JORGENSEN, ZACHARY. Practical Private Data Analysis with Differential Privacy. (Under the direction of Dr. Ting Yu and Dr. Douglas Reeves.)

The goal of private data analysis is to extract useful aggregate knowledge from sensitive datasets without disclosing information about individual constituents. Early work on this problem focused on anonymization techniques, which have since been criticized due to the feasibility of attacks that can lead to disclosure of sensitive attributes. Consequently, more robust notions, led by differential privacy, are now preferred. Differential privacy is a powerful framework for providing strong, formal privacy guarantees that are robust against informed adversaries with access to auxiliary information. However, achieving strong differential privacy guarantees for complex analyses, without significantly harming utility, is non-trivial in general. This dissertation presents practical techniques for satisfying differential privacy for important tasks in the social network analysis domain, as well as introduces a new personalized variant of differential privacy, which offers significantly improved utility on datasets comprising users with diverse privacy requirements.

Specifically, our first work tackles the problem of generating item recommendations that are personalized on a user’s social network, while simultaneously preventing the disclosure of sensitive user-item preferences (e.g., product purchases, ad clicks, web browsing history, etc.). We propose and evaluate a privacy-preserving framework for a natural class of social recommendation algorithms and show that it is possible to attain strong differential privacy guarantees while maintaining good recommendation accuracy.

Our second work introduces personalized differential privacy (PDP), a novel generalization of differential privacy that enables contributors to a sensitive dataset to independently specify a personal privacy requirement for their data. We give several mechanisms for satisfying this new privacy definition and show that by taking personal privacy requirements into account, PDP offers a significantly higher level of utility for many analysis tasks, compared to traditional
differential privacy.

In our third work, we return to the social network analysis domain, where we focus on the problem of publishing social network graphs without disclosing the sensitive relationships and node attributes of the individuals contained in the graph. To that end, we present a differentially private framework that takes a sensitive social graph $G$ and generates a synthetic graph $\tilde{G}$ that mimics the important structural characteristics (e.g., degree distribution and clustering coefficients) and attribute correlations observed in $G$, without violating privacy. We experimentally evaluate our solution on four real-world social network datasets and show that it produces realistic synthetic graphs with strong privacy guarantees.
Practical Private Data Analysis with Differential Privacy

by

Zachary Jorgensen

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2015

APPROVED BY:

__________________________  _________________________
Dr. Rada Chirkova            Dr. Laurie Williams

__________________________  _________________________
Dr. Ting Yu                 Dr. Douglas Reeves
Co-chair of Advisory Committee
Co-chair of Advisory Committee
ACKNOWLEDGEMENTS

This has been a long journey and I have many people to thank for helping me get to this point.

First, I’d like to express my sincere appreciation to my advisor, Ting Yu, for his guidance, patience and financial support over the years. Ting encouraged my interests in privacy-preserving data analysis and introduced me to differential privacy, which is the foundation of all of the work in this dissertation. At the same time, he afforded me considerable freedom to explore interesting tangents and to develop my own research interests. Even after departing from NCSU, Ting continued to work closely with me to ensure that I was able to complete my research. Ting, thanks for being a wonderful advisor and for always making time for me when I needed guidance.

I’d like to thank the rest of my committee, Doug Reeves, Laurie Williams and Rada Chirkova for their time, helpful guidance and thoughtful feedback, which helped improve the quality of this dissertation. Additionally, I’d like to say a special thanks to Doug Reeves for taking the reigns as my co-advisor upon Ting’s departure. Doug, thanks for your support and for making yourself available when I needed advice.

I had the privilege of collaborating with Graham Cormode, from the University of Warwick, on two of the works in this dissertation (Ch. 4 and 5). Graham is a brilliant researcher and I learned a lot from working with him. His input helped to greatly improve the quality of those chapters and the associated papers. Graham, thanks for all of your help and friendly guidance.

I’d like to thank my lab mates: Entong, Wei, Alley, Ling, Vasant, Ashwin, Adwait, Ben and Al for attending numerous practice talks and for helpful discussions along the way.

My amazing family has shown me constant support and unconditional love, not only through my seemingly never-ending educational journey, but throughout my life. They never stopped believing in me, even when I did, and they helped me push through when I wanted to give up. I feel incredibly blessed to have such a loving and supportive family. I’m especially grateful to my mother, Kathy Jorgensen, who is without doubt the hardest working person I’ve ever
known. She instilled in me an appreciation for education and for hard work and has shown me unconditional love and support in everything that I have done. I’m also very grateful to my in-laws, Bill and Harriet Hartman, for their love and support, especially during the chaotic final months of my Ph.D. I’d also like to thank my dear friends Cody and Melinda Weaver for their friendship and for providing much needed distractions that kept me from burning out during my final year. They also graciously gave me a place to stay when the lease on my apartment ended before the completion of my dissertation.

Last, but certainly not least, I owe an enormous debt of gratitude to my wife, Rachael, for her constant love, support and patience. Rachael, thank you for being an amazing companion, for picking up my slack at home during deadlines and for never ceasing to believe in me. You encouraged me to keep going during my lowest lows and helped me celebrate during my high points. Thank you for patiently sticking by me all these years. I love you dearly.

Again, my sincere thanks to all those who had a hand in helping me complete this journey; it would not have been possible without you all.
# TABLE OF CONTENTS

**LIST OF TABLES** .................................................. vi

**LIST OF FIGURES** .................................................. vii

Chapter 1 Introduction .............................................. 1
  1.1 Motivation .................................................... 1
  1.2 Contributions ............................................... 4
  1.3 Organization ............................................... 6

Chapter 2 Background and Related Work ............................ 7
  2.1 Differential Privacy .......................................... 7
    2.1.1 Achieving Differential Privacy ....................... 8
    2.1.2 Properties of Differential Privacy ................. 11
  2.2 Related Work ............................................... 11

Chapter 3 Differential Privacy for Personalized Social Recommendations .......................... 18
  3.1 Motivation .................................................... 18
    3.1.1 Contributions ......................................... 21
  3.2 Model ......................................................... 22
    3.2.1 Setting .................................................. 22
    3.2.2 Personalized Social Recommenders ................. 22
    3.2.3 Assumptions and Attacker Model ................. 25
    3.2.4 Privacy Model ........................................ 26
    3.2.5 Measuring Recommendation Accuracy ............. 27
  3.3 Related Work ............................................... 28
  3.4 Proposed Framework ......................................... 31
    3.4.1 Rationale .............................................. 32
    3.4.2 Algorithm and Analysis .............................. 39
  3.5 Experimental Study ......................................... 43
    3.5.1 Datasets ............................................... 43
    3.5.2 Experimental Setup ................................... 44
    3.5.3 Analysis of Privacy-Accuracy Trade-off ........ 46
    3.5.4 Comparison with Other Approaches ............... 50
  3.6 Summary .................................................... 52

Chapter 4 Personalized Differential Privacy ....................... 54
  4.1 Motivation .................................................... 54
    4.1.1 Contributions ......................................... 56
  4.2 Related Work ............................................... 57
  4.3 Personalized Differential Privacy ......................... 60
    4.3.1 Privacy Definition .................................... 60
    4.3.2 Properties of PDP .................................... 62
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Mechanisms for Achieving PDP</td>
<td>64</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Baseline Mechanisms</td>
<td>65</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Achieving PDP via Sampling</td>
<td>66</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Direct Approach</td>
<td>71</td>
</tr>
<tr>
<td>4.5</td>
<td>Experimental Study</td>
<td>76</td>
</tr>
<tr>
<td>4.5.1</td>
<td>PDP for Count and Median</td>
<td>78</td>
</tr>
<tr>
<td>4.5.2</td>
<td>PDP for Multiple Linear Regression</td>
<td>83</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary</td>
<td>86</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>87</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Contributions</td>
<td>89</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Organization</td>
<td>90</td>
</tr>
<tr>
<td>5.2</td>
<td>Preliminaries</td>
<td>91</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Setting and Notation</td>
<td>91</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Attributed Graph Model (AGM)</td>
<td>91</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Privacy Model</td>
<td>95</td>
</tr>
<tr>
<td>5.3</td>
<td>Privately Computing the Model Parameters</td>
<td>96</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Attribute-Edge Correlations</td>
<td>97</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Node Attribute Distribution</td>
<td>107</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Structural Model: TriCycLe</td>
<td>108</td>
</tr>
<tr>
<td>5.4</td>
<td>AGM-DP: Putting It All Together</td>
<td>123</td>
</tr>
<tr>
<td>5.4.1</td>
<td>End-to-End Workflow</td>
<td>123</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Privacy Analysis</td>
<td>127</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Extensions</td>
<td>127</td>
</tr>
<tr>
<td>5.5</td>
<td>Empirical Analysis</td>
<td>129</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Datasets</td>
<td>130</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Experiment Setup</td>
<td>131</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Statistics Evaluated</td>
<td>132</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Results</td>
<td>133</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Discussion</td>
<td>138</td>
</tr>
<tr>
<td>5.6</td>
<td>Related Work</td>
<td>139</td>
</tr>
<tr>
<td>5.7</td>
<td>Summary</td>
<td>142</td>
</tr>
<tr>
<td>6.1</td>
<td>Review of Contributions</td>
<td>144</td>
</tr>
<tr>
<td>6.2</td>
<td>Future Work</td>
<td>147</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Graph and Network Analysis</td>
<td>147</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Differential Privacy in Practice</td>
<td>148</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Weaker Notions of Privacy</td>
<td>149</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
<td>150</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1  Summary of data sets. ....................................................... 44
Table 4.1  Experiment parameters (defaults underlined). .................... 78
Table 5.1  Summary of the notation used in Chapter 5 ........................ 92
Table 5.2  Datasets used in empirical analysis. The columns are number of nodes $n$, number of edges $m$, maximum degree $d_{max}$, average degree $d_{avg}$, number of triangles $n_\Delta$, and average local clustering coefficient $\bar{C}$. ............... 131
Table 5.3  Results for AGM-DP with the FCL (AGMDP-FCL) and TriCycLe (AGMDP-TriCL) structural models on the Last.fm dataset for different privacy settings, $\epsilon$. .................................................. 134
Table 5.4  Results for AGMDP-FCL AGMDP-TriCL on the Petster dataset for different privacy settings, $\epsilon$. .................................................. 134
Table 5.5  Results for AGMDP-FCL AGMDP-TriCL on the Epinions dataset for different privacy settings, $\epsilon$. .................................................. 135
Table 5.6  Results for AGMDP-FCL AGMDP-TriCL on the Pokec dataset for different privacy settings, $\epsilon$. .................................................. 138
LIST OF FIGURES

Figure 3.1 (Last.fm) Comparison of average $NDCG@N$ of each private social recommender for different settings of $\epsilon$ and $N$ on \textbf{Last.fm}. Note that the privacy guarantee varies from weak to strong as you move left to right along the x-axis. ................................................................. 45

Figure 3.2 (Flixster) Comparison of average $NDCG@N$ of each private social recommender for different settings of $\epsilon$ and $N$ on \textbf{Flixster}. (Note the different y-axis values used in this figure compared to Fig. 3.1.) .................. 46

Figure 3.3 Relationship between user degree (log scale) and $NDCG@50$ due to approximation error (i.e., $\epsilon = \infty$), observed for CN similarity measure on Last.fm (3.3a) and Flixster (3.3b). A similar relationship was seen for the other similarity measures. ................................................................. 47

Figure 3.4 Figs. 3.4a and 3.4b show $NDCG@50$ of the naïve NOU and NOE baselines, as well as adaptations of the recent LRM [135] and GS [61] approaches, on \textbf{Last.fm} with $\epsilon \in \{1.0, 0.1\}$. ................................................................. 48

Figure 4.1 (\textit{Count}) RMSE of each mechanism for the \textit{count} task, as four parameters are varied. ................................................................. 79

Figure 4.2 (\textit{Median}) RMSE of each mechanism in for the \textit{median} task, as four parameters are varied. ................................................................. 80

Figure 4.3 (\textit{Linear Regression}) RMSE of each mechanism for \textit{linear regression}, as four parameters are varied. ................................................................. 85

Figure 5.1 Adding or removing a single edge can create/destroy up to $n - 2$ triangles. ................................................................. 96

Figure 5.7 Fig. 5.7a provides a graphical overview of our differentially private adaptation of AGM, with Fig. 5.7b depicting the sampling step (corresponding to the right-most block in 5.7a). ................................................................. 125
Chapter 1

Introduction

1.1 Motivation

In our increasingly digital and connected society, large amounts of detailed, personal data are continuously collected and stored by commercial entities, governments and other organizations. Search engines, such as Google and Bing, record our search queries and the web sites we visit. Online stores, like Amazon, track the products we view and buy, even as brick-and-mortar stores track our purchases through loyalty card programs. Credit card companies and credit agencies track where we shop and how we spend, while telephone providers retain records of who we communicate with, how often and for how long. Recently, the explosive growth of online social networks, such as Facebook and Google Plus, has created vast stores of digital records of our social interactions, the groups we belong to, the places we visit and the things we like.

These data hold enormous value, both for the commercial world as well as for society in general. For example, purchasing and consumption records provide valuable insights for advertising and marketing, as well as input to recommendation systems, which help users discover useful content or products, while increasing sales for the provider. In the health sector, the rapidly increasing adoption of electronic medical records has created new opportunities for improving medical research and the quality of medical care [68]. In the government sector, there
are initiatives to utilize collected data to “enhance operational efficiency, transparency, citizen’s well-being and engagement in public affairs, economic growth, and national security [65]”.

However, mining and analyzing sensitive data creates important privacy challenges that must not be overlooked. Organizations want to benefit from user data but are under legal and ethical obligations to preserve the privacy of their customers, citizens or users. The goal of privacy-preserving data analysis is to extract useful patterns and knowledge from private data sets without disclosing the sensitive information of the individuals contained in the data. In the typical setting, a sensitive data set, comprised of data contributed by many individuals, resides with a trusted data holder. It is convenient to think of the data set as a table where each row, or record, corresponds to a single individual\(^1\). Data analysts who wish to conduct analyses on the sensitive data set must go through the data holder, rather than accessing it directly. We emphasize that the challenge here is different from that targeted by access control techniques; here, analyses may require access to an entire data set, and the goal is to ensure that the output of the analysis preserves the privacy of individuals, while providing useful information about the population as a whole. For example, we might want to learn whether there is a link between the consumption of diet soda and the development of Type 2 diabetes, without disclosing that Bob, a participant of the study, has diabetes.

Much of the prior work on this problem has centered around the idea of anonymization. For example, \(k\)-Anonymity, formalized in [118], anonymizes a data set by suppressing or generalizing identifying attributes of all records, such that every record is indistinguishable (in terms of its identifying attributes) from at least \(k - 1\) others. However, this approach has the flaw that if \(k\) individuals in an anonymity group all have same sensitive value (or all have “bad” values) then privacy is still violated. Moreover, \(k\)-anonymity, as well as subsequent anonymization approaches (e.g., [81, 128, 130]) do not hold up in the presence of collusion or adversaries with auxiliary information about the data set.

Within the last decade, differential privacy [26, 29] has emerged as a powerful framework for

---

\(^1\)In reality, data sets may contain multiple records for an individual and can take on other forms such as graphs.
providing strong, formal privacy guarantees for aggregate data analysis. Informally, differential privacy guarantees that the output of an analysis task will be almost the same regardless of whether any one record is present in the input data. Thus, an adversary who sees the output will not be able to infer whether any particular individual contributed his or her data to the input. Since an individual’s privacy risk does not increase significantly by participating in the analysis, privacy concerns are alleviated and the incentive for an individual to withhold (or falsify) their data is effectively removed. In contrast to prior attempts to define privacy, the guarantee that differential privacy provides does not depend on what auxiliary information an adversary may have about the input data: an adversary who knows the full input, except for one record \( r \), will still be unable to deduce even whether \( r \) was in the input [27].

A common approach for satisfying differential privacy for computations with numerical output\(^2\) is to inject random noise that has been carefully calibrated according to the computation’s sensitivity (i.e., the maximum impact that any one record can have on the output), and a configurable privacy parameter [31]. Since injecting random noise makes the result of the computation less precise, the challenge is to find a good balance between privacy and utility.

Since its introduction, differential privacy has grown into an active area of research and is increasingly regarded as the gold standard for privacy-preserving data analysis; however, achieving its strong privacy guarantees for many complex, yet common, analysis tasks remains an elusive goal. This dissertation contributes to the growing differential privacy literature by introducing novel applications of differential privacy to the tasks of social recommendation and attributed graph modeling and generation. We also introduce a new privacy definition that generalizes differential privacy to incorporate user-level privacy guarantees, and give novel mechanisms for satisfying the new definition. In addition to increased flexibility, our privacy definition and accompanying mechanisms offer the potential for significant utility gains (relative to differential privacy) by ensuring that users receive only the level of privacy that they individually require. The next section discusses each of our main contributions in detail.

\(^2\)Differential privacy can also be achieved for non-numeric functions using the exponential mechanism [88], which is discussed in Chapter 2.
1.2 Contributions

Our first work (Chapter 3) considers the problem of producing item recommendations that are personalized based on a user’s social network, while simultaneously preventing the disclosure of sensitive user-item preferences (e.g., product purchases, ad clicks, web browsing history, etc.). Our main contribution is a privacy-preserving framework for a class of social recommendation algorithms that provides strong, formal privacy guarantees under the model of differential privacy. Existing mechanisms for achieving differential privacy lead to an unacceptable loss of utility when applied to the social recommendation problem. To address this, the proposed framework incorporates a clustering procedure that groups users according to the natural community structure of the social network and significantly reduces the amount of noise required to satisfy differential privacy. Although this reduction in noise comes at the cost of some approximation error, we show that the benefits of the former significantly outweigh the latter. We explore the privacy-utility trade-off for several different instantiations of the proposed framework on two real-world data sets and show that useful social recommendations can be produced without sacrificing privacy. We also experimentally compare the proposed framework with several existing differential privacy mechanisms and show that the proposed framework significantly outperforms all of them in this setting.

Our second work (Chapter 4) highlights and addresses an important practical limitation of differential privacy—that the same level of privacy protection is afforded for all individuals, whether they want it or not. In practice, it is common that the data subjects have quite different expectations regarding the acceptable level of privacy for their data. Consequently, differential privacy may lead to insufficient privacy protection for some users, while over-protecting others. We argue that by accepting that not all users require the same level of privacy, a higher level of utility can often be attained by not providing excess privacy to those who do not want it. We propose a new privacy definition called personalized differential privacy (PDP), a generalization of differential privacy in which users specify a personal privacy requirement for their data. We
then introduce several novel mechanisms for achieving PDP. Our primary mechanism is a general one capable of automatically transforming any existing differentially private algorithm into one that satisfies PDP. We also present a more direct approach for achieving PDP, inspired by the well-known exponential mechanism [88]. We demonstrate our framework through extensive experiments on real and synthetic data. We show that in many cases, satisfying PDP, rather than differential privacy, yields a significantly higher utility, while meeting the privacy requirements of all users.

