not a reaction to demonstrate the weaknesses of degree anonymization. However, it was the first algorithm that showed the feasibility of large-scale, passive de-anonymization of real-world social networks [NS09]. Although time restrictions prevent us from testing the passive attack on a degree anonymized network, we believe a significant amount of individuals can be re-identified using a passive attack. And even if not the case, the passive attack is an important result on its own.

The target of the passive attack is an anonymized social network which is possibly modified by introducing noise [NS09]. The auxiliary information of the adversary consists of a second social network that has *some* overlap with the network to be attacked. Note that this overlap doesn't need to be large, even an overlap of only 15% suffices. With this auxilairy information the adversary is able to re-identify a significant amount of individuals using only the structure of both networks.

10.2 The Setting

The passive attack attempts to re-identity nodes in an anonymized graph. Anonymization is modeled by publishing only a subset of attributes, and that the released attributes themselves are insufficient for re-identification. Furthermore the data release process may involve perturbation or sanitization that changes the graph structure in some way to make re-identification attacks harder [NS09]. An example of sanitization can be degree anonymization. We will call the sanitized target network S_{SAN}. Because of how the graphs are obtained and published some nodes might have no outgoing edges, and we will call such nodes *crawled nodes*.

In addition to S_{SAN} the adversary also has access to a different network S_{AUX} whose membership partially overlaps with S_{SAN}. This auxiliary network can be obtained in several ways. It can be creating by crawling a similar, but public, social network. Or a government-level agency can use modern surveillance techniques to create their own social graph. Another possibility is that a hacker can obtained an auxiliary graph from a data breach. When carrying out the attack it makes no difference how S_{AUX} was obtained. The only requirement is that there is a partial membership overlap between S_{SAN} and S_{AUX}.

The actual de-anonymization of S_{SAN} is done in two steps. The first step, called *seed identification*, is the biggest challenge and consists of de-anonymizing an initial number of nodes. In the second step, called *propagation*, the de-anonymization is propagated by selecting an arbitrary, still anonymous node, and finding the most similar node in the auxiliary graph. Both steps will be discussed in detail.

10.3 Seed Identification

Seed identification consists of creating an mapping between an initial subset of nodes in S_{SAN} to nodes in S_{AUX}. Note that a mapping effectively de-anonymizes the node in question, as we now know its identity from the auxiliary information. Seed identification can be achieved using two methods. The first one uses individual auxiliary information [NS09], and the second one is based on the weighted graph matching problem [NSR11].

10.3.1 Individual Auxiliary Information

To de-anonymize an initial number of nodes an attackers *individual auxiliary information* can be used. It consists of detailed information about a very small number of members of the target network S_{SAN}. With this information the attacker can determine the location of these members in the anonymized graph, or if they are not present in the graph. An example used in the original paper of Narayanan and Shmatikov is a clique of k nodes which are present in both S_{SAN} and S_{AUX}. The attack must also know the degree of each node and the number of common neighbors each pair of nodes in the clique has.

Based on this information the seed identification step searches the target graph for the unique clique with matching node degrees and common neighbors counts. This closely resembles the active attack discussed in chapter 8, and this method is essentially a pattern search. Remark that only few seeds are needed, so only a limited amount of individual auxiliary information is needed. An attacker can collect such information by bribing individuals that are included in both networks, some individuals share all their information publicly which the attacker can crawl, or an attacker can attempt to hack a selected number of social profiles in order to retrieve enough information.

Originally this method was used to find seeds between two different social networks. Specifically, in the paper by Narayanan and Shmatikov, it was used to find seeds for crawls of Twitter and Flickr. And these are *different* social networks.

10.3.2 Combinatorial Optimization Problem

In this method the structure of both graphs is used to find initial seeds. Before we explain it further, it's important to know is that this attack was designed to find seed mappings between two networks that originated from the *same* social network. The target network S_{SAN} was created by Kaggle by crawling Flickr during 21-28 June 2010. Between mid-December 2010 and mid-January 2011 the researcher created the auxiliary network S_{AUX} by also crawling Flickr. The crawls were made by initially downloading random nodes, and subsequently sampling uniformly from the outbound neighbors of these nodes. Not all nodes had outgoing edges in the target graph, such edges may exist in reality but weren't included in the crawls (these are positions were the crawling process was terminated). We will call nodes without outgoing edges *crawled nodes.*