In our third work (Chapter 5), we return our focus to preserving privacy in the social network analysis (SNA) domain, where we explore the challenge of publishing attributed social networks with strong privacy guarantees. Specifically, our goal is to be able to generate synthetic graphs that approximate the characteristics of an input social network graph, while satisfying differential privacy. Compared to running analyses on an input graph directly, graph publishing has a number of advantages, as highlighted in [43]: it eliminates the need to compute the sensitivity\(^3\) of each analysis task, which can be highly non-trivial for complex analyses; it facilitates exploratory analyses, which would be otherwise unfeasible, given that every analysis performed directly on a graph leaks some information about the graph; and it enables an analyst to hide the details of an analysis (e.g., a proprietary algorithm) from the data owner. Our work advances the state of the art by considering social networks that have nodal attributes (e.g., gender, age, sexual preference, etc.) that may be correlated with the structure of the graph. Although attributed social graphs are common in practice, prior works on differentially private graph publishing have focused only on approximating graph structure. The ability to privately generate graphs that preserve attribute correlations as well as structural properties will enable the study of complex properties such as homophily\(^4\), which is a common phenomenon in social graphs [85]. To that end, we present a differentially private framework that takes a sensitive social graph \(G\) and generates a synthetic graph \(\tilde{G}\) that mimics the important structural

\(^3\)As will be explained in Chapter 2.1, differential privacy is often achieved by injecting noise into a computation, where the scale of the noise is based on the sensitivity of the computation—or the extent to which a small change in the input can affect the output.

\(^4\)Homophily refers to the tendency for similar nodes to form connections (edges) in a network.
characteristics (e.g., degree distribution and clustering coefficients) and attribute correlations observed in $G$, while protecting the relationships and attributes of the individuals represented in the graph. We experimentally evaluate our solution on four real-world social network datasets and show that it produces realistic synthetic graphs with strong, formal privacy guarantees.

1.3 Organization

In the next chapter, we provide a thorough review of the differential privacy model, its properties and the basic mechanisms commonly used to satisfy its definition. We also provide a comprehensive summary of related work in the area of differential privacy. Chapter 3 presents our work on privacy-preserving social recommendation. In Chapter 4, we introduce the notion of personalized differential privacy (PDP), a new privacy definition with per-user privacy guarantees. In Chapter 5 we present our differentially private framework for publishing sensitive attributed social networks. Chapter 6 contains concluding remarks and a summary of the research presented in this dissertation.
Chapter 2

Background and Related Work

In this chapter we introduce some background and notation common to all the works in this dissertation. In particular, we review the notion of differential privacy and the mechanisms commonly used to achieve it; we also state some of its useful properties. We then review the relevant literature surrounding differential privacy to help situate our work.

2.1 Differential Privacy

We typically model a data set as a set of tuples from a universe \( \mathcal{D} \), with one or more attributes \( A_1, \ldots, A_d \). The notion of neighboring data sets is an important one for differential privacy. We say that two data sets are neighboring if one is a proper subset of the other and the larger data set contains exactly one additional tuple.

**Definition 2.1.1 (Neighboring Data Sets).** Two data sets \( D, D' \subseteq \mathcal{D} \) are said to be neighboring, or neighbors, denoted \( D \sim D' \), if \( D \subseteq D' \) and \( |D'| = |D| + 1 \) (or vice versa). We write \( D \sim D' \) to denote that \( D \) and \( D' \) are neighbors and that \( t \in D' \) and \( t \notin D \).

An alternate definition of neighboring data sets says that \( D \) and \( D' \) differ only in the value of a tuple \( t \) (instead of in the presence of \( t \)); this alternate definition is used in the differential privacy literature when it simplifies the task at hand [30].
A mechanism \( \mathcal{M} : \mathcal{D} \to R \) is a randomized algorithm that takes a data set as input and returns an output from some range \( R \). The notation \( \mathcal{M}(x) \) then denotes the probability distribution on \( R \) induced by input \( x \). Informally, a mechanism is said to be \textit{differentially private} if the probability distribution \( \mathcal{M}(D) \) on any data set \( D \) is approximately the same as \( \mathcal{M}(D') \), for every \( D \sim D' \). In other words, the mechanism’s behavior should be (mostly) insensitive to the presence or absence of any one tuple in the input. More formally,

\textbf{Definition 2.1.2} (\( \epsilon \)-Differential Privacy [26, 31]). \textit{Mechanism} \( \mathcal{M} : \mathcal{D} \to R \) \textit{satisfies} \( \epsilon \)-\textit{differential privacy} if for all pairs \( D \sim D' \subset \mathcal{D} \) and any set \( O \subseteq R \) of possible outputs,

\[
\Pr[\mathcal{M}(D) \in O] \leq e^\epsilon \times \Pr[\mathcal{M}(D') \in O]
\]

This definition protects against, for example, an adversary who knows the full input except for any one tuple \( t \): they are still unable to deduce even whether \( t \) was in the input. In the definition, \( \epsilon > 0 \) is a publicly known \textit{privacy parameter} that controls the strength of the differential privacy guarantee: a larger \( \epsilon \) yields weaker privacy, while a smaller \( \epsilon \) leads to stronger privacy. Typical settings for \( \epsilon \) are 0.01, 0.1, or possibly, \( \ln 2 \) or \( \ln 3 \) [28]. When epsilon is small, \( e^\epsilon \approx 1 + \epsilon \).

\textbf{2.1.1 Achieving Differential Privacy}

For real valued functions, i.e., \( f : \mathcal{D} \to \mathbb{R}^d \), the most common way to satisfy differential privacy is to inject carefully chosen random noise into the output. The magnitude of the noise is adjusted according to the \textit{global sensitivity} of the function, or the maximum extent to which any one tuple in the input can affect the output. Formally,

\textbf{Definition 2.1.3} (Global Sensitivity [31]). \textit{The global sensitivity of a function} \( f : \mathcal{D} \to \mathbb{R}^d \), is

\[
\Delta f = \max_{D \sim D'} \| f(D) - f(D') \|_1
\]

where \( \| \cdot \|_1 \) is the \( L_1 \) norm.
Note that global sensitivity does not depend on the input data, but is a property of function \( f \) alone. A function \( f \) can be made \( \epsilon \)-differentially private by adding random noise drawn from the Laplace distribution with mean zero and scale \( \frac{\Delta f}{\epsilon} \) to its output. We will subsequently use the notation \( \text{Lap}(\lambda) \) to denote the Laplace distribution with mean 0 and scale \( \lambda \).

**Theorem 2.1.1** (Laplace Mechanism [31]). For a function \( f : \mathcal{D} \to \mathbb{R}^d \), the mechanism that returns \( f(D) + z^d \), where each \( z_i \) is drawn i.i.d. from \( \text{Lap}(\frac{\Delta f}{\epsilon}) \) satisfies \( \epsilon \)-differential privacy.

For functions where adding noise does not make sense or the output space is non-numeric, the exponential mechanism [88] can be used to achieve differential privacy. The exponential mechanism uses the concept of a score function, denoted \( s(D, r) \), that returns a real-valued score to indicate the quality of output \( r \) with respect to the true output, \( f(D) \). A higher score is assumed to mean that \( r \) is closer to (i.e., a better approximation for) the true output, \( f(D) \).

For a given score function \( s \), the exponential mechanism, denoted \( E_s^\epsilon \) is defined as follows.

**Theorem 2.1.2** (Exponential Mechanism [88]). For a score function \( s : \mathcal{D} \times \mathbb{R} \to \mathbb{R} \), the mechanism \( E_s^\epsilon(D) \) that outputs \( r \in \mathbb{R} \) with probability proportional to \( \exp\left(\frac{\epsilon s(D, r)}{2\Delta_s}\right) \), satisfies \( \epsilon \)-differential privacy.

Here, \( \Delta_s \) is the global sensitivity of the score function and is defined slightly differently than in the context of the Laplace mechanism: \( \Delta_s = \max_{D \sim D', r \in \mathbb{R}} |s(D, r) - s(D', r)| \).

**Smooth Sensitivity.** Global sensitivity, described above, is a worst-case measure of how much the output of a computation could change in response to a single change to the input. It is a function of the computation itself and is independent of the actual input data; that is, it must consider how a single change to any possible input could affect the corresponding output. Consequently, inputs that are possible, though extremely rare in practice, can result in a very high sensitivity. Although taking the actual input into account when computing sensitivity could lead to lower sensitivity, and hence less additive noise, doing so could also leak information about the data set through the noise magnitude itself.
To address the issues with global sensitivity, Nissim et al. [98] introduced the notion of smooth sensitivity. Rather than determining the noise scale based on the global sensitivity, they proposed using a smooth bound on the local sensitivity. Local sensitivity measures the maximum difference in the output of a function $f$ when run on true input $D$ and any neighboring input $D'$.

**Definition 2.1.4 (Local Sensitivity [98]).** For $f : \mathcal{D} \rightarrow \mathbb{R}^d$ and $D \in \mathcal{D}$, the local sensitivity of $f$ at $D$ is

$$LS_f(D) = \max_{D': D \sim D'} \|f(D) - f(D')\|_1.$$ 

Rather than using the local sensitivity directly, which can leak information about the true input $D$ (thus violating differential privacy), one can compute a smooth upper bound on the local sensitivity.

**Definition 2.1.5 (Smooth Sensitivity [98]).** For $\beta > 0$, the $\beta$-smooth sensitivity of a function $f$ at $D$ is

$$S_{f,\beta} = \max_{D' \in \mathcal{D}} \left( LS_f(D') \cdot e^{\beta d(D, D')} \right)$$

where $d(D, D')$ is the distance between $D$ and $D'$, i.e., the number of additions or deletions needed to turn $D$ into $D'$.

In [98] the authors show that adding noise according to the smooth sensitivity satisfies the slightly weaker $(\epsilon, \delta)$-differential privacy. For example, adding noise from $\text{Lap} \left( \frac{2S_{f,\beta}(D)}{\epsilon} \right)$ satisfies $(\epsilon, \delta)$-differential privacy, where $\beta = \epsilon / 2 \ln(1/\delta)$ [98].

Although smooth sensitivity has proven useful for certain analysis tasks, such as the median and minimum spanning tree [98] and various graph analysis tasks [126], it is non-trivial to compute smooth sensitivity for arbitrary functions, in general; moreover, it can also be computationally hard to compute for some functions (e.g., [80]).
2.1.2 Properties of Differential Privacy

An important practical property of differential privacy is composability.

**Theorem 2.1.3** (Sequential Composition [87]). Given a set of \( k \) mechanisms \( M_1, \ldots, M_k \), each of which independently satisfies \( \epsilon_i \)-differential privacy, if we run these mechanisms on an input \( D \) in sequence, then the full output is \( \epsilon' \)-differentially private, where \( \epsilon' = \sum_{i=1}^{k} \epsilon_i \).

In other words, when we perform multiple computations on the same data set (or a set of non-disjoint data sets), each mechanism leaks a little privacy, which accumulates; this leads to a graceful degradation of the overall privacy guarantee.

On the other hand, given a set of disjoint inputs, the resulting sequence of computations is \( (\max_{\epsilon_i}) \)-differentially private.

**Theorem 2.1.4** (Parallel Composition [87]). Given a set of computations, each of which independently satisfies \( \epsilon_i \)-differential privacy, the sequential execution of those computations on disjoint inputs satisfies \( \max_{\epsilon_i} \)-differential privacy.

The next theorem states that post-processing the output of a differentially private algorithm does not impact the privacy guarantee.

**Theorem 2.1.5** (Post-processing [63]). If \( M \) satisfies \( \epsilon \)-differential privacy, and \( A \) is a randomized algorithm whose input space is the output space of \( M \), and whose randomness is independent of both the input data and the randomness of \( M \), then \( M' \equiv M \circ A \) also satisfies \( \epsilon \)-differential privacy.

Using these composition theorems, one can easily construct complex differentially private algorithms from individual differentially private components.

2.2 Related Work

Since its introduction by Dwork et al. [26, 31] in 2006, an active research area has grown around the differential privacy model. The main challenge of differential privacy is how to satisfy the
definition without destroying the utility of the analysis. Consequently, much of the literature has focused on making differential privacy practical for specific applications and problem settings. We categorize the prior work based on the targeted setting or problem area below.

**Histograms.** Histograms are an important tool for summarizing the distribution of random variables. A significant amount of prior work [132, 45, 131, 3, 133, 36, 105] has focused on optimizing differential privacy for private histogram publishing, in which a data domain is partitioned into disjoint bins and the goal is to privately release counts of the number of records in each bin; the noisy counts can be used, for example, to answer arbitrary range queries over a data set.

**Correlated Workloads.** The *Matrix Mechanism (MM)* [71], and the closely related *Low-rank Mechanism (LRM)* [135] consider how to optimize utility for a workload of correlated, linear queries under differential privacy. In the MM framework, a data set is represented as a column vector of counts and the workload as a matrix in which each row corresponds to a query (i.e. a linear combination of the counts). The idea is to find a different, smaller set of queries, called a *strategy* that can be answered more accurately than the original workload (due to having a lower sensitivity), and to use noisy answers to the strategy queries to derive approximate answers for the original workload. While [71] formalizes the idea, they do not provide an efficient method for deriving suitable strategies for arbitrary workloads. The low-rank mechanism (LRM) [135] uses low-rank approximation to efficiently derive strategies. The *rank* of a workload matrix—-or “the number of non-negative eigenvalues”—is often small when there is a lot of correlation among the rows and columns of the matrix [135]. LRM uses this property to reduce the amount of noise required to answer low-rank workloads. Given an $m \times n$ workload matrix $W$, which contains linear queries over a data set $D$, LRM decomposes $W$ into matrices $B$ (of size $m \times r$, and $L$ of size $r \times n$, such that $r < m$ and $W \approx BL$. It then adds Laplace noise to $LD$, yielding $\hat{LD}$. Noisy answers to the original workload queries are then obtained via the product $BL\hat{D}$. The key is that $LD$ has lower sensitivity than $WD$, requiring less noise. The squared error of LRM is shown to scale linearly with the rank of the workload.
**Data Mining and Learning.** Data mining and machine learning under differential privacy has received significant attention. Several works have focused on the task of privately learning decision trees (e.g., [38, 50, 142]). Chaudhuri et al. [18] applied differential privacy to regularized logistic regression and support vector machines and proposed the idea of perturbing the objective functions of the minimization problem, rather than adding noise to the output; however, their approach only works with cost functions that are convex and doubly differentiable. Zhang et al. [140] generalized the idea to all types of optimization functions with their proposed functional mechanism. The problem of learning with kernels under differential privacy is also studied in [51]. Jagannathan et al. [49] studied differentially private learning in the setting where both private and public training data are available and they use the public data to “boost” the accuracy of the learned classifier. Mohammed et al. [92] combined generalization with differential privacy to publish data privately, which could then be used for data mining tasks, such as learning decision trees. Dishabi and Azgomi [25] use Haar wavelet transforms for differentially private data clustering. Differentially private frequent itemset mining (FIM) for transaction data is studied by [8] and [138]. The goal of FIM, in general, is to identify top $K$ sets of items that appear together most frequently in the input transactions.

**Recommendation Systems.** Initial work by McSherry and Mirinov [86] integrated differential privacy into item-based, collaborative filtering recommendation systems, such as those used in the NetFlix Prize competition. Machanavajjhala et al. [82] studied social recommendation systems, which recommend items based on a user’s social network. They concluded that generating social recommendations that were both accurate and preserved differential privacy was unfeasible. However, their analysis considered a graph model in which there was no distinction between the sensitivity of item preferences and social relations—both were considered to be private. Our work in Chapter 3 exploits the observation that social edges are typically public in many real-world social networks and proposes an effective technique for achieving differential privacy for the user-item edges while maintaining good utility.

**Event Streams.** Several recent works ([32, 17, 16, 13]) have focused on the problem of
providing differential privacy for continuous queries over event streams, where events arrive at a trusted aggregator at every time step and are represented with a binary value (i.e., a 1 means that some event of interest occurred at the current time step, while a 0 means it did not). Events in the stream are assumed to be independent and the goal is to provide event-level privacy—that is, two length $t$ streams are considered to be neighboring if they differ in at most one position (e.g., 101110 and 111110). In [32, 17], the authors consider the problem of publishing, at every time step, a running count of 1’s seen so far. A subsequent work [13] studies various decayed sum functions, including: window sums, exponentially decayed and polynomial decayed sums. Instead of considering a single query that counts the number of 1’s seen so far, [16] considers a setting in which there are multiple continuous sliding window count queries, and the sliding window may be defined differently for every query. Kellaris et al. [62] proposed the notion of $w$-event privacy, which generalizes event-level privacy on infinite streams. This notion protects events occurring within a set of $w$ consecutive time steps.

**Time Series Analysis.** Sever recent works [115, 107, 35] consider queries over time series data, in which there is a value associated with each user at every time step (e.g., a user’s current body weight) and the typical goal is to privately answer a query at each time step (e.g., “how many users weigh more than 200 lbs?”). Thus, each user may affect every query answer. In [107], the values for each user at different time steps are assumed to be correlated. The Fourier Perturbation Algorithm (FPA) proposed in [107] reduces the error for answering $n$ queries from $\Theta(n)$, using standard techniques, to $\Theta(k)$, where $k << n$. For a length $n$ time-series, the approach applies a discrete Fourier transform (DFT) to the $n$-length sequence of (non-noisy) query answers, yielding a sequence of $n$ Fourier coefficients. The first $k$ coefficients are then taken and perturbed, while the remaining coefficients are discarded; subsequently, the inverse DFT is applied to the perturbed coefficients to produce an $\epsilon$-differentially private approximation of the original sequence. [115] also considers periodic aggregation of time series data; however, unlike [107], their satisfies the weaker $(\epsilon, \delta)$-differential privacy and also assumes that there is no temporal correlation among a user’s events from different time steps. The primary focus of
the paper is the problem of computing a query at each time step in the presence of an untrusted aggregator. In [35, 34] an adaptive system for releasing real-time aggregate statistics over time-series data is presented. Like [107], each user’s events are assumed to be temporally correlated, but a trusted aggregator is assumed. They attempt to address the problem of real-time analysis; their goal is to release a count of the number of 1’s at each time step (in real-time) while providing user-level DP. The main idea behind the approach is to stretch the privacy budget over as much of the time series as possible by adaptively sampling the time series based on the detected dynamicity. Kalman filtering on dynamic data, in the context of \((\epsilon, \delta)\)-differential privacy is studied in [69].

**Location and Trajectory Analysis.** A trajectory is a temporal sequence of coordinates that represents the movement of an object (e.g., a person or vehicle) over a finite time period. [54] studied the problem of how to publish a ship’s entire trajectory sequence, with a publicly known starting point and terminal point, while preserving \(\epsilon\)-differential privacy. The proposed mechanism requires the full trajectory to be available and is therefore not suitable for real-time publishing of trajectories. [46] studies the problem of mining location history to discover “interesting geographic locations” under differential privacy. Traffic pattern analysis under differential privacy is studied in [94].

**Graph and Social Network Analysis.** Private network analysis has recently become a hot topic in the differential privacy community. Extending differential privacy to network data—where the input data set is now considered to be a graph—requires redefining the notion of “neighboring data sets”. Two main interpretations that have been studied in many recent works are edge privacy—neighboring graphs differ in the presence or absence of a single edge—and node privacy—neighboring graphs differ in the presence or absence of a single node and all incident edges [44]. In general, node differential privacy is significantly more difficult to satisfy, as real-world networks often do not have bounded degree, leading to a very high sensitivity for most tasks; consequently, most of the existing work has focused on edge privacy. Recent work has mainly followed one of two directions: (1) methods for privately computing specific
statistics over sensitive graphs, such as degree distribution [44], subgraph counting [58, 10, 139], clustering coefficient [126], and frequent subgraph mining [113]; and (2) private graph publication, which typically involves privately fitting a generative graph model to an input graph and then generating a representative synthetic graph (e.g., [91, 110, 125, 19, 129]). Very recently, a few works have attempted to address node privacy by introducing new notions of sensitivity [20], and the idea of using low-sensitivity transforms to project an input graph onto the set of degree-bounded graphs, which have lower global sensitivity [59, 10].

**Related Privacy Models.** Inspired by differential privacy, various closely related privacy definitions have emerged in the literature. Several of these related definitions introduce practical assumptions that relax differential privacy to enable better utility for difficult problems. For example, Machanavajjhala et al. [83] argued that assuming that an adversary has complete knowledge of all but one tuple in the input is unrealistic in some settings; they introduced $\epsilon$-privacy, which incorporates reasonable assumptions about the adversary’s background knowledge to improve utility. Several other related definitions [106, 70, 74] also involve placing restrictions on the distribution describing the adversaries prior knowledge. [7] formalizes the notion of noisless privacy, which as the name suggests, exploits the randomness that is already inherent in some data sets to quantify privacy, without the need to introduce additional noise. Gehrke et al. [40] proposed the notion of zero-knowledge privacy, which in contrast to the works above, is strictly stronger than differential privacy and is useful in the social network analysis domain where there is often significant correlation among connected individuals. Crowd-blending privacy [39] combines random sampling with a grouping technique to achieve differential privacy and the stronger notion of zero-knowledge privacy. Tramèr et al. [121] proposes a weaker characterization of differential privacy that enables a better trade-off between privacy and utility by weakening the adversary’s assumed prior belief about the contents of the database. Interestingly, they show that satisfying privacy with bounded priors is equivalent to satisfying $\epsilon$-differential privacy with a larger $\epsilon$ (i.e., less noise). Our work introducing personalized differential privacy (PDP) in Chapter 4 has a similar motivation to the works discussed
here, which is to circumvent differential privacy’s harsh impact on utility; however, our privacy notion does not weaken the privacy guarantee, rather it eliminates excess noise by considering individual privacy requirements.
Chapter 3

Differential Privacy for Personalized Social Recommendations

3.1 Motivation

Today’s Internet users are constantly overloaded with information, and recommendation systems have become an essential tool for separating relevant and interesting content from the mounds of chaff. Recent years have witnessed the astounding growth and penetration of online social networks, spurring interest in the idea of leveraging social relations to generate more useful recommendations. A number of early studies (e.g. [116, 41]) found that recommendations based on social connections are useful, and in many cases superior to those produced by traditional, collaborative filtering recommendation systems. More recently, a wide variety of social recommendation approaches have appeared in the literature, some of which augment traditional collaborative filtering approaches with social similarity measures (e.g. [77, 48, 137]) while others are based on random walks (e.g. [67]), probabilistic matrix factorization (e.g. [24]) and collaborative topic regression (e.g. [104]). In many cases, these social recommendation approaches have been shown to outperform traditional, socially agnostic approaches.

Although the benefits of using social information in the recommendation process can be
significant, it also poses a significant threat to personal privacy. In particular, social recommendations may allow one user to easily make valid inferences about another user’s preferences for sensitive items. Although this threat is also to some extent present in traditional, socially-agnostic recommenders, social recommenders are more vulnerable to privacy attacks for several reasons: (1) the recommendations are, by nature, derived from smaller, more targeted sets of users, (2) the recommendations sources are typically people known by the receiver, and (3) the users of a social network have considerable power to affect the structure of the social network (and therefore the input to the social recommender). These facts enable simple but powerful privacy attacks. For instance, a malicious (or simply curious) user, Bob, could easily manipulate the social graph so that another user, Alice, is Bob’s only real recommendation source (see Section 3.2.3); consequently, every recommendation that Bob receives reveals a potentially sensitive preference that Alice has for some item. In addition to protecting individual privacy and reducing liability for recommendation service providers, privacy-preserving approaches for social recommendation will likely encourage broader participation by providing users with “peace of mind”.

It is well known that syntactic privacy approaches, like k-anonymity [118], do not guarantee privacy in the presence of colluding adversaries or those with auxiliary information [81]. Thus, in this work we aim for provable privacy guarantees, enabled by a mathematically rigorous privacy model known as differential privacy [26]. Informally, differential privacy guarantees that almost nothing can be learned about individual input records based on the output of the algorithm; thus a user loses nothing by participating. Differential privacy is usually achieved by injecting random noise that is carefully calibrated according to an algorithm’s sensitivity—the maximum change in the output resulting from the addition or removal of a single input record. Intuitively, the sensitivity dictates how much random noise is necessary to mask the presence of any one individual’s data. In the case of social recommendation, one must consider how the recommendations of all users may be affected by changing a single item preference.

The main challenge in applying differential privacy to any task is how to balance privacy
with utility. Social recommendation presents an especially difficult challenge due to its inherent high sensitivity. The high sensitivity comes from the fact that a single user may have many neighbors in the social graph, and hence may affect the recommendations received by many users.

Machanavajjhala et al. [82] recently presented somewhat negative theoretical results suggesting the unfeasibility of producing social recommendations that are both accurate and private. However, the social graph model used in their analysis made no distinction between item preferences and social relations—both were considered to be equally sensitive. We observe that this is often not the case in practice. For instance, in many popular online social networks, including (but not limited to) Orkut, Twitter, MySpace, RenRen and Weibo, the user-to-user edges are visible to everyone, and there is no option to change that setting. Although Facebook and Google+ do allow users to hide their list of friends, they are publicly visible by default; moreover, a recent study [55] found that less than half of Facebook users change the default setting. On the other hand, item preferences are typically considered to be sensitive in practice; for example, the content that a user “likes” is usually only shared with the user’s friends. Item preferences could also represent behavior such as ad clicks, product purchases, photo views, or website visits—these are all things that might be useful in a social recommendation system, but they are also things that most people would not want displayed to the public, or even to their friends in some cases. Motivated by the observations above, we explore differentially private social recommendation in the setting where social relations are public knowledge, while user-to-item edges are private and must be protected. Within this context, our task is to generate top-n ranked lists of item recommendations, personalized for each individual user based on his/her social relations, while simultaneously preventing the leakage of information about the presence (or absence) of individual item preferences through the output (i.e., the recommendations themselves).
3.1.1 Contributions

We present a formalization of the social recommendation problem that explicitly differentiates user nodes from item nodes and social relations from item preferences; this distinction enables us to decouple the computations on the typically insensitive social graph from those on the often sensitive item preferences.

Next, we develop a privacy-preserving framework for a natural class of social recommendation algorithms based on structural similarity measures, that provides differential privacy guarantees for user-item preferences. Our approach achieves strong privacy and high utility by incorporating a novel user clustering phase that groups users according to the social graph structure, significantly reducing sensitivity, and hence the amount of noise required to satisfy differential privacy. Although the clustering phase introduces some approximation error, we show that its impact is small compared to that of the noise it replaces. To the best of our knowledge, this is the first work to propose an effective approach for achieving differential privacy in social recommendation systems, without significantly degrading accuracy.

Finally, we present empirical results for several concrete instantiations of the proposed framework on two real-world data sets, demonstrating that it is possible to make accurate social recommendations while simultaneously providing a high degree of privacy for item preferences. Additionally, we experimentally compare our approach against two traditional methods for achieving differential privacy, as well as to two recently proposed differential privacy mechanisms ([135, 61]) that have proven successful in similar settings. Our results show our approach to significantly outperform these other approaches under both high and low privacy settings.

The rest of this chapter is organized as follows. Section 3.2 formalizes our model and setting, while Section 3.3 reviews relevant related work. In Section 3.4 we present our framework for privacy-preserving social recommendation. Section 3.5 presents our experimental results and Section 3.6 concludes the chapter.
3.2 Model

In this section we formalize our setting and specify the basic class of non-private social recommenders targeted by our privacy-preserving framework.

3.2.1 Setting

Our setting assumes the existence of two distinct graphs, a social graph and a preference graph, that comprise the input to a social recommendation system. They are defined as follows:

Definition 3.2.1 (Social Graph). A social graph, $G_s = (U, E_s)$, consists of a set of user nodes, $U$, and a set of edges, $E_s$, where a social edge $(u, v) \in E_s$ represents a social connection (e.g. friendship) between two users $u, v \in U$.

Definition 3.2.2 (Preference Graph). A preference graph, $G_p = (U, I, E_p)$, is a bipartite graph consisting of the same set of users $U$, a set of items, $I$, and a set of directed edges, $E_p$. A preference edge $(u, i) \in E_p$ expresses a positive preference of user $u \in U$ for item $i \in I$.

For simplicity, we assume that the preference graph is unweighted, or equivalently that every edge $(u, i)$ has weight $w(u, i) = 1$. For notational convenience, we assume that $w(u, i) = 0$, $\forall (u, i) \notin E_p$. Many real-world recommendation settings can be modeled with unweighted preference graphs; for example, an edge $(u, i)$ might indicate that $u$ has purchased product $i$, that $u$ has listened to a song by artist $i$, that $u$ ‘Likes’ content $i$, or that $u$ has visited web page $i$, to name a few. Our approach could be easily extended to support weighted edges (e.g., ratings).

3.2.2 Personalized Social Recommenders

We consider a family of personalized, social, top-$n$ recommendation systems (henceforth referred to simply as social recommenders, or just recommenders where the context is clear) that are
based on a utility function that takes only $G_s$ and $G_p$ as input and computes for all user-item pairs $v, i$ a utility value $\mu_u^i$ indicating the utility of recommending item $i$ to user $u$. The social recommender outputs, for each target user $u \in U$, a personalized recommendation list, $R_u$, consisting of the top-$n$ items, ranked by utility.

Our model assumes that the utility function depends on an underlying structural similarity measure, denoted $sim(u, v)$, that operates solely on the structure of $G_s$ and returns a positive numeric value indicating the similarity of users $u, v \in U$ (or 0 if not similar). Since these similarity measures operate solely on the social graph, we refer to them henceforth as social similarity measures. Abusing this notation slightly, we will frequently use $sim(u)$ to denote the set of users with non-zero similarity to $u$, i.e. $sim(u) = \{v \in U | sim(u, v) > 0\}$. We call $sim(u)$ the similarity set of $u$.

Many existing structural similarity measures that can be plugged into this model (see [79] for a good survey covering many different measures). Prior work in the link analysis literature has demonstrated the usefulness of structural similarity measures for predicting missing edges in social networks [75], and other graphs [79]; moreover, several measures have been shown to outperform collaborative filtering approaches for recommendation on certain data sets [48], making them appealing candidates for the social recommendation task. For concreteness, we will consider four well-known social similarity measures in this work. Let $\Gamma(u)$ denote the set containing $u$’s immediate neighbors in $G_s$. In the context of two arbitrary users $u, v \in U$, the four measures are defined briefly below (see [75, 79] for more details):

- **Common Neighbors (CN)**: two users $u, v \in U$ are similar if they share many common neighbors; formally, $sim(u, v) = |\Gamma(u) \cap \Gamma(v)|$.

- **Graph Distance (GD)**: the similarity of $u$ and $v$ is equal to the the reciprocal shortest path distance between them, i.e., $sim(u, v) = \frac{1}{d}$, where $d$ is the length of the shortest path between $u$ and $v$.

- **Adamic/Adar (AA)**: AA is similar to CN but gives more weight to users in the inter-
section with low degree; it is defined as $\text{sim}(u, v) = \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(x)|}$.

- **Katz (KZ):** in this path-based measure, the similarity of $u$ and $v$ is based on the number of short paths between them; it is defined as $\text{sim}(u, v) = \sum_{l=1}^{k} \alpha^{l} \cdot |\text{paths}^{l}_{uv}|$, where $\text{paths}^{l}_{uv}$ is the set of all length-$l$ paths between $u$ and $v$, and $\alpha$ is a small damping factor (e.g., 0.05 or 0.005).

For GD and KZ it is common to limit $d$ and $k$ (e.g., to 2 or 3) respectively, since in social graphs, the number of reachable users explodes after two hops due to the small-world property [96] of social graphs.

The utility of recommending an item $i$ to a user $u$ is computed by a utility query:

**Definition 3.2.3 (Utility Query).** Given a similarity measure, $\text{sim}$, the utility of recommending an item $i$ to user $u$ is given by:

$$\mu_{u}^{i} = \sum_{v \in \text{sim}(u)} \text{sim}(u, v) \times w(v, i) \quad (3.1)$$

We now formally define top-$N$ social recommender as follows:

**Definition 3.2.4 (Top-N Social Recommender).** Given a social graph $G_{s} = (U, E_{s})$, a preference graph $G_{p} = (U, I, E_{p})$ and a social similarity measure $\text{sim}$, a top-$N$ social recommender identifies, for all users $u \in U$, a size $N$ ranked list of items $R_{u} \subset I$ that have the highest utility with respect to user $u$.

**Remark.** Although it could be beneficial to use both social and non-social data in the recommendation process, our focus is on purely social recommenders in this work. Enabling differential privacy guarantees for such hybrid recommenders would be an interesting subject for future work.
3.2.3 Assumptions and Attacker Model

The social recommender, as well as the underlying data, is assumed to reside with a trusted and secured central party and that the recommender has full, unfettered access to the data. In practice, the recommender would typically (though not necessarily) reside with the owner of the preference data, while the social graph data would be obtained from a third-party social network through an API (e.g., Facebook’s Open Graph, Twitter API, etc.). Our concern is with preventing the sensitive preference information from being leaked through the social recommender’s output (i.e., the recommendations themselves). We remark that the trusted server model is not uncommon in previous works on differential privacy; in particular, the work of [86] that studies differential privacy for non-social recommenders makes the same assumption.

Second, we assume that $G_s$ and $G_p$ are static at the time that recommendations are computed. That is, we take a snapshot of the graphs at time $t_i$ and generate recommendations for all users from that snapshot. In this work, we focus on the problem of making private social recommendations only for a single snapshot. Enforcing differential privacy over dynamic graphs is a non-trivial extension and we leave it as a subject for future work. We remark that most of the related works discussed in this work are also subject to the same limitation (e.g., [86, 61, 135]).

Adversary Model. We consider a powerful, informed adversary $A$, whose goal is to deduce the existence (or absence) of a specific preference edge, called the target edge. $A$ is assumed to have complete knowledge of the social graph and of the inner-workings of the recommendation algorithm. Also, consistent with the model of differential privacy, $A$ could potentially have knowledge of all preference edges in $G_p$, except for the target edge. Finally, we assume $A$ may observe all recommendations output by the social recommender. The latter two assumptions account for possible collusion among users, the possibility of a single user controlling multiple user accounts, and arbitrary background knowledge that may be available to the adversary.

The adversary modeled above may, at first glance, seem unrealistically powerful; however, we argue, by way of the following example, that this is in fact the appropriate model. For
concreteness, consider the CN (or AA) similarity measure defined in Section 3.2.2. One possible attack proceeds as follows: (1) $A$ identifies an immediate neighbor $a$ of the victim in the social graph that has no other neighbors; (2) $A$ then creates a fake user account $b$ and establishes a new social edge $(b, a)$, either through collusion or a simple profile-cloning attack [9]. A profile cloning attack involves setting up a fake node to impersonate one of the victim’s existing friends and then issuing a friend request to the victim\(^1\). (3) $A$ observes the recommendations received by $b$. Since $b$’s only friend is $a$, and only the victim and $b$ are friends with $a$, all recommendations received by $b$ reveal that victim has a positive preference for some item. Note that in the case where the victim has no neighbors with a single friendship edge, the attacker could link two fake nodes together and then trick the victim into accepting a friendship request from one of them using a profile cloning attack. This attack could also be extended to the GD and KZ similarity measures by chaining $d−1$ or $k−1$ fake nodes together, respectively, depending on the distance cutoff parameters (i.e., $d$ and $k$) in use by those similarity measures.

The thing to note from the example above is that, from the defender’s point of view, any node in the graph could be the attacker, any node could be the victim, and there is no reliable way to know which nodes are colluding; moreover, there is no reliable way to know which edge is the target edge and which edges the attacker has knowledge of due to auxiliary information. Thus, to ensure privacy, we must provide privacy guarantees for every preference edge as though it were the target edge. Differential privacy allows us to make such guarantees.

### 3.2.4 Privacy Model

To meet the challenges of our adversary, we adopt the mathematically rigorous differential privacy model [26], which offers provable privacy guarantees against such adversaries. A thorough review of differential privacy was presented in Chapter 2. Extending the differential privacy definition (Def. 2.1.2) to our setting, we have the following:

\(^1\)In a recent study involving actual Facebook users, [9] showed a success rate $> 60\%$ for this simple attack.
**Definition 3.2.5** (Private Social Recommender). For a fixed social graph, $G_s$, a social recommender $R$ is $\epsilon$-differentially private if for all preference graphs $G_{p1} = (U, E_{p1})$ and $G_{p2} = (U, E_{p2})$ that differ by at most one edge (i.e. $E_{p1} \subseteq E_{p2}$ and $|E_{p2} - |E_{p1}| \leq 1$, or vice versa), and for any subset $S$ of the possible recommendation lists in $\text{Range}(R)$, 

$$Pr[R(G_{p1}, G_s) \in S] \leq \exp(\epsilon) \times Pr[R(G_{p2}, G_s) \in S]$$

Thus, a social recommender preserves the privacy of preference edges if the probability of outputting any list of recommendations is nearly the same, with or without any one preference edge. Recall that an algorithm with sensitivity $\Delta$ can be made $\epsilon$-differentially private by adding random noise drawn from $\text{Lap}(\frac{\Delta}{\epsilon})$ to its output (Theorem 2.1.1). The parameter $\epsilon > 0$ controls the strength of the privacy guarantee, where a smaller $\epsilon$ yields stronger privacy. The variance of $\text{Lap}(\lambda)$ is $2\lambda^2$, and thus the expected error is $\sqrt{\text{var}(\text{Lap}(\lambda))} = \frac{\sqrt{2\Delta}}{\epsilon}$.