Since the graphs can contain millions of nodes, it's essential to first reduce the search space. The assumption that will be used is that, with high probability, nodes having a high in-degree in both graphs roughly correspond to each other. In other words nodes with the highest in-degree will be present in both graphs. Note that for the crawling process used to obtain S_{SAN} and

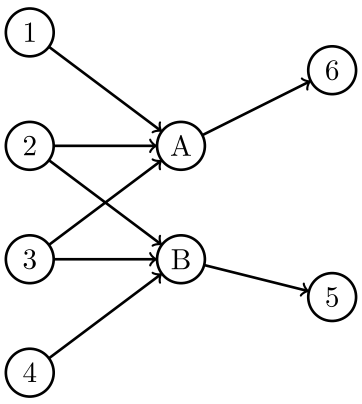


Figure 10.1: Example to who high in-degree nodes A and B.

S_{AUX} this is highly likely.

Now, in order to reduce the search space, we can pick two small subsets of nodes, where all nodes have a high in-degree, knowing that with high probability there will be a large correspondence between these subsets. The next step is to find the actual correspondence between these high in-degree nodes. Because we are not sure if all edges among the top k nodes are present in the target graph, it's not possible to search for the mapping that minimizes the edges mismatches between them.

Instead, for each graph separately, the cosine similarity of the sets of in-neighbors is calculated for every pair of nodes. The set of in-neighbors of a node are all the nodes with a directed edge from themselves to the node under consideration. Here the cosine similarity between two set of nodes X and Y is defined as

$$sim(X,Y) = \frac{|X \cap Y|}{\sqrt{|X||Y|}}$$

For example, the cosine similarity of the nodes A and B in figure 10.1 is

$$\frac{|\{2,3\}|}{\sqrt{3 \cdot 3}} = \frac{2}{3}$$

The idea is now that the cosine similarity between two nodes is roughly the same as the cosine similarity between the corresponding nodes in the other graph. Thus we must find a mapping that minimizes the differences between the cosine similarity of all pairs of nodes. This can be reduced to an existing problem by regarding the cosine similarity as the weight of an edge between these two nodes. Doing this for both graphs creates two complete weighted graphs. Now the goal is to find a mapping such that the weights of corresponding edges are as close to each other as possible, which is called the *weighted graph matching* problem. It can be solved using existing

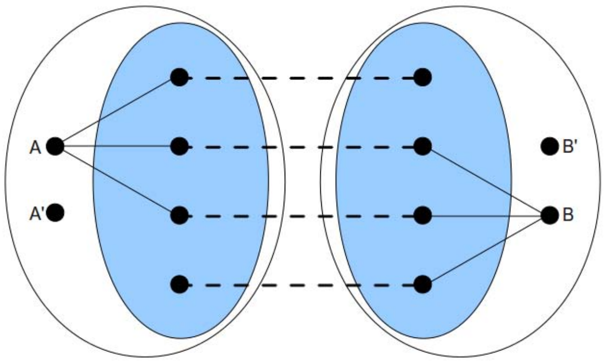


Figure 10.2: Blue nodes have already been mapped. The similarity between A and B is $\frac{2}{3}$. Possible edges between A and A' or B and B' do not influence the similarity. [NSR11, Nar11]

algorithms or heuristics [NSR11]. We end up with a mapping between the high in-degree nodes, completing the seed identification step.

10.4 Propagation

In order to propagate the de-anonymization, the algorithm stores a partial mapping between the nodes and iteratively extends this mapping. How the mapping is extended depends on how the input graphs were obtained. Sometimes different strategies are needed in different situations, as is evidenced by the different algorithms used in the two papers by Narayanan. In this section we can combine both methods to create a general algorithm, but note that in practice our presented algorithm may require further tweaking.

Similarity Measure

First we need to measure the similarity between two unmapped nodes in both graphs. For this we calculate the cosine similarity between the already mapped neighbors of both nodes. Nodes mapped to each other will be treated as identical for the purpose of cosine comparison. Figure 10.2 shows an example. Note that only edges to already mapped nodes influence the cosine similarity. However, this calculation is not trivial, as we also have to take into account edge directionality and the fact that not all nodes are crawled (i.e., some nodes have no outgoing edges). This is solved by ignoring the outgoing edges of the nodes unless *both* have been crawled. The final similarity measure is shown in algorithm 10.1.

Algorithm 10.1 Similarity scores for two nodes.