### 3.2.5 Measuring Recommendation Accuracy

The primary challenge to incorporating differential privacy into social recommenders (and indeed any task) is how to do so without significantly harming the utility, or accuracy of the output. In order to validate our approach, we will need a way to measure the accuracy of our private social recommenders relative to their non-private counterparts (i.e., an instance of definition 3.2.4 with the same similarity measure and inputs). Common accuracy metrics, such as precision and recall, are inappropriate for this task because they fail to take into account the rank and utility of the items. For example, with precision and recall, failing to recommend the top ranked item would incur the same penalty as failing to recommend the $N^{th}$ item, even though higher ranked items have higher utility for the user. Similarly, if an item $i$ with utility $\mu$ appears in the true ranking but is replaced by a different (but equal-utility) item $j$ in the private ranking, it will incur an unnecessary penalty.

Based on these considerations, we adopt the **normalized discounted cumulative gain (NDCG)**
as our accuracy metric in this work. Formally, let $\hat{A}$ denote a top-$N$, private social recommender and let $\hat{R}_u^N$ denote the list of ranked recommendations output by $\hat{A}$ for user $u$. Likewise, let $A$ denote the non-private counterpart to $\hat{A}$ and let $R_u^N$ denote the ranked recommendations output by $A$. We will use $\mu_u^i$ to denote the ideal utility for item $i$ with respect to user $u$—that is, the utility computed by $A$. Then, the NDCG for the top $N$ recommendations, for a given data set, is defined as:

$$NDCG@N = \sum_{u \in U} \frac{DCG(\hat{R}_u^N, u)}{DCG(R_u^N, u)} \times \frac{1}{|U|} \quad (3.2)$$

where $DCG(X, u) = \sum_{i \in X} \frac{\mu_u^i}{\max(1, \log_2 p(i) + 1)}$, and $p(i)$ is the index of item $i$ in $X$. This metric essentially computes the accuracy of a recommendation list by summing up the true utility of each item, logarithmically discounted by the item’s position in the list of recommendations, thus giving more weight to the items near the top of the list. An $NDCG@N$ of one indicates that the top $N$ items recommended by the private recommender are perfectly ranked by their true utilities, while an $NDCG@N$ of zero has the opposite meaning.

In the next section, we briefly discuss related work. We then present our proposed framework in Section 3.4.

### 3.3 Related Work

Aside from the previously mentioned theoretical work of Machanavajjhala et al. [82], there have been few works to study privacy in social recommendation. Danezis et al. [23] proposed a k-anonymity like approach for private social recommendations, which breaks the social graph up into $k$-plexes\(^2\) and subsequently propagates item preferences from each member of a $k$-plex to all other members. This approach, like k-anonymity in general, is not resistant to collusion or auxiliary information attacks. It is precisely these limitations that motivate our use of differential

\(^2\) A $k$-plex is a clique with up to $k$ missing edges.
privacy in the present work, which provides strong guarantees in the presence of such attacks.

McSherry and Mirinov [86] integrated differential privacy into traditional, non-social recommendation systems, such as those used in the NetFlix Prize competition. Specifically, they considered item-based collaborative filtering algorithms that make recommendations to users based on a sanitized, global, item-item covariance matrix that is constructed using item ratings contributed by all users. This is different from our setting, in which the recommenders are necessarily user-based and where recommendations are personalized according to each user’s social network. In the social setting, the sensitivity—and hence the amount of noise required to satisfy differential privacy—grows with the size of the social graph, which can be very large in practice.

Private approaches for computing social recommendations in distributed settings have recently been proposed [47, 57]. However, these approaches use cryptographic techniques to eliminate the need for a central aggregator with access to all of the raw preference data; these approaches do not prevent privacy leaks through the output of the recommendation system (i.e., the recommendations themselves), which is our goal in the present work. We consider a centralized setting in the present work, where the recommendation system resides at a secure and trusted central party, with unfettered access to the data.

Since differential privacy was first formalized, a significant body of work has emerged leading to more advanced mechanisms for achieving differential privacy. We now briefly review a selection of recent works that we feel are most relevant to the present work. A common approach for reducing the perturbation error introduced by differential privacy in high-dimensional domains is generalization. For instance, in [42] user locations are generalized (e.g., from landmark up to zip code up to city up to state, etc.) to increase the magnitude of counts relative to the noise being added. Since the magnitude of the noise is independent of the number of records used to compute the counts, larger counts yield a smaller relative error. In contrast, the clustering techniques used in the present work perform grouping rather than generalization and do not require a pre-defined taxonomy tree.
A number of recent works (e.g., [132, 45, 131, 3, 133]) focus on private histogram publishing, where the goal is to release a noisy version of a histogram that can be used to answer arbitrary range queries. The general idea behind many of these approaches is to group similar counts, where the counts associated with each bin have a sensitivity of one, since a single record in the underlying database affects only one count. The basic idea is to group the bins of the histogram such that counts of the bins in each group are similar and can be approximated by their noisy mean. While histogram counts have a sensitivity of one, the sensitivity of the utility queries in our setting can be arbitrarily high, as a single preference edge may affect an arbitrary number of query answers. Therefore, these approaches are unsuitable for our task.

Differentially private frequent itemset mining (FIM) for transaction data is studied by [8] and [138]. The goal of FIM, in general, is to identify top \(K\) sets of items that appear together most frequently in the transactions. While this task may appear similar to the social recommendation problem in some respects, a critical difference is that in the FIM task, a single global ranking is produced, while in social recommendation a personalized ranking is produced for every user from overlapping sets of preference edges. Personalization implies significantly higher sensitivity, and hence more noise.

Recently [107] proposed a dimensionality reduction approach for privately answering queries on time series data, where a single user’s data affects every query answer. The approach produces a private approximation of a sequence of counts by first applying a discrete Fourier transform (DFT), then adding noise to a subset of the resulting Fourier coefficients, and finally applying the inverse DFT. This approach could be adapted to our setting by viewing the set of utility values as a timeseries, however this approach does not take advantage of query structure and thus leads to much higher levels of noise. It is useful mainly for time series data and is known to perform poorly when there is no natural ordering to the sequence of counts [61].

The Matrix Mechanism (MM) [71], and the closely related Low-rank Mechanism (LRM) [135] consider how to optimize utility for a workload of correlated, linear queries under differential privacy. The idea is to find a different, smaller set of queries, called a strategy that can
be answered more accurately under differential privacy than the original workload, and to use the noisy answers to the strategy queries to derive approximate answers to the original workload. This idea was formalized in [71] but no efficient method for deriving suitable strategies for arbitrary workloads was provided. A more recent work [135] proposes the low-rank mechanism (LRM), which uses low-rank approximation to derive the strategy queries for an arbitrary workload. In Section 3.5, we show that LRM can be adapted naturally to our setting, but with negative empirical results.

Very recently, [61] considered a similar problem to ours (although in an entirely different context), in which the goal was to privately publish a set of counts, where a single user’s data can affect multiple counts. For example, each count may correspond to a location (e.g., a restaurant or place of business) and the count indicates the number of users that have “checked-in” at that location. Each user may check-in at multiple places, and thus may affect multiple counts. The goal is to publish the set of counts while preserving user-level differential privacy; that is, the impact of a single user’s data (e.g., all of his check-ins) on the set of counts should be masked. The proposed group and smooth (GS) approach involves grouping similar counts and approximating the counts in each group by the noisy mean of the group. A sampling technique is proposed for finding an approximately optimal grouping based on rough, noisy estimates of the counts. In Section 3.5.4, we show that the ideas used in GS can be adapted to our task, but that the resulting approach is unable to effectively cope with the high sensitivity of our task.

### 3.4 Proposed Framework

Our goal in this paper is to create a framework that can be used to easily transform non-private, top-n social recommenders (as defined in Section 3.2.2), into differentially private social recommenders that are able to provide non-trivial privacy guarantees while also maintaining good recommendation accuracy for most users.

At a high level, our approach involves clustering the preference edges of $G_p$ into disjoint
clusters such that the edges in each cluster frequently co-occur in many different utility queries. This process minimizes the sensitivity of the utility queries and greatly reduces the amount of noise required to satisfy differential privacy. However, the clustering process essentially trades perturbation error for approximation error and the challenge is to find a clustering such that the approximation error is outweighed by the reduction in perturbation error. The problem is further complicated by the fact that if the preference graph is used to guide the clustering process in any way, then additional noise (and hence additional perturbation error) must be injected to ensure that no privacy is leaked through the clustering process. The surprising finding of this paper is that the insensitive structure of the social graph alone can be used to guide the creation of an effective clustering, without incurring any additional perturbation error.

3.4.1 Rationale

To better explain the rationale behind our approach, we begin with a high-level discussion of two strawman approaches that satisfy differential privacy, but that fail to provide useful recommendation accuracy. We then use the short-comings identified in the strawman approaches to motivate our proposed user clustering approach, which achieves the same level of privacy, but with a significantly higher utility. We start by making some simplifying observations and introducing additional notation to simplify the discussion.

Our goal is to design a differentially private algorithm that takes a social graph $G_s = (U, E_s)$ and a preference graph $G_p = (U, I, E_p)$ and returns, for every user in $U$, a personalized list of the top $N$ items ranked by utility. Recall that in our model, generating a list of recommendations for an arbitrary user $u$ involves first computing the set of utility queries $\mu_u = \{\mu_u^i | i \in I\}$, then sorting the items in $I$ according to their utilities $\mu_u$, and finally returning the top $N$ items. Our challenge, then, is to design an algorithm that can compute perturbed answers to the utility queries in a differentially private manner; if we can answer the utility queries privately, then the act of sorting and outputting the top $N$ items also preserves privacy, since post-processing
over sanitized data does not pose any additional privacy risk.

A second simplifying observation is that answering utility queries for an arbitrary item $i$, i.e., $\mu^i_u = \sum_{v \in \text{sim}(u)} \text{sim}(u, v) \cdot w(v, i)$, requires only the social graph (to compute the similarity scores) and the subset $G^i_p \subset G_p$ of preference edges to item $i$, i.e., $G^i_p = (U, i, E^i_p = \{(v, i) \in E_p | v \in U\})$. Let $A$ denote the algorithm that takes as input $G_s = (U, E_s)$, an item $i \in I$ and the subgraph $G^i_p$, and outputs the utilities for all users with respect to $i$, i.e., $A(G_s, i, G^i_p) = \{\mu^i_u | u \in U\}$. If we can devise an $\epsilon$-differentially private version of $A$, denoted $\hat{A}$, then it follows from parallel composition (Theorem 2.1.4), that calling $\hat{A}(G_s, i, G^i_p)$ for all $i \in I$, in sequence, also satisfies $\epsilon$-differential privacy. Thus, our focus in the following discussion will be on devising algorithm $\hat{A}$. At the end of our discussion, we will present pseudocode for the end-to-end algorithm, as well as a proof of its privacy guarantees.

3.4.1.1 Strawman Approaches

One naïve approach, which we will refer to as Noise on Utility (NOU), to satisfying $\epsilon$-differential privacy for $A$ involves a direct application of the Laplace mechanism (Theorem 2.1.1) to the utility values output by $A$. That is, $\hat{A}(G_s, i, G^i_p) = \{\mu^i_u + \text{Lap}(\frac{\Delta A}{\epsilon}) | u \in U\}$, where each $\text{Lap}(\frac{\Delta A}{\epsilon})$ is drawn independently from the Laplace distribution with mean 0 and scale $\frac{\Delta A}{\epsilon}$. The sensitivity $\Delta_A$ is determined by considering the greatest possible impact that adding or removing any arbitrary preference edge could have on the output of $A$, i.e., $\Delta_A = \max_{u \in U} \sum_{v \in U} \text{sim}(v, u)$. Thus, although this approach satisfies $\epsilon$-differential privacy, the sensitivity may be arbitrarily large in practice. Intuitively, the high sensitivity comes from the fact that a single preference edge $(u, i)$ may affect the computed utility value for an arbitrarily large number of other users, depending on how well $u$ is connected in the social graph. For most social similarity measures, the sensitivity will be determined by the user with the highest degree in the social graph and, consequently, the magnitude of the noise introduced into each utility value will greatly exceed the actual value, leading to very poor results. Our experimental results (Section 3.5) confirm that NOU leads to nearly a complete loss of utility, even for very lenient privacy settings (e.g.,
A second naïve approach, which we refer to as Noise on Edges (NOE), involves injecting independent Laplace noise directly into the weight of each preference edge (i.e., \( \hat{w}(u, i) = w(u, i) + \text{Lap}(\frac{1}{\epsilon}) \)) and using the resulting sanitized edges as input to the original algorithm \( A \). Note that non-existent edges are represented as edges with a weight of zero. The expected error for an arbitrary utility query \( \mu^i_u \) would be \( \frac{\sqrt{2}}{\epsilon} \sum_{v \in \text{sim}(u)} \text{sim}(u, v) \). Meanwhile, the true value of \( \mu^i_u \) is at most \( \sum_{v \in \text{sim}(u)} \text{sim}(u, v) \) (but often much less, due to the sparsity of real-world preference graphs). Thus for any non-trivial \( \epsilon \) the error is expected to drown out the true signal, leading to poor recommendation accuracy. We demonstrate the poor accuracy of this approach via empirical results in Section 3.5.

### 3.4.1.2 Proposed Approach

The approach that we propose in this paper is essentially an extension of NOE. The general idea is to group multiple preference edges together, forming disjoint clusters, and then to add noise to the average edge weight of each cluster. We then use the noisy averages in place of the true weights to derive privacy-preserving estimates for the utility values, which are then used to determine what items to recommend. As we will explain, this allows us to spread a small quantity of noise over multiple edges. More formally, let \( E^i_p \) denote the set of preference edges in \( G^i_p \) with the addition of zero-weight edges for all \( (v, i) \notin G^i_p \). Let \( S(E^i_p) = \Phi_S = \{c_1, \ldots, c_n\} \) denote a clustering of \( E^i_p \) into disjoint clusters \( c_i \), produced using a clustering strategy \( S \). For now, let us assume that \( S \) simply clusters the edges randomly, without regard to their weights.

Equation (3.3) formalizes the process of computing the noisy averages, while equation (3.4) uses them to derive a perturbed estimate \( \hat{\mu}^i_u \) for utility query \( \mu^i_u \) (i.e., the utility of recommending \( i \) to \( u \)). Note that since we are only considering the preference edges for a single arbitrary item \( i \) at the moment, one may think of a cluster as containing users or preference edges, interchangeably—each user is associated with a single preference edge, which in-turn belongs

---

\(^3\)We assume, in the worst case, that the adversary knows \( S \); that is, he knows which edges map to which clusters.
to a single cluster. We will use this fact to simplify the equations and discussions that follow.

\[ \bar{c} = \text{Lap} \left( \frac{1}{|c|\epsilon} \right) + \sum_{v \in c} \frac{w(v, i)}{|c|} \]  

(3.3)

\[ \hat{\mu}_u^i = \sum_{c \in \Phi_S} \sum_{v \in \text{sim}(u) \cap c} \text{sim}(u, v) \cdot \bar{c} \]  

(3.4)

Note that the sensitivity of the Laplace noise in equation (3.3) is \( \frac{1}{|c|} \), because adding/removing a single edge \( e \) to/from \( E_p^i \) could change \( \bar{c} \) by at most \( \pm \frac{1}{|c|} \), where \( c \) is the cluster to which \( e \) was assigned by \( S \). To clarify, since \( S \) randomly assigns edges to clusters without regard to their edge weights, adding or removing edge \( e \) (which, is equivalent to changing its weight to one or zero, respectively) does not change \( e \)'s cluster assignment; thus, the average edge weight for only one cluster is affected by the addition/deletion. Moreover, since \( S \) does not use the sensitive edge weights to determine the clustering, it does not violate privacy.

Computing the utility estimates with equations (3.3) and (3.4) introduces two sources of error into each estimate: perturbation error, due to the Laplace noise, and approximation error, due to averaging. Equation (3.5) quantifies the total error introduced into an arbitrary utility estimate.

\[ \text{Err}[\hat{\mu}_u^i] = AE_u^i + \sum_{c \in \Phi_S} \sum_{v \in \text{sim}(u) \cap c} \text{sim}(u, v) \cdot \text{Err} \left[ \text{Lap} \left( \frac{1}{|c|\epsilon} \right) \right] \]

(3.5)

The \( AE_u^i \) term refers to the approximation error and will be discussed shortly. The right-hand side of (3.5) characterizes the perturbation error due to the Laplace noise. Observe that when each user is in its own singleton cluster, the error will be the same as that of NOE; but, as the cluster size increases, the expected perturbation error quickly disappears. Intuitively, the bigger a cluster gets, the less sensitive its average weight is to any one preference edge, allowing a smaller amount of noise to provide the same level of privacy.

The reduction in noise does not come for free, however. As equation (3.6) shows, the price
that we pay comes in the form of approximation error, $AE_u^i$, due to averaging:

$$AE_u^i = \mu_u^i - \sum_{c \in \Phi_{S_U}} \sum_{v \in \text{sim}(u) \cap c} \text{sim}(u,v) \times \bar{c}$$

$$= \sum_{v \in \text{sim}(u)} \text{sim}(u,v) \times w(v,i) - \sum_{c \in \Phi_{S_U}} \sum_{v \in \text{sim}(u) \cap c} \text{sim}(u,v) \times \bar{c}$$

$$= \sum_{c \in \Phi_{S_U}} \sum_{v \in \text{sim}(u) \cap c} \text{sim}(u,v) \times (w(v,i) - \bar{c})$$

(3.6)

From the equation above, it is obvious that simply assigning edges to clusters at random will lead to a significant amount of approximation error; however, by using a more intelligent clustering strategy, we can exchange a large amount of perturbation error for a relatively small amount of approximation error.

**Devising a Clustering Strategy.** An immediate observation from equation (3.6) is that approximation error occurs when the weight of a preference edge differs significantly from the average weight of its cluster. Based on this observation, we might attempt to design a clustering strategy that minimizes approximation error by clustering such that the edges in each cluster have uniform weight. For example, consider a hypothetical strategy $S_U$ that optimally clusters the preference edges into two clusters $c_0, c_1$ where $c_0 = \{(u,i)|w(u,i) = 0\}$, and $c_1 = \{(v,i)|w(v,i) = 1\}$. Then, the approximation error becomes:

$$AE_u^i = \sum_{c \in \Phi_{S_U}, v \in \text{sim}(u) \cap c} \text{sim}(u,v) \times (w(v,i) - \bar{c})$$

$$= \sum_{c \in \Phi_{S_U}, v \in \text{sim}(u) \cap c} \text{sim}(u,v) \times 0$$

$$= 0.$$

However, unlike the random clustering strategy that we have considered up to now, this strategy requires looking at the private edge weights to determine the proper cluster assignment for each edge. Intuitively, for such a strategy to preserve privacy, the probability of any edge being
assigned to \( c_0 \) would have to be close to its probability of being assigned to \( c_1 \), which suggests that many edges must be mis-clustered, leading to poor results.

Instead of clustering based on the edge weights, we propose a clustering strategy that uses *only the public social graph* as input, and importantly, does not require the introduction of additional noise. The key observation, based on equation (3.6), is that low approximation error can be obtained for an arbitrary utility estimate \( \hat{\mu}_u^i \), without the need to look at the sensitive edge weights, by clustering such that the edges belonging to the users that are most similar to \( u \) are assigned to the same cluster and constitute a large fraction of the edges in that cluster\(^4\).

The following simple example illustrates. Suppose that there is a cluster \( c = \{(v, i) | v \in \text{sim}(u)\} \) and assume for simplicity that \( \text{sim}(u, v) = x, \forall v \in \text{sim}(u) \); then, rewriting equation (3.6), we can see that the approximation error cancels out:

\[
AE_u^i = \sum_{c \in \Phi} \sum_{v \in \text{sim}(u) \cap c} \text{sim}(u, v) \times (w(v, i) - \bar{c})
\]

\[
= \sum_{v \in \text{sim}(u)} x \times (w(v, i) - \bar{c})
\]

\[
= x \times \left( \sum_{v \in \text{sim}(u)} w(v, i) - \sum_{v \in \text{sim}(u)} \bar{c} \right)
\]

\[
= x \times \left( \sum_{v \in \text{sim}(u)} w(v, i) - \sum_{v \in \text{sim}(u)} \frac{1}{\sum_{v \in \text{sim}(u)} w(v, i)} \right)
\]

\[
= 0.
\]

Of course, in real social graphs, any given user will usually belong to many different similarity sets, so it will not be possible to produce clusters that are both disjoint and that perfectly correspond to the similarity sets of all users, as the example above assumes. However, the observation above suggests that approximation error can be significantly reduced by building clusters around groups of edges belonging to users that have *high mutual similarity to many of*

---

\(^4\)Recall that computing the similarity of two users, \( \text{sim}(u, v) \) uses only the public social graph; thus, no privacy is leaked by this clustering strategy.
the same users.

Remark. Before proceeding with our discussion, we point out that since we are now talking about clustering based only on the similarity sets—without any regard to the preference edge weights—we can view this conceptually as producing a single clustering of the users, from which the cluster assignments for the preference edges, with respect to any item \( j \), can be derived. That is, regardless of which item we are computing utility estimates for, the cluster structure is the same. To clarify further, this implies that two preference edges \((u, i)\) and \((v, i)\) are in the same cluster if and only if edges \((u, j)\) and \((v, j)\) are in the same cluster (for any \( u, v \in U \) and \( i, j \in I \)).

Clustering via Community Structure. For the social similarity measures discussed in Section 3.2.2 (and other natural choices), we make the intuitive observation that groups of users with high mutual similarity tend to be highly interconnected via many short paths in the social graph. In the graph analysis literature, such regions of highly interconnected nodes are called communities, and they are an inherent and well-studied characteristic of real social graphs [37]. In light of this connection, we investigated community detection as a clustering strategy for reducing approximation error.

For this work, we adopted the popular Louvain method [11], due to its demonstrated ability to quickly and accurately detect communities in very large graphs [37]. Louvain is a fast, heuristic approach that involves the greedy maximization of modularity to detect communities. Modularity is a measure of the density of edges within clusters relative to that of the same clusters in a hypothetical null model—a graph with the same nodes and degree distribution but with the edges randomly re-wired. The modularity of a clustering \( \Phi \), denoted \( Q(\Phi) \), is defined as

\[
Q(\Phi) = \sum_{c \in \Phi} \frac{|E_c|}{2|E_s|} - \left( \frac{\sum_{u \in c} \deg(u)}{2|E_s|} \right)^2
\]

where \( E_c \) denotes the set of edges in cluster \( c \), and \( \deg(u) \) is the degree of user \( u \). Louvain begins by placing each user node in its own cluster. It then considers each node of the graph
in sequence, moving it into one of its neighbors’ clusters, if doing so would lead to a gain in modularity. This process is repeated until convergence, at which time the algorithm contracts each cluster into a single super-node and repeats the process on the new graph. These two steps are repeated until the modularity is maximized. We additionally incorporate a multi-level refinement step (see [108]) that helps to make the clustering output more stable under different initial node orderings. Louvain has a running time that is linear in the number of edges, allowing it to scale to graphs with billions of edges [37].

Remark. One may question why we did not choose to cluster the users by applying a matrix clustering algorithm, such as K-Means, directly to the user similarity matrix. We point out that, unlike community detection, matrix clustering algorithms typically require the number of clusters to be specified a priori, which can be difficult to determine in practice\(^5\). Moreover, scalability would likely be an issue for such approaches, given the enormous size of modern social graphs.

3.4.2 Algorithm and Analysis

Algorithm 1 presents the end-to-end algorithm for our framework. The algorithm takes as input the two graphs, the number of recommendations \(N\), and the privacy parameter, \(\epsilon\). It returns the set \(R\) containing, for each user \(u \in U\), a personalized list \(R_u\) of the top \(N\) item recommendations. In lines 1–7, the set of users is clustered into disjoint clusters and noisy averages of the edge weights are computed for each cluster, item pair, i.e., equation (3.3). Lines 8–17 compute the noisy utility values for every user-item pair, following equation (3.4). Finally, in lines 18–20, a personalized recommendation list is produced for each user from the noisy item utilities. We reiterate that createClusters\((G_s)\) technically clusters the users (not the preference edges) into disjoint clusters; thus, size\(\(c)\) in the algorithm refers to the number of users in cluster \(c\). The same clustering of the users is used for computing the noisy utility estimates for all items (i.e.,

\(^5\)Note that we cannot simply try different values of \(k\) and pick the one that yields the most accurate utility estimates with equation (3.4), as comparing the estimates against the true utilities would violate differential privacy.
Algorithm 1 Privacy-Preserving Social Recommender

Input: $G_s, G_p, N, \epsilon$

Output: The set $R$ containing a recommendation lists $R_u$ for each user with the top $N$ items ranked by utility.

1: $\Phi \leftarrow \text{createClusters}(G_s)$
2: for item $i$ in $I$ do
3: for cluster $c$ in $\Phi$ do
4: $w^i_c \leftarrow 0$
5: for user $u$ in cluster $c$ do
6: $w^i_c \leftarrow w^i_c + w(u, i)$
7: $\hat{w}^i_c \leftarrow \frac{w^i_c}{\text{size}(c)} + \text{Lap}\left(\frac{1}{\text{size}(c)\epsilon}\right)$
8: for user $u$ in $U$ do
9: Initialize $R_u \leftarrow \emptyset$
10: for item $i$ in $I$ do
11: $\hat{\mu}_u^i \leftarrow 0$
12: for cluster $c$ in $\Phi$ do
13: Initialize $\text{sim.sum} \leftarrow 0$
14: for user $v$ in $c \cap \text{sim}(u)$ do
15: $\text{sim.sum} \leftarrow \text{sim.sum} + \text{sim}(u, v)$
16: $\hat{\mu}_u^i \leftarrow \hat{\mu}_u^i + \text{sim.sum} \times \hat{w}^i_c$
17: Append the tuple $(i, \hat{\mu}_u^i)$ to the list $R_u$
18: Sort $R_u$ in descending order of utility
19: Truncate $R_u$ to the top $N$ items
20: Add $R_u$ to $R$
21: return $R$
for every iteration of the loop beginning on line 2).

The framework can be easily customized by plugging in different social similarity measures for \( \text{sim}(u, v) \) in line 15 or alternate different clustering algorithms on line 1. As long as both the similarity measure and the clustering algorithm operate solely on the social graph, \( G_s \), then \( \epsilon \)-differential privacy is satisfied, as we will prove next.

### 3.4.2.1 Privacy Analysis

**Theorem 3.4.1.** Algorithm 1 satisfies \( \epsilon \)-differential privacy for preference edges.

**Proof.** Consider a social graph \( G_s = (U, E_s) \) and two arbitrary preference graphs, \( G_p = (U, I, E_p) \) and \( G'_p = (U, I, E'_p) \), such that \( E'_p \) is obtained from \( E_p \) by adding or removing a single preference edge. We can view Algorithm 1 as consisting of three modules:

1. \( \text{createClusters}(G_s) \), on line 1, clusters the user nodes into a set of disjoint clusters, \( \Phi \), based on the social graph.

2. \( A_w(\Phi, \epsilon, G_p) \) is the module consisting of lines 2–7, which takes the clustering \( \Phi \) and the preference graph \( G_p \), and outputs \( \hat{w} = \{\hat{w}^i_c\mid i \in I; c \in \Phi\} \), containing the noisy averages for each cluster-item pair.

3. \( A_R(G_s, \Phi, N, \hat{w}) \), consisting of lines 8–21, uses the noisy averages \( \hat{w} \) output by \( A_w \) to compute the utility estimates for every user-item pair and outputs \( R \) containing the top \( N \) items \( R_u \) for every user \( u \in U \).

Observe that only module \( A_w \) makes use of the private preference data. The clusters output by \( \text{createClusters} \) are derived solely from the public social graph, and module \( A_R \) uses only public data and the sanitized output, \( \hat{w} \), from \( A_w \). Therefore, to show that Algorithm 1 is \( \epsilon \)-differentially private, it suffices to show that module \( A_w \) satisfies definition 2.1.2; that is, we need to show that for any set of possible outputs \( O \subseteq \text{Range}(A_w) \),

\[
Pr[A_w(\Phi, \epsilon, G_p) \in O)] \leq e^\epsilon Pr[A_w(\Phi, \epsilon, G'_p) \in O)].
\]
Observe that the outer loop of module $A_w$ (beginning on line 2) is run one time for every item. Let us consider an arbitrary iteration $j$ (that is, the iteration for item $j$). Ignoring the Laplace noise added on line 7, for the moment, we can view the output of iteration $j$ as the vector $w_j = \langle w^j_c \mid c \in \Phi \rangle$, which contains the item weight averages (for item $j$) for each cluster in $\Phi$. Observe that adding or removing a single arbitrary preference edge $(v,j)$ to/from $G_p$ could change $w^j_c$ by at most $\pm 1/|c|$, where $c$ is the cluster containing user $v$.\footnote{Recall that the clusters contain users, so adding or removing a preference edge does not change $|c|$.} We can view the process of computing the noisy averages $w^j$ as performing a sequence of disjoint computations, since each average uses a different subset of the preference edges (e.g., $w^j_c$ is computed only from the subset of edges $\{(v,j) \mid v \in c \} \subset G_p$). In line 7, we add independent Laplace noise from $Lap(\frac{1}{\epsilon|c|})$ to each of the averages, which satisfies $\epsilon$-differential privacy for each, by Theorem 2.1.1. By Theorem 2.1.4, computing the sequence of averages in $w^j$ satisfies $\epsilon$-differential privacy, and therefore iteration $j$ as a whole satisfies $\epsilon$-differential privacy. Finally, since each iteration of $A_w$ operates on a disjoint set of the preference edges (i.e., those attached to a single item), the parallel composition property (Theorem 2.1.4) applies again, and module $A_w$ as a whole satisfies $\epsilon$-differential privacy. Therefore, we have

$$Pr[A_w(\Phi, \epsilon, G_p) = \hat{w}] \leq e^\epsilon Pr[A_w(\Phi, \epsilon, G_p') = \hat{w}]].$$

which holds for every $\hat{w} \in \text{Range}(A_w)$; thus, we have

$$Pr[A_w(\Phi, \epsilon, G_p) \in O] = \sum_{\hat{w} \in O} Pr[A_w(\Phi, \epsilon, G_p) = \hat{w}] \leq \sum_{\hat{w} \in O} e^\epsilon Pr[A_w(\Phi, \epsilon, G_p') = \hat{w}]$$

$$= e^\epsilon Pr[A_w(\Phi, \epsilon, G_p) \in O],$$

as desired. This completes the proof. \hfill \Box
3.5 Experimental Study

Due to the data-dependent nature of our proposed approach, we conducted a series of experiments designed to evaluate the trade-off between privacy and accuracy for different instantiations of our framework on two real-world data sets. In particular, we sought to validate our hypothesis that clustering according to the community structure of the social graph would produce clusters that effectively balance approximation error and perturbation error. We also sought to determine how the number of recommendations, \( N \), and the privacy setting, \( \epsilon \), impacts the accuracy of the recommendations. Additionally, we compared our approach against the two naïve baseline approaches, NOU and NOE (Section 3.4.1.1), and against adaptations of two recent differential privacy approaches (GS [61] and LRM [135]) (Section 3.3). The next section discusses the two data sets used in our experiments and Section 3.5.2 outlines our general methodology. The results of our experiments are presented in Sections 3.5.3 and 3.5.4, followed by a discussion of the results and our conclusions in Section 3.6.

3.5.1 Datasets

We used two publicly available data sets that were crawled from two online, entertainment-oriented communities: (1) Last.fm\(^7\), a social music service, and (2) Flixster\(^8\), a social movie rating web site.

3.5.1.1 Last.fm data set [15]

This relatively small data set contains undirected social edges between user nodes that indicate mutual friendship, and directed edges from user nodes to artist nodes expressing listened-to relations where each edge is weighted by the number of times the user listened to a song by the artist. The social graph consists of one main connected component containing 97.4% of the user nodes and 19 small connected components with 2 to 7 nodes each. We constructed a preference

\(^7\)http://ir.ii.uam.es/hetrec2011/datasets.html
\(^8\)http://www.sfu.ca/~sja25/datasets
graph $G_p$ by discarding 636 listened-to edges ($< 0.07\%$ of the edges in $G_p$) with weight $< 2$, and assigning a weight of 1 to the remaining edges.\(^9\) Table 3.1 summarizes the pre-processed data set.

### 3.5.1.2 Flixster data set [52]

This much larger data set contained more than 700K users and almost 49K movies. We chose to use the main connected component of the social graph induced by the set of users with at least one preference edge (movie rating). The resulting data set contained around 137 thousand users and almost 49K items (movies). Each preference edge $(u, i)$ had a weight in the range $[0.5, 5]$, indicating the rating given to movie $i$ by user $u$. To get rid of ratings that were likely to indicate dislike, we discarded edges having weight (rating) $< 2$ (approximately 6.6\% of the edges) and assigned a weight of 1 to the remaining edges.

Table 3.1 Summary of data sets.

<table>
<thead>
<tr>
<th></th>
<th>Last.fm</th>
<th>Flixster</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>U</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>E_s</td>
<td>$</td>
</tr>
<tr>
<td>avg. user degree</td>
<td>13.4 (std. 17.3)</td>
<td>18.5 (std. 31.1)</td>
</tr>
<tr>
<td>$</td>
<td>I</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>E_p</td>
<td>$</td>
</tr>
<tr>
<td>avg. item degree</td>
<td>48.7 (std. 6.9)</td>
<td>54.8 (std. 218.2)</td>
</tr>
<tr>
<td>$sparsity(G_p)$</td>
<td>0.997</td>
<td>0.999</td>
</tr>
</tbody>
</table>

### 3.5.2 Experimental Setup

We instantiated our framework with each of the four social similarity measures introduced in Section 3.2.2, yielding four private social recommenders, denoted AA, CN, GD\(^{10}\) and KZ\(^{11}\), respectively. We used $NDCG$ (Section 3.2.5) as our accuracy metric. The $NDCG$ values reported

---

\(^9\)Listening to an artist only once is unlikely to indicate a positive preference.

\(^{10}\)For GD, we limited the maximum distance $d = 2$.

\(^{11}\)For KZ, we limited the maximum path length $k = 3$ and fixed the damping factor $\alpha = 0.05$. 
for a given data set are averages over the users in the data set. Experiments were repeated 10 times. For the Flixster data set, we generated recommendations for a random subset of 10,000 users; however, we emphasize that we used all 137,372 users in the clustering phase and for computing the recommendations of the randomly selected users. To cluster the social graphs, we ran the Louvain algorithm 10 times (with different random node orderings each time) and selected the clustering with the highest modularity; we emphasize that this had no impact on the privacy guarantee, since only the public social graph was used in the clustering procedure. For the Flixster graph, clustering took an average of only 26 seconds per run; it took less than a second per run on Last.fm. For Last.fm, 35 clusters were produced. The main connected component of the graph was divided into 16 clusters containing an average of 115 (std. 164) users each, while the remaining 19 clusters corresponded directly to the 19 small components of the social graph. The largest cluster contained 28.5% of the users. For Flixster, 46 clusters were produced with an average size of 2,986 users (std. 6,399 users). The largest cluster contained 18.3% of the users. We used the Louvain implementation with multi-level refinement from Pajek v3.07\(^\text{12}\). All experiments were performed on a single machine with an Intel Core i7 (2.8Ghz) and 12G of RAM.

![Graphs showing NDCG performance](image-url)

(a) Avg. \(NDCG@10\)  
(b) Avg. \(NDCG@50\)  
(c) Avg. \(NDCG@100\)

Figure 3.1 (Last.fm) Comparison of average \(NDCG@N\) of each private social recommender for different settings of \(\epsilon\) and \(N\) on Last.fm. Note that the privacy guarantee varies from weak to strong as you move left to right along the x-axis.

\(^{12}\text{http://pajek.imfm.si}\)
3.5.3 Analysis of Privacy-Accuracy Trade-off

We evaluated our framework using a variety of privacy settings ($\epsilon \in \{\infty, 1.0, 0.6, 0.1, 0.05, 0.01\}$) and numbers of recommendations ($N \in \{10, 50, 100\}$) to investigate their effect on the recommendation accuracy ($NDCG@N$). The results for Last.fm and Flixster are shown in Figs. 3.1 and 3.2, respectively. Recall that a smaller $\epsilon$ leads to stronger privacy, and hence a greater amount of noise. We included $\epsilon = \infty$ (i.e., no noise) to see how much of the accuracy loss was due to the approximation error alone. For perspective we also include results for the two naïve baseline approaches, NOU and NOE (from Section 3.4.1.1), on Last.fm with $\epsilon \in \{0.1, 1.0\}$ and $N = 50$ (Figs. 3.4(a) and 3.4(b)). The baseline results for Flixster were comparable and thus omitted for space.

3.5.3.1 Effect of Approximation Error

We start by analyzing the impact of the approximation error alone (i.e., $\epsilon = \infty$). We will focus our discussion on NDCG@50 (Figs. 3.1b and 3.2b), as similar observations can be made for the other values of $N$. Looking at the left-most points ($\epsilon = \infty$) of each line in Fig. 3.1b, we can see that for Last.fm, the approximation error accounted for a loss of between 0.13 and 0.19 in
average accuracy, depending on the recommender. On Flixster (Fig. 3.2b), the accuracy loss due to approximation error was $< 0.1$ for all four recommenders. A closer look at the results from the user-level also revealed a strong relationship between user degree (in the social graph) and accuracy. On average, users with more neighbors in the social graph were less affected by approximation error than those with few neighbors. For example, Figs. 3.3a and 3.3b illustrate this effect, as observed for the CN similarity measure on Last.fm and Flixster, respectively. Each point on the scatter plots represents the NDCG@50 for one user (averaged over the 10 runs) as a function of the user’s degree (log scale). For users with degree $> 10$ in the Last.fm social graph, the average NDCG@50 was 0.969, while for users with degree $\leq 10$ the average was considerably lower at 0.809. For the Flixster social graph, users with degree $> 10$ had an average NDCG@50 of 0.975, while for users with degree $\leq 10$ the average was 0.871. The explanation for this effect is that low degree users tend to have much smaller similarity sets that make up only a small fraction of the clusters that contain them; as a result, their utility estimates are heavily affected by the other non-similar users in the clusters. However, not all low-degree users experienced excessive amounts of approximation error; we observed several common exceptions. Some low-
degree users with high-degree neighbors were able to accumulate large enough similarity sets. Other low-degree users happened to have similarity sets that, despite being small, were attached to universally popular items, which are inherently more resistant to approximation error.