Input:

v_{san}:	a node in the target graph
v_{aux}:	a node in the auxiliary graph
D:	set of nodes in v_{san} de-anonymized so far
map:	current partial mapping between nodes in S_{SAN} and S_{AUX}
C_{SAN}:	set of crawled nodes in S_{SAN}
C_{AUX}:	set of crawled nodes in S_{AUX}
$N^+(\cdot)$:	out-neighborhood of a node
$N^-(\cdot)$:	in-neighborhood of a node

Output: Similarity score between v_{san} and v_{aux}.

```
1: $N_{\text{SAN}} := \text{map}[N^-(v_{\text{san}}) \cap D] \cap C_{\text{AUX}}$
2: $N_{\text{AUX}} := N^-(v_{\text{aux}}) \cap \text{map}[C_{\text{SAN}}]$
3: if $v_{\text{san}} \in C_{\text{SAN}}$ and $v_{\text{aux}} \in C_{\text{aux}}$ then
4:     $N_{\text{SAN}} := N_{\text{SAN}} \cup \text{map}[N^+(v_{\text{san}}) \cap D] \cap C_{\text{AUX}}$
5:     $N_{\text{AUX}} := N_{\text{AUX}} \cup N^+(v_{\text{aux}}) \cap \text{map}[C_{\text{SAN}}]$
6: end if
7: return $\text{CosineSim}(N_{\text{SAN}}, N_{\text{AUX}})$
```

Further Details

To increasing the probability that a mapping is correct the eccentricity of a mapping is used. It measures how much an item stands out from the rest, and is defined as

$$\frac{\max(X) - \max_2(X)}{\sigma(X)}$$

where max and $\max_2$ denote the highest and second highest values, respectively, and where σ is the standard deviation [NS09]. It's now possible to reject a mapping if the eccentricity is below a certain threshold.

During the executing of the algorithm it will revisit already mapped nodes. The newly calculated mapping for a node may be different than the current mapping, and if so it will be updated accordingly. This is done because at the early stages of the algorithm only few mapped nodes are available, thus making an incorrect mapping more likely. As more and more mappings become available this error can be detected and corrected. Practically this is done by trying to find a mapping for every node during each iteration, even for already mapped ones.

When a possible mapping from a node in S_{SAN} to S_{AUX} is found, the graphs are switched. Only if then the same mapping is found on the switched graphs will it be added. This increases the probability that the mapping is correct. By doing this is also makes no difference which graph is the target network, and which is the auxiliary network. Switching them will still result

```
def matchScores(lgraph, rgraph, mapping, lnode):
    return [ similarity (lnode, rnode) for rnode in rgraph.nodes]

def propagationStep(lgraph, rgraph, mapping):
    for lnode in lgraph.nodes:
        scores [lnode] = matchScores(lgraph, rgraph, mapping, lnode)
        if eccentricity (scores [lnode]) < theta: continue
        rnode = (pick node from rgraph.nodes where
                    scores [lnode][node] = max(scores[lnode]))

        scores [rnode] = matchScores(rgraph, lgraph, invert(mapping), rnode)
        if eccentricity (scores [rnode]) < theta: continue
        reverse_match = (pick node from lgraph.nodes where
                    scores [rnode][node] = max(scores[rnode]))

        if reverse_match != lnode:
            continue

        mapping[lnode] = rnode

def passiveAttack():
    while not converged:
        propagationStep(lgraph, rgraph, seedMapping)
```

Listing 10.1: Passive Attack

in the same output. For this reason we will also use the more general terms `lgraph` and `rgraph`, denoting the left graph and right graph, respectively.

Propagation Step

The passive attack is can now be implemented. Each iteration it calculates the similarity score of all pair of nodes, and if the eccentricity is high enough the mapping is added. Theta will be a parameter denoting the threshold on the eccentricity for a mapping to be added. It will continue to iterate over all node pairs until not more changes are made to the mapping. The final algorithm is shown in listing 10.1.

10.5 Experiments

10.5.1 Anonymized Flickr Graph

As already mentioned in section 3.1.4 a passive attack was performed on a supposedly anonymized social graph used in the Kaggle link prediction challenge. For the challenge they crawled a subset of the Flickr social network. Fake edges where added to the resulting graph, and contestants had to use link prediction to determine whether selected edges were fake or genuine.

In an attempt to preserve privacy—and to prevent cheating—the released graph was naively anonymized.

Narayanan, Shi and Rubinstein where able to re-identify 64.7% of the nodes with an accuracy of 95.2% [NSR11]. First they made their own crawl of Flickr to obtain the auxiliary graph used in the passive attack. Because both graphs stem from the same network the seed identification was done using the *combinatorial optimization problem*. During the contest they found a seed mapping of 10 nodes among the top 20 nodes by mere visual inspection of the matrix of cosines [NSR11]. Later on they developed an automated approach to the weighted graph matching problem.