Figure 3.4 Figs. 3.4a and 3.4b show NDCG@50 of the naïve NOU and NOE baselines, as well as adaptations of the recent LRM [135] and GS [61] approaches, on Last.fm with $\epsilon \in \{1.0, 0.1\}$.

### 3.5.3.2 Effect of Privacy Parameter ($\epsilon$)

Let us now look at the results when noise is added into the mix to satisfy differential privacy. We will again focus our discussion on NDCG@50 (Figs. 3.1b and 3.2b). For Last.fm, the noise introduced by privacy settings $\epsilon = 1$ and $\epsilon = 0.6$ had very little effect on the average NDCG@50 compared to $\epsilon = \infty$. As $\epsilon$ became closer to 0.1, the Laplace noise had a more significant impact, reducing accuracy to between 0.7 and 0.73, depending on the recommender. Privacy settings below 0.1 led to poor accuracy in general. Comparing these results to that of the baseline approaches in Fig. 3.4, the benefits of our user clustering approach are substantial. Both baselines performed very poorly, even under weak privacy settings (Fig. 3.4a). For NOU in particular, the recommendations were essentially no better than random guessing. NOE performed much better than NOU under low noise ($\epsilon = 1.0$), but its accuracy was equally as
poor as NOE with $\epsilon = 0.1$. Note that the results for LRM and GS, which are also included in Fig. 3.4, will be discussed below, in Section 3.5.4.

On the Flixster data set (Fig. 3.2), all four instantiations of our framework were remarkably resistant to the Laplace noise, even under the strongest privacy settings. Note the different y-axis values used in the Flixster plots compared to the Last.fm plots. For $\epsilon \geq 0.05$, the noise had little to no impact on average accuracy. With $\epsilon < 0.05$, the accuracy began to fall sharply; however, even with $\epsilon$ as low as 0.01 (which is considered to be a very strong privacy guarantee), the accuracy was $\geq 0.79$ on average, for all four recommenders. The higher accuracy observed for the Flixster data set, compared with Last.fm, appears to be due primarily to the larger average user degree in the Flixster social graph (see Table 3.1), which led to much larger sets of mutually similar users on average, and in-turn much larger clusters. This enabled a more significant reduction in perturbation error without significantly increasing approximation error.

### 3.5.3.3 Effect of the Number of Recommendations (N)

For Last.fm, we observed that the average $NDCG$ generally decreased as $N$ increased from 10 to 100 (comparing Figs. 3.1a to 3.1c), though the effect was most evident for small values of $\epsilon$. For example, with $\epsilon = 0.1$, $NDCG@100$ dropped by $\approx 0.1$ for all of the recommenders, $\approx 0.11$ for AA, CN and GD, and by $\approx 0.1$ for KZ, relative to $NDCG@10$ for the same $\epsilon$; for $\epsilon = 0.6$, on the other hand, $NDCG@100$ only dropped by $\leq 0.03$ for all recommenders, for KZ and by $\approx 0.02$ for the others, compared to $NDCG@10$. This effect can be explained by the fact that the items that are farther down in the ranking have a lower utility and are thus more affected by noise; as $N$ gets larger, there is a greater opportunity for such items to be displaced by the many zero-utility items. On Flixster, increasing $N$ up to 100 only had a negative effect for the strongest privacy setting $\epsilon = 0.01$, where $NDCG@100$ dropped by less than 0.035 for all recommenders, relative to $NDCG@10$. For larger values of $\epsilon$, increasing the number of recommendations actually had a positive impact. This appears to be a product of the higher user degree and the higher average item degree in the Flixster data set, which led to
a greater number of high-utility items.

3.5.4 Comparison with Other Approaches

In this set of experiments, we compared our approach with two recently proposed differential privacy approaches: the low-rank mechanism (LRM) of [135], and an adaptation of the group and smooth (GS) approach of [61]. Although neither of these approaches was designed specifically for the private social recommendation task, they appear to be the closest competitors to our approach in the current literature. We first briefly explain how we adapted these approaches to our setting, and then present the results.

3.5.4.1 Low-rank Mechanism

LRM can be applied quite naturally to our task, as follows. LRM considers an $m \times n$ workload matrix $W$ with rows corresponding to linear queries, and a data set represented as an $n \times 1$ column vector, $D$. Thus, the product $WD$ is a vector containing the answers to the queries. Let $W$ denote a $|U| \times |U|$ workload matrix of utility queries, where $W_{u,v} = \text{sim}(u, v)$. Let $D_i$ be a length $|U|$ column vector containing a 1 at position $v$ when $(v, i) \in G_p$, or a zero otherwise. Note that $W$ is the same for all $D_i$, since the similarity scores only depend on the social graph. LRM decomposes $W$ into matrices $B$, of size $|U| \times r$, and $L$ of size $r \times |U|$, such that $r < |U|$ and $W \approx BL$. It then adds Laplace noise to $LD_i$, yielding $\hat{LD}_i$. Noisy answers to the utility queries for item $i$ are then obtained via the product $B\hat{LD}_i$. The key is that $LD_i$ has lower sensitivity than $WD_i$, requiring less noise. We used the Matlab LRM implementation published by the authors of [135]. LRM takes two parameters: (1) $r$ determines the rank of $BL$ and it suggested in [135] to choose $r$ between $\text{rank}(W)$ and $1.2 \cdot \text{rank}(W)$; we used $\text{rank}(W)$ in our experiments. (2) $\gamma$ determines the error tolerance for the decomposition of $W$; we used $\gamma = 10$ ($\gamma = 1$ for KZ, since it produces significantly smaller utility values), as suggested in [135].
3.5.4.2 Group and Smooth

Although the setting and task in [61] is different from ours, the idea behind the approach can be extended to our problem. GS can be thought of as extending NOU similarly to the way our approach extends NOE; however, GS relies on uniformity to reduce approximation error. The idea is to compute answers to all of the utility queries (non-privately), and then to arrange them in groups of size $m$, such that the values in each group are approximately uniform (using a separate, differentially private procedure discussed below); the values in each group can then be approximated by the noisy group average. The noise added to each group average is $Lap \left( \frac{\Delta}{\epsilon^2} \right) = Lap \left( \frac{2\Delta}{\epsilon} \right)$, where $\Delta = \frac{1}{m} \times \max_{u \in U} \sum_{v \in U} sim(v, u)$. The challenge is how to group the query answers so that they are approximately uniform, without leaking privacy. [61] proposes a sampling approach that amounts to computing rough estimates $\hat{\mu}_u^i$ for the query answers by ensuring that each preference edge $(v, i)$ is used in at most one query estimate, selected at random from $\{\hat{\mu}_u^i | u \in sim(v)\}$; then noise is added to each utility estimate $\hat{\mu}_u^i$ with $\frac{\epsilon}{2}$ and $\Delta = \max_{v \in sim(u)} sim(u, v)$. Finally, the true answers to the utility queries are sorted, using the rough estimates as the sorting keys, and then grouped consecutively in groups of size $m$. In [61], the $m$ that gave the lowest error relative to the rough utility estimates (not the true utilities) was selected; however, for simplicity we used the $m$ that gave the best NDCG relative to the true utilities.\footnote{This technically violates differential privacy and gives GS an unfair advantage that it would not have in practice.} The reader is referred to [61] for more details on GS and proof that it satisfies $\epsilon$-differential privacy.

3.5.4.3 Comparison Results

Fig. 3.4 shows the NDCG@50 of LRM and GS on Last.fm, along side the results for NOE and NOU. Although both significantly outperformed the NOU baseline, neither approach was able to achieve a reasonable level of accuracy on this data set, even under weak privacy ($\epsilon = 1.0$). In fact, both approaches were outperformed by the NOE baseline. The poor performance of
GS appeared to be due to the ineffectiveness of the sampling procedure on the highly sparse preference data, which led to a poor groupings with non-uniform values. For LRM, we observed that all of the workload matrices had high rank—the average rank was 1808, which is close to the maximum rank of 1892 for workloads of this size—and LRM is known to perform poorly on workloads with high rank [135]. The high rank of the workload matrices in this setting seems to be consistent with the general findings of [122] that sparse symmetric matrices typically have high rank. We also note that the number of utility queries, even for this small data set, was larger than the largest workload considered in [135] (1892 queries vs. 1024). Our results are consistent with the findings in [135], where LRM was shown to perform worse as the number of queries increases. Given that the number of queries in our task is proportional to the number of users, we would expect the results to be even worse in practice, where the number of users is often in the millions. The poor results of LRM and GS illustrate the futility of NOU-based approaches for the social recommendation task.

In this experiment, we used only the default parameters for LRM. It is possible that using a lower value for $r$ or a higher value for $\gamma$ (both of which would lead to more approximation error but potentially lower sensitivity) could yield better results in this setting; however, since the right choice for these settings depends on the data, it is unclear how to tune them without violating differential privacy.

### 3.6 Summary

Our results on the Last.fm and Flixster data sets demonstrate the ability of the proposed framework to achieve strong differential privacy guarantees without significantly degrading the utility of the recommendations produced. These findings confirm our hypothesis that clustering according to the natural community structure of the social graph leads to clusters that strike an effective balance between approximation error and perturbation error for several natural social similarity measures. We observed especially good results for our approach on the larger data
set (Flixster), suggesting that our approach is likely to perform better for larger, more active communities; however, reasonable results were also observed for the smaller Last.fm data set suggesting that our approach is suitable for applications with smaller social communities as well. There are a number of avenues for extending this work. A main focus for future work will be extending our framework to provide differential privacy guarantees when recommendations are made over dynamic graphs. We also plan to explore improvements to the clustering strategy, including (1) optimizing it more for the specific similarity measure being used, and (2) investigating post-processing heuristics to clean up the clustering by, for example, pruning low-quality clusters. We also plan to extend our framework to handle weighted preference edges (e.g., ratings) and evaluate the impact of different weighting schemes. Finally, we would like to evaluate the framework for a larger variety of social similarity measures and on additional data sets as they become available.
Chapter 4

Personalized Differential Privacy

4.1 Motivation

Differential privacy [26, 31] is a powerful framework for providing strong, formal privacy guarantees for aggregate data analysis. Differential privacy ensures that no individual user can significantly affect the output of an aggregate computation; consequently, an adversary observing the output is unable to determine, with high probability, whether a particular user’s data was present in the input. A common approach for achieving differential privacy is to inject random noise that is carefully calibrated according to the sensitivity of the computation (i.e., the maximum impact that any one user can have on the output), and a global privacy parameter.

In this work we propose a novel privacy definition to address an important limitation of differential privacy—that it provides only a uniform level of privacy protection for all users in a dataset. This “one size fits all” approach ignores the reality that data privacy is a personal and multifaceted concept, and that different individuals may have very different expectations for the privacy of their personal data. Indeed, several studies in the psychology literature have observed that individuals typically fall into several distinct groups or clusters based on their privacy attitudes [6, 2, 1]. In particular, Berendt et al. [6] conducted a large-scale study of attitudes to privacy in e-commerce. They were able to distinguish a clear group of “privacy
fundamentalists” and a distinct group of individuals with only marginal concern for privacy. The remaining respondents exhibited privacy attitudes in between the two extremes and differed on the focus of the privacy concern. Similar clusterings of users based on privacy attitudes have been observed by other researchers [2, 1], and have identified other factors which contribute, such as as cultural values [89], and income, age and political views [90].

In practice, when faced with a dataset comprising multiple users with different privacy expectations, a data analyst employing differential privacy has limited options. One option is to set the global privacy level high enough to satisfy even the privacy fundamentalists in the dataset. This is likely to introduce an unacceptable amount of noise into the analysis outputs, resulting in poor utility. On the other hand, setting a lower privacy level may force the analyst to exclude a significant portion of the dataset from analysis (i.e., the data belonging to the fundamentalists), which may also significantly harm utility. In contrast to traditional differential privacy, the personalized privacy model we introduce allows privacy requirements to be specified at the user-level. Within the proposed model, we present mechanisms that are able to take these individual privacy requirements into account to guarantee precisely the required level of privacy to each user, while in many cases providing significantly better utility than the naïve options discussed above.

As well as the possibility of improved data utility, allowing users a degree of control over the disclosure of their data offers other important benefits. There is evidence that when users are given control over their privacy they are more inclined to want to contribute their data for analysis in the first place, and to do so truthfully [117]. Indeed, the importance of control in the context of privacy is emphasized in a widely accepted definition of privacy from the psychology literature, due to Westin [127], as “the ability of the individual to control the terms under which personal information is acquired and used”.

55
4.1.1 Contributions

In this paper we consider the setting in which a trusted data analyst desires to publish aggregate statistics computed from a dataset comprising personal data of many individual users. Every user may potentially have a different privacy requirement for his or her data and the analyst would like to publish useful aggregate information about the data, while simultaneously complying with the individual privacy requirements of the contributors. To that end, we propose a new privacy framework called Personalized Differential Privacy (PDP), a generalization of differential privacy in which the privacy requirements are specified at the user-level, rather than by a single, global privacy parameter. The privacy guarantees of our framework have the same strength and attack resistance as differential privacy, but are personalized to the preferences of all users in the input domain. We also show that the composition properties of differential privacy carry over to PDP, allowing complex privacy-preserving algorithms to be constructed from individual PDP components.

We introduce two novel mechanisms for achieving PDP. Our main goal is to design mechanisms that can take advantage of the non-uniform privacy requirements to attain better utility than could be achieved with differential privacy. Our first mechanism is general and can be used to easily and automatically convert any existing differentially private algorithm into a PDP algorithm. The mechanism is a two-step procedure that involves a non-uniform sampling step at the individual tuple level, followed by the invocation of an appropriate differentially private mechanism on the sampled dataset. In the sampling step, the inclusion probabilities for each tuple are calculated according to the individual privacy requirements of the corresponding user. We show that the two sources of randomness introduced by this two-step procedure combine to yield the precise personalized guarantee demanded. Our second mechanism is a more direct approach to achieve PDP, inspired by the well-known exponential mechanism of McSherry and Talwar [88]. The mechanism is applicable to common aggregates such as counts, medians, and min/max and can be shown to outperform the sampling-based mechanism in certain scenarios.
In particular, we find that it is generally preferable to the sampling-based mechanism for \textit{counts}, which are especially sensitive to the effects of sampling.

We conducted an extensive experimental study of several instantiations of the PDP framework, on both synthetic and real datasets. In particular, we studied both of our mechanisms for the important \textit{count} and \textit{median} functions. We then investigated the application of our sampling-based mechanism to the more complex task of multiple linear regression. Our results demonstrate the broad applicability of our framework and the utility gains that are possible by taking personal privacy preferences into account.

In the next section, we begin by reviewing related work. In Section 4.3 we introduce our new privacy definition, followed by a discussion of how to satisfy the definition for arbitrary tasks in Section 4.4. Section 4.5 presents our experimental study and Section 4.6 concludes the chapter.

4.2 Related Work

A line of work, started by Xiao and Tao [120], introduced personalized privacy for k-anonymity. K-anonymity requires that every record in a dataset be indistinguishable from at least $k - 1$ others, in terms of their identifying attributes [118]. This is often accomplished through the generalization or suppression of attribute values. In Xiao and Tao’s approach, users specify their privacy preferences by indicating a minimum level of generalization for their data in the generalization hierarchy. Subsequently a slew of related approaches (e.g., [134, 123, 114, 124]) extended this to other methods for achieving $k$-anonymity and related definitions. However, these definitions have been criticized due to the feasibility of attacks that can lead to disclosure of sensitive attributes [81, 72, 136], and more robust notions, led by differential privacy, are now preferred.

Our primary mechanism for achieving personalized differential privacy involves the use of sampling to introduce non-uniform uncertainty at the tuple level. Although this is the first work
to use sampling to realize a personalized notion of privacy, it has previously been combined with
differential privacy for other purposes [60, 73, 76, 61, 39]. Li et al. [73] showed that uniform
random sampling in combination with differential privacy amplifies the privacy guarantee. That
result was motivated by the observation that since random sampling is often already an inherent
part of data collection, one can take advantage of that existing randomness to lower privacy
costs. Prior to this, Kasiviswanathan et al. [60] implicitly used this amplification effect to build
a private PAC learner for parity functions. Gehrke et al. [39] proposed a relaxation of differential
privacy, called crowd-blending privacy which, although strictly weaker than differential privacy
alone, when preceded by a random sampling step also satisfies differential privacy. Aside from
targeting a very different objective, our use of sampling in the present work also differs from all
prior work described above by sampling tuples from the input data independently, with non-
uniform probabilities: the inclusion probability for each tuple depends on the corresponding
individual’s privacy requirement (as well as a global threshold). It is this non-uniformity that
enables the personalized guarantees of PDP.

The basic concept of personalized differential privacy was independently suggested by several
different groups at the same time, ours included. Each of these works differs from ours in
important respects. Most similar to our work is that of Alaggan et al. [4], which proposes
heterogeneous differential privacy, a privacy definition similar in spirit to ours but differs in its
presentation and, more importantly, in the proposed mechanisms for satisfying the definition.
The “stretching” mechanism proposed in that work—which is based on the Laplace mechanism
and works by rescaling the input values according to the corresponding privacy parameters—
applies only to a limited subset of real-valued functions; specifically, it cannot be used for
functions like median, min/max, and many others, that rely on the exponential mechanism.
It is also fundamentally incompatible with some other types of queries, such as counting the
number of non-zero values in a dataset, where rescaling does not alter the answer to the query.
In contrast, our primary mechanism for achieving PDP has no such restrictions; it can be used
to automatically convert any differentially private algorithm—whether it is an instance of the
Laplace mechanism, the exponential mechanism or even a composition of multiple differentially private modules—into one that satisfies our personalized privacy definition. Additionally, unlike [4], we compare our PDP mechanisms to several baselines and demonstrate the advantages of PDP over standard differential privacy with regard to improved utility.

Ebadi et al. [33] also proposed a variant of differential privacy called personalized differential privacy that is defined similarly to ours. However, in contrast to our work, they focused on the interactive setting, where multiple queries are submitted to a database over time and providing personalized privacy amounts to a privacy budget management problem. Specifically, each user/record has a personal privacy budget and the task is to optimize the composition process over time so that, for example, a user’s budget is not wasted by including it in queries where it cannot possibly contribute, or by not penalizing a record’s privacy budget if it was added to the database after a query was answered. By contrast, our mechanisms are more general and are able to ensure different levels of privacy for every record touched by a single aggregate query. We believe that the two approaches are complementary. Similar to the above work, Niknami et al. [97] also studied the personalization of differential privacy from the budget management point of view, but in the context of spatial databases. Specifically, they considered the task of answering range counting queries over spatial databases in which the different subregions of a geometric space may have different privacy requirements. As with the previous work, their approach involves non-uniform allocation of the privacy budget to different subregions (i.e., to the counting queries asked against the different subregions) and then uses standard mechanisms (e.g., Laplace mechanism) and composition rules to provide personalized privacy.

Finally, it is worth mentioning a recent line of work on privacy auctions (surveyed in [99]), that is ostensibly similar to the present work. This line of work is mainly concerned with how to accurately compute statistics over a population of users who demand financial compensation for any privacy loss incurred by their participation. Users specify a (possibly non-uniform) valuation on their privacy that expresses their privacy cost incurred by participating in an \( \epsilon \)-differentially private analysis (as a function of \( \epsilon \)), and hence the amount of compensation due
if their data is used. The analyst’s job is to choose the users from whom to “buy data”, such that the analyst’s financial cost is minimized, while the computed statistic meets some utility goal. In other words, users are not guaranteed a certain level of privacy, but rather that they will be compensated in proportion to their privacy valuation, should their data be used. The privacy auction mechanisms ultimately provide a uniform privacy guarantee. In contrast, our setting allows users to individually specify a specific minimum level of privacy for their data, and the mechanisms that we develop guarantee at least the required privacy levels of all users.

4.3 Personalized Differential Privacy

In this section, we introduce Personalized Differential Privacy (PDP) and discuss its properties. In Section 4.4 we present several mechanisms that satisfy the definition.

4.3.1 Privacy Definition

We model a dataset as a set of tuples from a universe $\mathcal{D}$, with one or more attributes $A_1, \ldots, A_d$. Every tuple in a dataset is assumed to be associated with a user in $\mathcal{U}$, where $\mathcal{U}$ is the universe of users (e.g., all of the customers of an online store, all of the patients of a given hospital, etc.).

**Definition 4.3.1** (Dataset). A dataset $D \subset \mathcal{D}$ is a set of tuples $D = \{t_1, \ldots, t_i, \ldots\}$ from universe $\mathcal{D}$, where $t_i \in A_1 \times \ldots \times A_d \times \mathcal{U}$; the $A_i$ are attributes; and $\mathcal{U}$ denotes the universe of users. We write $t_u$ to denote the user associated with tuple $t$.

The attributes $A_1, \ldots, A_d$ may be numeric or categorical. Note also that a dataset $D$ will not necessarily contain a tuple for every $u \in \mathcal{U}$. Moreover, depending on the semantics of the data, or the analysis task being considered, it may be possible for a dataset to contain multiple tuples for the same user (e.g., all of the products a user has purchased), while in other cases it may not make sense for a dataset to contain more than one tuple per user (e.g., a tuple contains a user’s annual salary).
For both differential privacy and personalized differential privacy, the notion of neighboring datasets is an important one. Recall from Def. 2.1.1 that two datasets are said to be neighboring if one is a proper subset of the other and the larger dataset contains exactly one additional tuple. We write $D \sim D'$ to denote that $D$ and $D'$ are neighbors and that $t \in D'$ and $t \notin D$.

In contrast to traditional differential privacy, in which the privacy guarantee is controlled by a single, global privacy parameter (i.e., $\epsilon$ in Def. 2.1.2), PDP makes use of a privacy specification, in which each user in $U$ independently specifies the privacy requirement for their data. More formally,

**Definition 4.3.2 (Privacy Specification).** A privacy specification is a mapping $\Phi : U \rightarrow \mathbb{R}_+$ from users to personal privacy preferences, where a smaller value represents a stronger privacy preference. The notation $\Phi^u$ is used to denote the privacy preference corresponding to user $u \in U$.

For convenience, we may describe a specific instance of a privacy specification as a set of ordered pairs, e.g., $\Phi := \{(u_1, \epsilon_1), (u_2, \epsilon_2), \ldots\}$ where $u_i \in U$ and $\epsilon_i \in \mathbb{R}_+$. We also assume that a privacy specification contains a privacy preference for every $u \in U$, or that a default privacy level, say $\epsilon_{\text{def}} = 1.0$, is used. As will become clear, the privacy preference values in our model can be interpreted similarly to the $\epsilon$ parameter in traditional $\epsilon$-differential privacy, so we expect privacy preferences to fall in the range $(0.01, 1.0)$. In practice it may be unreasonable to expect typical users to choose a meaningful numerical privacy setting. Rather, we envision a scenario in which a domain expert associates appropriate values with user-friendly descriptors (e.g., low, medium and high privacy) and users choose from those. This represents one possibility; in general, choosing an appropriate privacy parameter for differentially private systems is an open problem and we do not consider it further in this paper.

Our model assumes that the privacy specification is public knowledge. This mirrors the situation in traditional differential privacy, where the global privacy setting $\epsilon$ is assumed to be a public parameter. However, this means that the user’s privacy parameter must not indicate anything about their sensitive values. We believe this to be a reasonable assumption, given that
the privacy specification is defined at the user-level, rather than the tuple level. That is, one can think of the privacy setting as being a function of the user that owns the data rather than a function of the data itself. For example, a politician might have a higher privacy preference for her online browsing history (for instance) than the average user, by virtue of her profession—not because the data itself is inherently any more or less sensitive than that of an average user. In practice, a user might specify their privacy preferences at registration time (e.g., upon joining a service), before any data is generated; then all data that is subsequently produced by that user will use that previously-specified privacy setting. Nevertheless, it should be emphasized that the desired privacy guarantees may not hold in settings where this assumption does not hold. We now formalize our personalized privacy definition.

**Definition 4.3.3 (Personalized Differential Privacy (PDP)).** In the context of a privacy specification \( \Phi \) and a universe of users \( \mathcal{U} \), a randomized mechanism \( \mathcal{M} : \mathcal{D} \rightarrow \mathcal{R} \) satisfies \( \Phi \)-personalized differential privacy (or \( \Phi \)-PDP), if for every pair of neighboring datasets \( D, D' \subset \mathcal{D} \), with \( D \sim D' \), and for all sets \( O \subset \mathcal{R} \) of possible outputs,

\[
\Pr[\mathcal{M}(D) \in O] \leq e^{\Phi_u} \times \Pr[\mathcal{M}(D') \in O],
\]

where \( u \in \mathcal{U} \) is the user associated with tuple \( t \), and \( \Phi_u \) denotes \( u \)'s privacy preference.

Intuitively, PDP offers the same strong, semantic notion of privacy that traditional differential privacy provides, but the privacy guarantee for PDP is personalized to the needs of every user simultaneously. As we will demonstrate later in the paper, PDP opens the door for attaining a higher level of utility when not all users require the same strong privacy level.

### 4.3.2 Properties of PDP

We start by formalizing the relationship between PDP and traditional differential privacy.

**Theorem 4.3.1 (Differential Privacy Implies PDP).** Let \( \mathcal{U} \) denote a universe of users and let \( \mathcal{D} \) denote the associated universe of tuples. Any mechanism \( \mathcal{M} : \mathcal{D} \rightarrow \mathcal{R} \) that satisfies \( \epsilon \)-differential
privacy also satisfies Φ-PDP, with privacy specification Φ = \{(u, ε) | u ∈ U\}.

The proof follows immediately from the definitions of differential privacy (Def. 2.1.2) and PDP (Def. 4.3.3).

The ability to compose nicely is an important property for practical privacy definitions. The composition properties of traditional differential privacy extend naturally to PDP. For simplicity, our statement assumes that mechanisms operate on datasets with the same schema (i.e., they have the same attributes).

**Theorem 4.3.2** (Composition). Let \( M_1 : D_1 → R \) and \( M_2 : D_2 → R \) denote two mechanisms that satisfy PDP for \( Φ_1 \) and \( Φ_2 \), respectively. Let \( U_1 \) and \( U_2 \) denote the associated universes of users. Finally, let \( D_3 = D_1 ∪ D_2 \). Then, for any \( D \subset D_3 \), the mechanism \( M_3(D) = g(M_1(D ∩ D_1), M_2(D ∩ D_2)) \) satisfies \( Φ_3 \)-PDP, where \( Φ_3 = (\{(u, Φ_1^u + Φ_2^u) | u ∈ U_1 ∩ U_2\} ∪ \{(v, Φ_1^v) | v ∈ U_1 \setminus U_2\} ∪ \{(w, Φ_2^w) | w ∈ U_2 \setminus U_1\}) \), and \( g \) is an arbitrary function of the outputs of \( M_1 \) and \( M_2 \).

**Proof.** Let \( D \sim D' \), with \( D, D' \subset D_{1,2} \) be an arbitrary pair of neighboring datasets. First, let us consider the case where \( \tau \in U_1 \cap U_2 \). We have that \( \tau \in D' \cap D_1 \) and \( \tau \in D' \cap D_2 \). To simplify notation a bit, let \( ε_1 = Φ_1^{\tau U} \) and \( ε_2 = Φ_2^{\tau U} \). For any \( O \subseteq \text{Range}(M_3) \), we can write

\[
\Pr[M_3(D) ∈ O] = \sum_{(r_1, r_2) ∈ O} \Pr[M_1(D ∩ D_1) = r_1] \cdot \Pr[M_2(D ∩ D_2) = r_2].
\]

Applying Def. 4.3.3 for both \( A_1 \) and \( A_2 \), we have that for any \( O \subseteq \text{Range}(A_3) \),

\[
\Pr[A_3(D) ∈ O] ≤ \sum_{(r_1, r_2) ∈ O} (e^{ε_1} \Pr[A_1(D') = r_1]) (e^{ε_2} \Pr[A_2(D') = r_2])
\]

\[
= e^{(ε_1 + ε_2)} \sum_{(r_1, r_2) ∈ O} \Pr[A_1(D') = r_1] \Pr[A_2(D') = r_2]
\]

\[
= e^{(ε_1 + ε_2)} \Pr[A_3(D') ∈ O]
\]

Thus, \( Φ_3^{\tau U} = (ε_1 + ε_2) = (Φ_1^{\tau U} + Φ_2^{\tau U}) \) as claimed. Next we consider the case in which \( \tau U \in U_1 \setminus U_2 \).
Observe that \( \Pr[A_2(D) = r_2] = \Pr[A_2(D') = r_2] \), since \( D \cap U_2 = D' \cap U_2 \). Thus, we have

\[
\Pr[A_3(D) \in O] \leq \sum_{(r_1, r_2) \in O} \left( e^{\epsilon_1} \Pr[A_1(D') = r_1] \right) \Pr[A_2(D') = r_2] \\
= e^{\epsilon_1} \sum_{(r_1, r_2) \in O} \Pr[A_1(D') = r_1] \Pr[A_2(D') = r_2] \\
= e^{\epsilon_1} \Pr[A_3(D') \in O]
\]

and \( \Phi_3^{t_\mathcal{U}} = \epsilon_1 = \Phi_1^{t_\mathcal{U}} \), as claimed. An analogous argument can be made for the case in which \( t_\mathcal{U} \in U_2 \setminus U_1 \).

One may also compose a differentially private mechanism with a PDP mechanism, yielding a new PDP mechanism.

**Corollary 4.3.3 (Composition with Differential Privacy).** Let \( \mathcal{M}_1 : \mathcal{D} \to R \) be a mechanism that satisfies \( \epsilon \)-differential privacy, and let \( \mathcal{M}_2 : \mathcal{D} \to R \) be a mechanism that satisfies \( \Phi \)-PDP, in the context of a universe of users \( \mathcal{U} \). Then the mechanism \( \mathcal{M}_3(D) = g(\mathcal{M}_1(D), \mathcal{M}_2(D)) \) satisfies \( \Phi_3 \)-PDP, where \( \Phi_3 = \{(u, \Phi^u + \epsilon) | u \in \mathcal{U}\} \), for any \( D \subset \mathcal{D} \), and \( g \) is an arbitrary composition function.

The proof follows immediately from Theorems 4.3.2 and 4.3.1.

### 4.4 Mechanisms for Achieving PDP

In this section, we present general mechanisms for achieving PDP for arbitrary functions. We begin by establishing some naïve baseline mechanisms which represent the limited options available to an analyst when employing traditional differential privacy in the presence of non-uniform privacy preferences. In that sense, the baselines will allow us to compare PDP against traditional differential privacy.
4.4.1 Baseline Mechanisms

The naïve baseline mechanisms that we introduce now technically achieve PDP, but fail to take advantage of the personalized privacy preferences to benefit utility. In the rest of the section, we will use the notation $DP^f_\epsilon(D)$ to denote any mechanism that computes the function $f$ on an input $D$ and satisfies the traditional $\epsilon$-differential privacy definition. For example, if $f$ is the mean function, $DP^f_\epsilon$ could be an instance of the Laplace mechanism; or, if $f$ is the median, $DP^f_\epsilon$ might be realized by an instance of the exponential mechanism. However, $DP^f_\epsilon$ could also be a more complex composition of multiple differentially private components.

The first baseline mechanism is simply a direct application of Theorem 4.3.1. That is, we find the strongest privacy preference in a given privacy specification (i.e., $\alpha = \min_u \Phi^u$) and then invoke $DP^f_\alpha$ using that as the global privacy parameter.

**Definition 4.4.1** (Minimum). Given function $f : D \rightarrow R$, dataset $D \subset \mathcal{D}$, and a privacy specification $\Phi$, the $\Phi$-PDP Minimum mechanism $M_f(D, \Phi)$ releases $DP^f_\alpha(D)$, where $\alpha = \min_u \Phi^u$.

**Proof.** We need to show that for any arbitrary neighboring datasets $D \sim D' \subset \mathcal{D}$, and any $O \subseteq \text{Range}(M_f)$, $\Pr[M_f(D, \Phi) \in O] \leq e^{\Phi^u} \Pr[M_f(D', \Phi) \in O]$. Since $DP^f_\alpha(D)$ satisfies $\alpha$-differential privacy for $\alpha = \min_u \Phi^u$, it also satisfies $\Phi^\alpha$-PDP for $\Phi^\alpha = \{(u, \alpha) | u \in \mathcal{U}\}$ (by Thm. 4.3.1). Thus, $\Pr[M_f(D, \Phi) \in O] \leq e^{\min_u \Phi^u} \Pr[M_f(D', \Phi) \in O] \leq e^{\Phi^u} \Pr[M_f(D', \Phi) \in O]$, as desired.\[\square\]

Although Minimum satisfies PDP, it gains no benefit from the personalized privacy preferences; most users will receive a much stronger level of privacy than they require. If we have a dataset where there are relatively few very privacy conscious users and a larger set of less concerned users, another option is to simply discard all of the tuples belonging to the privacy conscious users. We would then add noise according to the strictest remaining user. This is the idea behind the next baseline mechanism.

**Definition 4.4.2** (Threshold). Given function $f : D \rightarrow R$, dataset $D \subset \mathcal{D}$, and a privacy
specification \( \Phi \), the \( \Phi \)-PDP Threshold mechanism \( T_f(D, \Phi, t) \) first constructs from \( D \) the dataset \( D_t = \{ x \in D | \Phi^{xU} \geq t \} \) and then releases \( DP_f^t(D_t) \).

Proof. First, consider an arbitrary pair of neighboring datasets \( D \sim D' \subset D \), with \( \Phi^{xU} < t \). In this case, the datasets \( D_t \) and \( D'_t \), constructed from \( D \) and \( D' \) by removing all the tuples belonging to users with privacy settings below \( t \), will be equivalent since \( x \) will be one of the tuples omitted from both \( D_t \) and \( D'_t \). Then clearly, for any \( O \subseteq \text{Range}(T_f) \), \( \text{Pr}[T_f(D, \Phi, t) \in O] = \text{Pr}[T_f(D', \Phi, t) \in O] \), which satisfies definition 4.3.3.

Next we consider the case where \( \Phi^{xU} \geq t \). In this case, \( D \sim D' \). Now, observe that \( T_f(D, \Phi, t) = DP_f^t(D_t) \), which satisfies \( t \)-differential privacy, and therefore also satisfies \( \Phi_t \)-PDP for \( \Phi_t = \{ (u, t) | u \in \mathcal{U} \} \) (by Thm. 4.3.1). Then, for any \( O \subseteq \text{Range}(T_f) \),

\[
\text{Pr}[T_f(D, \Phi, t) \in O] \leq e^{\epsilon} \text{Pr}[T_f(D', \Phi, t) \in O] \leq e^{\Phi^{xU}} \text{Pr}[T_f(D', \Phi, t) \in O],
\]

as desired.

\( \square \)

### 4.4.2 Achieving PDP via Sampling

We now present a smarter general purpose mechanism for achieving PDP that in many cases is able to attain a higher level of utility than the baselines. The mechanism works by introducing two independent sources of randomness into a computation: (1) non-uniform random sampling at the tuple level, where the inclusion probability for a tuple depends on the personal privacy preference of the corresponding user (and a global threshold \( t \)), and (2) additional uniform randomness introduced by invoking a traditional differentially private mechanism on the sampled input, where the privacy parameter \( \epsilon \) depends on \( t \). Combined, the two sources of randomness yield the precise amount of privacy required by each tuple.

**Definition 4.4.3 (The Sample Mechanism).** Consider a function \( f : \mathcal{D} \rightarrow R \), a dataset \( D \subset \mathcal{D} \), and a privacy specification \( \Phi \). Let \( RS(D, \Phi, t) \) denote the procedure that independently samples
each tuple \( x \in D \) with probability

\[
\pi_x = \begin{cases} 
\frac{e^{\Phi^x U} - 1}{e^t - 1} & \text{if } \Phi^x U < t \\
1 & \text{otherwise}
\end{cases}
\]

where \( \Phi^x U \) denotes the privacy preference of the user associated with tuple \( x \), and \( \min_u \Phi^u \leq t \leq \max_u \Phi^u \) is a configurable threshold. The Sample mechanism is defined as

\[
S_f(D, \Phi, t) = \text{DP}_t^f(RS(D, \Phi, t))
\]

where \( \text{DP}_t^f \) is any \( t \)-differentially private mechanism that computes the function \( f \).