10.5.2 Flickr and Twitter

The passive attack was also tested on two different social networks, namely Twitter and Flickr. The graphs were obtained by crawling the *public* profiles on the social network sites, so again a selection bias may be present due to the crawling process. First a ground truth was established, i.e., the true mapping between the two graphs. These mappings were based on exact matches of username and name of a profile. Human inspection of a random sample of the resulting ground truth indicated that the error rate is well under 5%.

Seed identification was done by simply selecting 150 pairs of nodes from the ground truth. This mapping was then fed to the passive attack algorithm. When completed 30.8% of the mappings were correctly re-identified, 12.1% where incorrectly re-identified and 57% of the ground truth mappings were not identified. Interestingly 41% of the incorrectly identified mappings (6.7% overall) where mapped to nodes that were neighbors of the true mapping. It's suggested that human verification can complete the re-identification process in those situations.

10.6 Conclusion

The passive attack is a potentially powerful attack, given that the adversary has access to enough auxiliary information. And the experiments illustrate that one must *not* underestimate the amount of auxiliary information an adversary can posses.

Chapter 11

Conclusion

We have argued that privacy is increasingly important now that more and more data is being collected. Previous events have taught us that intuitive methods to assure privacy are likely to fail. This is manifested by examples such as the AOL search fiasco, the HMO record linkage attack, the Netflix prize and the Kaggle machine learning contest. In all of these cases superficial attempts have been made to assure privacy, which caused all of them to fail. This motivated the search to a mathematical definition of privacy.

The holy grail of privacy definitions would be Dalenius desideratum, which states that *obtaining access to the database doesn't increase the chance of a privacy breach*. Unfortunately such a guarantee is impossible, as there is always a piece of auxiliary information that, combined with access to the database, will violate the privacy of an individual. After all, the goal of a database is to store data and learn new information from this data. It's unavoidable that this newly learned information can increase the chance of a possible privacy breach. From this the idea of relative disclosure prevention was suggested.

Relative disclosure prevention states that there is only a small increase of the possibility that a privacy breach occurs. This can give rise to a definition that *is* achievable in practice. One of the most promising definitions in this area is differential privacy. Roughly speaking it states that handing over your data—or keeping it private—doesn't significantly affect the chance of a privacy breach occurring. Written differently, hiding your data doesn't offer you much benefits when differential privacy is being used.

Bibliography

[AE11] Julian Assange and Laura Emmett. Interview with julian assange by RT. Retrieved 4 May, 2011, from `http://rt.com/news/wikileaks-revelations-assange-interview/`, May 2011.

[AFK+05] Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Approximation algorithms for k-anonymity. In *Proceedings of the International Conference on Database Theory (ICDT 2005)*, November 2005.

[And06] Nate Anderson. The ethics of using AOL search data. *Ars Technica*, August 2006.

[Arr06] Michael Arrington. AOL proudly releases massive amounts of private data. *Tech Crunch*, August 2006.

[AW89] Nabil R. Adam and John C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys (CSUR)*, 21:515–556, December 1989.

[BDK07] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. 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, pages 181–190. ACM, 2007.

[BDMN05] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the SuLQ framework. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems*, PODS 2005, pages 128–138. ACM, 2005.

[BL07] James Bennett and Stan Lanning. The Netflix prize. In *Proceedings of KDD Cup and Workshop*, August 2007.

[BMS04] Peter S Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, 110(1):44–91, 2004.

[Bol01] B. Bollobas. *Random Graphs.* Cambridge University Press, 2001.

[BZ06] Michael Barbaro and Tom Zeller. A face is exposed for AOL searcher no. 4417749. *The New York Times*, August 2006.

[Cho86] S.A. Choudum. A simple proof of the erdos-gallai theorem on graph sequences. *The BULLETIN of the Australian Mathematical Society*, 33:67–70, 1986.

[CSN09] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51:661–703, November 2009.

[Dal77] Tore Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15, 1977.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Theory of Cryptography Conference*, pages 265–284, February 2006.

[DN10] Cynthia Dwork and Moni Naor. On the difficulties of disclosure prevention in statistical databases or the case for differential privacy. *Journal of Privacy and Confidentiality*, 2(1):93–107, 2010.

[DNP$^+$10] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *Proceedings of The First Symposium on Innovations in Computer Science (ICS 2010).* Tsinghua University Press, January 2010.

[DNPR10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 715–724. ACM, 2010.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Lenoid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, March 2008.

[DRV10] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 51–60. IEEE Computer Society, 2010.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting erros without leaking partial information. In *Proceedings of the 37th ACM Symposiumm on Theory of Computing (STOC 2005)*, pages 654–663. ACM Press, 2005.

[Dwo06] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, July 2006.

[Dwo08] Cynthia Dwork. Differential privacy: a survey of results. In *Proceedings of the 5th international conference on Theory and applications of models of computation*, TAMC'08, pages 1–19. Springer, 2008.

[Dwo11] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54:86–95, January 2011.

[Fac] Facebook. Facebook's data use policy: Personalised adverts. Retrieved, 6 December 2011, from `https://www.facebook.com/about/privacy/advertising`.

[FCS$^+$06] Dan Frankowski, Dan Cosley, Shilad Sen, Loren Terveen, and John Riedl. You are what you say: privacy risks of public mentions. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 565–572. ACM Press, 2006.

[FG96] Bert Fristedt and Lawrence Gray. *A Modern Approach to Probability Theory.* Birkhauser Boston, 1996.

[Gav80] Ruth Gavison. Privacy and the limits of the law. *Yale Law Journal*, 1980.

[GKS08] Srivatsava Ranjit Ganta, Shiva Kasiviswanathan, and Adam Smith. Composition attacks and auxiliary information in data privacy. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 265–273. ACM Press, 2008.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1 of *Foundations of Cryptography.* Cambridge University Press, 2001.

[Gol06] Philippe Golle. Revisiting the uniqueness of simple demographics in the US population. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, WPES '06, pages 77–80. ACM Press, 2006.

[GW86] Geoffrey Grimmett and Dominic Welsch. *Probability: An Introduction.* Oxford University Press, 1986.

[Hay06] Brian Hayes. Can the tools of graph theory and social-network studies unravel the next big plot? *American Scientist*, 95, October 2006.

[Hay10] Michael Hay. *Enabling Accurate Analysis of Private Network Data.* PhD thesis, University of Massachusetts, 2010.

[HLMJ09] Michael Hay, Chao Li, Gerome Miklau, and David Jensen. Accurate estimation of the degree distribution of private networks. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining.* IEEE Computer Society, 2009.

[HMJ$^+$08] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. *Proc. VLDB Endow.*, August 2008.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[ILL89] Russell Impagliazzo, Leonid Anatolievich Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *Proceedings of the 21st annual ACM Symposium on Theory of Computing (STOC '89)*, pages 12–24. ACM Press, 1989.

[Kag10] Kaggle. IJCNN social network challenge. Retrieved 14 May, 2011, from `http://www.kaggle.com/c/socialNetwork`, 2010.

[Kai08] Marcus Kaiser. Mean clustering coefficients: the role of isolated nodes and leafs on clustering measures for small-world networks. *New Journal of Physics*, 10(8), 2008.

[KMN05] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. Simulatable auditing. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems*, PODS 2005, pages 118–127. ACM Press, 2005.

[KPR00] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. Auditing boolean attributes. In *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, PODS 2000, pages 86–91. ACM Press, 2000.

[Kum01] I. Ravi Kumar. *Comprehensive Statistical Theory of Communication.* Laxmi Publications Ltd, 2001.

[Lee95] Yoong-Sin Lee. Graphical demonstration of an optimality property of the median. *The American Statistician*, 49(4):369–372, 1995.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *23rd International Conference on Data Engineering*, pages 106–115, April 2007.

[LNK07] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58:1019–1031, May 2007.

[LT08] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 93–106. ACM, 2008.

[Mar10] Moxie Marlinspike. BlackHat Europe 2011: Changing threats to privacy. Retrieved 17 April, 2011, from `http://www.blackhat.com/html/bh-eu-10/bh-eu-10-archives.html`, April 2010.

[MB06] Elinor Mills and Anne Broache. Three workers depart AOL after privacy uproar. *CNET News*, August 2006.

[MFJ$^+$07] Matthew D. Mailman, Michael Feolo, Yumi Jin, Masato Kimura, Kimberly Tryka, et al. The NCBI dbGaP database of genotypes and phenotypes. *Nature Genetics*, 39:1181–1186, October 2007.

[MGK07] Ashwin Machanavajjhala, Johannes Gehrke, and Daniel Kifer. ℓ-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1, March 2007.

[MT07] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society.

[MV02] Milena Mihail and Nisheeth Vishnoi. On generating graphs with prescribed degree sequences for complex network modeling applications. Position Paper, ARACNE, 2002.

[MW04] Adam Meyerson and Ryan Williams. 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, pages 223–228. ACM, 2004.

[Nao10a] Moni Naor. Foundations of privacy - spring 2010: Lecture 1. Retrieved 21 April, 2011, from `http://www.wisdom.weizmann.ac.il/~naor/COURSE/foundations_of_privacy.html`, 2010.

[Nao10b] Moni Naor. Foundations of privacy - spring 2010: Lecture 2. Retrieved 21 April, 2011, from `http://www.wisdom.weizmann.ac.il/~naor/COURSE/foundations_of_privacy.html`, 2010.

[Nar11] Arvind Narayanan. 33 bits of entropy: Link prediction by de-anonymization: How we won the kaggle social network challenge. Retrieved, 7 Januari 2012, from `http://bit.ly/gkWcIZ`, March 2011.

[Net] Netflix. Netflix prize: Frequently asked questions. Retrieved May 14, 2011, from `http://www.netflixprize.com/faq`.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 111–125. IEEE Computer Society, 2008.

[NS09] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, pages 173–187. IEEE Computer Society, 2009.

[NSR11] Arvind Narayanan, Elaine Shi, and Benjamin Rubinstein. Link prediction by de-anonymization: How we won the Kaggle social network challenge. Retrieved 20 April, 2011, from `http://arxiv.org/abs/1102.4374`, February 2011.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[pan11] Panorama: we zijn gezien. Retrieved 12 May, 2011, from http://www.deredactie.be/cm/vrtnieuws/mediatheek/programmas/panorama/1.1014456, May 2011. Vlaamse Radio- en Televisieomroeporganisatie.

[Pos01] Robert C. Post. Three concepts of privacy. *Faculty Scholarship Series*, 2001.

[PPPM$^+$02] J J Potterat, L Phillips-Plummer, S Q Muth, R B Rothenberg, D E Woodhouse, T S Maldonado-Long, H P Zimmerman, and J B Muth. Risk network structure in the early epidemic phase of hiv transmission in colorado springs. *Sexually Transmitted Infections*, 78:i159–i163, 2002.

[RAW$^+$10] Jason Reed, Adam J. Aviv, Daniel Wagner, Andreas Haeberlen, Benjamin C. Pierce, and Jonathan M. Smith. Differential privacy for collaborative security. In *Proceedings of the Third European Workshop on System Security*, EUROSEC '10, pages 1–7. ACM, 2010.

[RR10] Aaron Roth and Tim Roughgarden. Interactive privacy via the median mechanism. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 765–774. ACM, 2010.

[SA04] Jitesh Shetty and Jafar Adibi. The enron email dataset: Database schema and brief statistical report, 2004.

[Sha] Zed Shaw. Programmers need to learn statistics or i will kill them all. Retrieved, 2 Januari 2012, from http://zedshaw.com/essays/programmer_stats.html.

[Sha02] Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of The European Association for Theoretical Computer Science: Computational Complexity Column*, 77:67–95, June 2002.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 2nd edition, 2006.

[Swe01] Latanya Arvette Sweeney. *Computational Disclosure Control - A Primer on Data Privacy Protection.* PhD thesis, Massachusetts Institute of Technology, 2001.

[Swe02] Latanya Sweeney. k-anonymity: a model for precting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, October 2002.

[SZD08] Zak Stone, Todd Zickler, and Trevor Darrell. Autotagging facebook: Social network context improves photo annotation. *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2008.

[TMP11] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Social Networks*, 1102(February 2004):82, 2011.

[TSK06] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Education, 2006.

[Uni08] United States Senate Committee on Commerce, Science and Transportation. Testimony of chris kelly: Chief privacy officer at facebook. Retrieved, 6 December 2011, from `http://tinyurl.com/dxl25lz`, July 2008.

[WB85] Samuel D. Warren and Louis D. Brandeis. *The right to privacy*, pages 172–183. Wadsworth Publ. Co., 1985.

[Wil97] Graham J. Wills. Nicheworks - interactive visualization of very large graphs. In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 403–414, 1997.

[Won06] Nicole Wong. Judge tells DoJ “no” on search queries. *Official Google Blog*, March 2006.

[YGKX08] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy*, pages 3–17, 2008.