**Theorem 4.4.1.** The Sample mechanism \( S_f \) satisfies \( \Phi \)-PDP.

**Proof.** We will use the notation \( D_{-x} \) (or \( D_{+x} \)) to mean the dataset resulting from removing (adding to) \( D \) the tuple \( x \). Thus, we can represent two neighboring datasets as \( D \) and \( D_{-x} \). We will show that for any \( D, D_{-x} \) and any \( O \in \text{Range}(S_f) \),

\[
\Pr[S_f(D, \Phi, t) \in O] \leq e^{\Phi^x U} \Pr[S_f(D_{-x}, \Phi, t) \in O].
\]

We will drop the subscript from \( \Phi^x U \) to simplify notation in the remainder of the proof. Observe that all of the possible outputs of \( RS(D, \Phi, t) \) can be divided into those in which \( x \) was selected, and those in which \( x \) was not selected. Thus, we can write \( \Pr[S_f(D, \Phi, t) \in O] \) as

\[
\sum_{Z \subseteq D_{-x}} (\pi_x \Pr[RS(D_{-x}, \Phi, t) = Z] \Pr[DP_t^f(Z+x) \in O]) \\
+ \sum_{Z \subseteq D_{-x}} ((1 - \pi_x) \Pr[RS(D_{-x}, \Phi, t) = Z] \Pr[DP_t^f(Z) \in O])
\]

(4.1)

\[
= \sum_{Z \subseteq D_{-x}} (\pi_x \Pr[RS(D_{-x}, \Phi, t) = Z] \Pr[DP_t^f(Z+x) \in O])
\]

\[
+ (1 - \pi_x) \Pr[S_f(D_{-x}, \Phi, t) \in O].
\]
Since $DP^f_t$ satisfies $\Phi_t$-PDP, for $\Phi_t = \{(u,t) | u \in U\}$ (Thm. 4.3.1), we have

$$\Pr[DP^f_t(Z+x) \in O] \leq e^t \Pr[DP^f_t(Z) \in O].$$

Thus, equation (4.1) can be rewritten as

$$\Pr[S_f(D,\Phi,t) \in O] \leq \sum_{Z \subseteq D-x} (\pi_x \Pr[RS(D-x,\Phi,t) = Z|(e^t \Pr[DP^f_t(Z) \in O])]
\quad + (1 - \pi_x) \Pr[S_f(D-x,\Phi,t) \in O]
\quad = \pi_x(e^t \Pr[S_f(D-x,\Phi,t) \in O]) + (1 - \pi_x) \Pr[S_f(D-x,\Phi,t) \in O]
\quad = (1 - \pi_x + \pi_xe^t) \Pr[S_f(D-x,\Phi,t) \in O].$$

In substituting for $\pi_x$ in (4.2), there are two cases for $x$ that we must consider: the case in which $\Phi^x \geq t$, and the case in which $\Phi^x < t$. Let us consider the former case first. By definition, when $\Phi^x \geq t$, tuple $x$ is selected with probability $\pi_x = 1$; thus substituting 1 for $\pi_x$ in (4.2), we have

$$\Pr[S_f(D,\Phi,t) \in O] \leq (1 - \pi_x + \pi_xe^t) \Pr[S_f(D-x,\Phi,t) \in O]
\quad = e^t \Pr[S_f(D-x,\Phi,t) \in O]
\quad \leq e^{\Phi^x} \Pr[S_f(D-x,\Phi,t) \in O];$$

as desired. Let us now consider the case in which $\Phi^x < t$. Expanding $\pi_x$ in $(1 - \pi_x + \pi_xe^t)$ in equation (4.2), we get:

$$1 - \pi_x + \pi_xe^t = 1 - \frac{e^{\Phi^x} - 1}{e^t - 1} + \frac{e^{\Phi^x} - 1}{e^t - 1} e^t
\quad = \frac{e^t - e^{\Phi^x} + e^{\Phi^x+t} - e^t}{e^t - 1}
\quad = \frac{-e^{\Phi^x} + e^{\Phi^x+t}}{e^t - 1} = \frac{(e^t - 1)e^{\Phi^x}}{e^t - 1} = e^{\Phi^x}.$$ 

Thus, we have $\Pr[S_f(D,\Phi,t) \in O] \leq e^{\Phi^x} \Pr[S_f(D-x,\Phi,t) \in O]$, and therefore $S_f(D,\Phi,t)$
satisfies $\Phi$-PDP.

**Remark.** Our Sampling mechanism is inspired by a result from [73], where the authors observed that random sampling has a “privacy amplification” effect when combined with differential privacy. Further discussion of prior work related to sampling in the context of differential privacy can be found in Section 4.2.

### 4.4.2.1 Discussion

We make a few important observations regarding the Sample mechanism. First, we emphasize that the Sample mechanism is not limited to simple aggregates like counts, sums, etc. In fact, the Sample mechanism is immediately applicable to arbitrarily complex functions, so long as there is a known differentially private algorithm for computing $f$, i.e., $DP^f$. The mechanism $DP^f$ could be a simple instantiation of the Laplace or exponential mechanisms, or a more complex composition of several differentially private mechanisms. The Sample mechanism essentially treats $DP^f$ as a black box that operates on a dataset of tuples.

Second, we note that the Sample mechanism effectively introduces two types of randomness—and hence two types of error—into $f$. The threshold $t$ optionally\(^1\) provides a means of balancing these types of error. A small $t$ results in fewer tuples being discarded by the sampling step (and lower sampling error), but results in more randomness (e.g., noise) due to $DP^f_t$. Observe that when $t = \max_u \Phi^u$, every tuple is provided with the precise amount of privacy it requires. When $t = \min_u \Phi^u$, the Sample mechanism collapses down to the Minimum baseline mechanism.

The tunable threshold is useful because the two types of error can impact the resulting output differently. Using $t = \max_u \Phi^u$, so that all users receive exactly the required amount of privacy, may not always give the best results; often, by using a lower threshold we can significantly reduce the sampling error while not introducing too much extra noise. As a concrete example, consider the count aggregate. In this case, we have $D \in \{0, 1\}^n$, and $DP^f_t = f(D) +$\(^1\)Simply setting $t = \max_u \Phi^u$ offers good results in many cases, as we will show later in our experimental study.
\[ \text{Lap}(1/t), \text{where } f = \text{count}(D) \text{ is the function that counts the number of non-zero tuples in } D. \]

Observe that the magnitude of the Laplace noise depends only on the sensitivity of \( f \) (which is 1 in this case) and \( t \). Thus, the larger \( D \) is, the smaller the noise relative to the count. The error due to sampling, on the other hand, depends not only on the privacy specification, but also on the density of the data. Thus for sufficiently large datasets, setting a lower threshold (i.e., \( t \ll \max_u \Phi^u \)) could greatly increase the sample rate for the users with strong privacy requirements, at the cost of slightly more noise, but a lower total error. The following example illustrates.

**Example 1.** For the count aggregate, suppose that we have a dataset \( D \) with \( n = 200 \) tuples, each corresponding to one user, with a selectivity value of 0.1. For simplicity, assume that users fall into either one of two groups, w.r.t. privacy preferences: conservative users have a strong privacy requirement, say \( \epsilon_C = 0.1 \), and liberal users have a relatively weak requirement, say \( \epsilon_L = 1 \). If we set \( t = \epsilon_L \), then each of the conservative tuples would be retained with probability \( \pi_C = \frac{e^{\epsilon_C - 1}}{e^1 - 1} \approx 0.125 \). Let \( \tilde{D} = \text{RS}(D, \Phi, t = \epsilon_L) \) denote the sampled dataset. If we assume that a majority of the users, say 65%, are conservative, then the squared error due to sampling is calculated as

\[
\text{Err(count}(\tilde{D})) = \text{Var(count}(\tilde{D})) + \text{Bias(count}(\tilde{D}))^2 = (n \cdot 0.65 \cdot 0.1) \cdot \pi_C \cdot (1 - \pi_C) + ((n \cdot 0.65 \cdot 0.1) \cdot (\pi_C - 1))^2 \approx 150,
\]

while the (additional) error due to the Laplace noise injected by \( \text{DP}_{1}^{f} \) is \( \text{Var} \left( \text{Lap}(1/1) \right) = 2(1/1)^2 = 2 \). However, observe that if we instead set \( t = 0.2 \), we get \( \pi_C \approx 0.475 \) and \( \text{Err(count}(\tilde{D})) \) is reduced to \( \approx 50 \), while the noise-related error increases to 50. Thus, the total squared error is reduced from 152 to 100.

More complex functions that have a relatively high sensitivity, but are robust to sampling, especially for larger datasets, will see less benefit from threshold optimization. Recall that global sensitivity is a worst case measure of the impact a single tuple can have on the output of a function; however, for many functions, while the global sensitivity may be quite high, the impact
that most tuples will actually have on the output is relatively small. For such functions, the same level of sampling error buys a significantly greater reduction in error due to \( DP^f \), as the sampling rate is independent of the sensitivity.

Precisely optimizing \( t \) for an arbitrary \( f \) may be non-trivial in practice because, although the error of \( DP^f_t \) may be quantified without knowledge of the dataset, the impact of sampling does depend on the input data. Therefore, care must be taken so as to not leak privacy through the tuning process. A possible option, in some cases, is to make use of old data that is no longer sensitive (or not as sensitive), and that comes from a similar distribution, to approximately optimize the threshold without violating privacy\(^2\). In other cases, it might be feasible to use a portion of the privacy budget of the more conservative users and estimate the required quantities from that subset of the data. We postpone an in-depth study of threshold optimization strategies for future work. In Section 4.5, we will demonstrate that for many functions, the simple heuristics of setting \( t = \max_u \Phi^u \) or \( t = \frac{1}{|U|} \sum_u \Phi^u \), often give good results on real data and privacy specifications.

### 4.4.3 Direct Approach

In the previous section, we introduced a two-step mechanism that achieves PDP by a sampling step, followed by a standard differentially private mechanism. Next we develop a more direct approach for achieving PDP, analogous to the exponential mechanism [88] for differential privacy. Our approach can be applied easily to aggregates like counts, medians, min/max and others. We first review the exponential mechanism and show that the score functions used to instantiate it for many aggregate functions can be represented in a common general form. We then show how that general form can be made to satisfy PDP.

In this section we assume that a dataset contains tuples with a single numeric attribute \( A \). We use a slightly different definition of neighboring datasets to other sections to simplify the presentation. Here, we assume that \( D \) and \( D' \) differ only in the value of a tuple \( t \) (instead of in

\(^2\) The idea of using older data is commonly used to estimate parameters for differentially private systems (e.g., in the GUPT system [93]).
the presence of $t$\footnote{This alternate definition is used in the differential privacy literature when it simplifies the task at hand [30].}. We also consider datasets that have different values for an arbitrary number of tuples and will use the notation $D \oplus D'$ to denote the set of tuples in which $D$ and $D'$ differ.

Given a function $f : D \rightarrow R$, recall that the exponential mechanism $\mathcal{E}_\varepsilon(D)$ outputs $r \in \text{Range}(f)$ with probability proportional to $\exp(\frac{s(D,r)}{2\Delta_s})$, where $s(D,r)$ is a real-valued score function that outputs a higher score the better $r$ is relative to $f(D)$, and $\Delta_s$ is the sensitivity of $s$. We observe that one form of score function can be used to instantiate $\mathcal{E}$ for many common aggregate functions. For brevity, we consider three exemplar aggregates: count, median, min. For the count function, $A$ is a binary indicator; for the other functions we assume that the value is an integer in the range $[lo, hi]$. The general score function is:

$$s(D,r) = \max_{f(D')=r} -|D \oplus D'|$$

That is, the score is inversely related to the number of changes to $D$ that would be required for $r$ to become the true answer. It is easy to see that $\Delta_s = 1$ for all three of the exemplar aggregates. Next observe that $s(D,r)$ is maximized when $r = f(D)$ (that is, when $r$ is the true answer), and the score becomes smaller (more negative) the further $r$ is from the true answer. For example, suppose we have $D = \langle 3, 5, 6, 9, 11 \rangle$. With respect to min, we have $s(D,2) = s(D,4) = s(D,5) = -1$, because we only need to change one element of $D$ to make any of those the minimum value; making 11 the minimum would require changing four values, so $s(D,11) = -4$. Similarly, for median we have, $s(D,5) = s(D,9) = -1$ and $s(D,3) = s(D,10) = s(D,11) = -2$, since changing a single element could cause 5 or 9 to become the median, while making 3, 10 or 11 the median requires two changes.

We need to understand the structure of this function further to satisfy PDP. For any $D$, $r$, there may be many $D'$ that maximize equation (4.3). For instance, in the example above, we can make 5 the median by changing either 6, 9 or 11 to any value that is $\leq 5$. In traditional differential privacy it is sufficient to treat all such $D'$ equivalently. However, in the context of
PDP, where each element has its own privacy setting, it becomes necessary to make a distinction among the different $D'$ that maximize (4.3) for a given $r$. To make the intuition more concrete, consider the privacy specification $\Phi = \langle 0.1, 1, 1, 0.5, 1 \rangle$ corresponding to $D = \langle 3, 5, 6, 9, 11 \rangle$ from the earlier example. According to the PDP definition, we need $\left| \frac{\Pr[E^*(D)=5]}{\Pr[E^*(D')=5]} \right| \leq e^{0.5}$ when $D'$ is formed by modifying the 9 (e.g., $D' = \langle 3, 4, 5, 6, 11 \rangle$), but we only require that the ratio is $\leq e^{1.0}$ when $D'$ is formed by changing the 6 or the 11. However, the definition must hold regardless of what $D'$ happens to be. That is, when computing the probability distribution for $E^*(D)$, the probability for $r$ must assume that $D'$ could be any neighboring dataset. Thus, in the example above, the probability of outputting 5 must be based on the strongest privacy requirement among the elements 6, 9 and 11, i.e., 0.5.

By modifying the exponential mechanism using the general score function of equation (4.3), with weighting to incorporate the privacy specification $\Phi$ in place of the fixed $\epsilon$, we arrive at the following PDP mechanism.

**Definition 4.4.4 (PÊ Mechanism).** Given a function $f : D \rightarrow R$, an arbitrary input dataset $D \subset D$, a privacy specification $\Phi$, the mechanism $P\mathcal{E}_\Phi^f(D)$ outputs $r \in R$ with probability

$$\Pr[P\mathcal{E}_\Phi^f(D) = r] = \frac{\exp\left(\frac{1}{2}d_f(D, r, \Phi)\right)}{\sum_{q \in R} \exp\left(\frac{1}{2}d_f(D, q, \Phi)\right)} \quad (4.4)$$

where $d_f(D, r, \Phi) = \max_{f(D') = r} \sum_{i \in D \Delta D'} -\Phi_{ix}$ \quad (4.5)

It is easy to verify that for the special case where the privacy preferences in $\Phi$ are uniform, $P\mathcal{E}_\Phi^f$ reduces to an instance of the original exponential mechanism. We now prove that $P\mathcal{E}$ satisfies PDP. The proof modifies that of the original exponential mechanism [88].

**Theorem 4.4.2.** The $P\mathcal{E}$ mechanism satisfies $\Phi$-PDP.

**Proof.** Let $D \sim D' \subset D$ be two arbitrary, neighboring datasets that differ in the value for tuple $x$. In the following, let $\epsilon_x = \Phi_{xi}$, the privacy requirement for tuple $x$. First, observe that for any $D$ and $r$, if $d_f(D, r, \Phi) = y$, then there is a dataset $X$ such that $f(X) = r$ and
\[
\sum_{i \in D \oplus X} \Phi^i u = -y \text{ (by Eq. 4.5). Then, since } D \oplus D' = \{x\}, \text{ it follows that }
\]
\[
\sum_{i \in D' \oplus X} \Phi^i u \geq \sum_{i \in D \oplus X} \Phi^i u - \sum_{i \in D' \oplus D} \Phi^i u \geq -y - \epsilon_x.
\]

Since \( f(X) = r \), it follows that
\[
d_f(D', r, \Phi) \leq y + \epsilon_x = d_f(D, r, \Phi) + \epsilon_x \quad (4.6)
\]

We will use this fact below. To prove that \( \mathcal{PE} \) satisfies \( \Phi \)-PDP, we need to show that,
\[
\Pr[\mathcal{PE}^f_\Phi(D) = r] \leq \exp(\epsilon_x).
\]

By equation (4.4), we have
\[
\frac{\Pr[\mathcal{PE}^f_\Phi(D) = r]}{\Pr[\mathcal{PE}^f_\Phi(D') = r]} = \frac{\left(\exp\left(\frac{d_f(D, r, \Phi)}{2}\right)\right)}{\sum_{q \in R} \exp\left(\frac{d_f(D, q, \Phi)}{2}\right)} \cdot \frac{\left(\frac{\sum_{q \in R} \exp\left(\frac{d_f(D', q, \Phi)}{2}\right)}{\sum_{q \in R} \exp\left(\frac{d_f(D', q, \Phi)}{2}\right)}\right)}{\left(\exp\left(\frac{d_f(D', r, \Phi)}{2}\right)\right)} = A \cdot B
\]

Rewriting \( A \) using equation (4.6), we have
\[
A = \frac{\exp\left(\frac{d_f(D, r, \Phi)}{2}\right)}{\exp\left(\frac{d_f(D', r, \Phi)}{2}\right)} = \exp\left(\frac{d_f(D, r, \Phi) - d_f(D', r, \Phi)}{2}\right) \leq \exp\left(\frac{\epsilon_x}{2}\right)
\]

Similarly, rewriting \( B \) we get
\[
B = \frac{\sum_{q \in R} \exp\left(\frac{d_f(D', q, \Phi)}{2}\right)}{\sum_{q \in R} \exp\left(\frac{d_f(D, q, \Phi)}{2}\right)} \leq \frac{\sum_{q \in R} \exp\left(\frac{d_f(D, q, \Phi) + \epsilon_x}{2}\right)}{\sum_{q \in R} \exp\left(\frac{d_f(D, q, \Phi)}{2}\right)}
\]
\[
\exp \left( \frac{\epsilon x}{2} \right) \sum_{q \in R} \exp \left( \frac{d_f(D, q, \Phi)}{2} \right) = \exp \left( \frac{\epsilon x}{2} \right)
\]

Thus, we have \( A \cdot B \leq \exp \left( \frac{\epsilon x}{2} + \frac{\epsilon x}{2} \right) = \exp (\epsilon x) \). It can be similarly shown that the ratio is also \( \geq \exp (-\epsilon x) \). Therefore, the \( \mathcal{PE} \) mechanism satisfies \( \Phi\text{-PDP} \). \( \square \)

### 4.4.3.1 Concrete Examples

We describe how \( d_f \) can be efficiently computed for our exemplar aggregates. In general, finding an efficient algorithm to compute \( d_f \) for an arbitrary \( f \) may be non-trivial.

**Count:** Let \( \text{count} : \{0, 1\}^n \rightarrow \mathbb{R} \), \( \mathbb{R} = \{0, 1, \ldots, n\} \), be the function that returns the number of 1’s in the input. Consider an arbitrary input \( D \) for which \( \text{count}(D) = x, x \leq n \). To compute \( d_{\text{count}}(D, r, \Phi) \) for an arbitrary \( r \in \mathbb{R} \), there are three possible cases to consider: (1) when \( r > x \), \( d_{\text{count}}(D, r, \Phi) \) is the sum of the \( r - x \) smallest privacy settings among all the 0 bits in \( D \); (2) when \( r < x \), \( d_{\text{count}}(D, r, \Phi) \) is the sum of the \( x - r \) smallest privacy settings corresponding to the 1 bits in \( D \); (3) when \( x = r \), \( d_{\text{count}}(D, r, \Phi) = 0 \). Note that this algorithm requires sorting the privacy specification \( \Phi \), to although this need only be done once.

**Median:** Let \( R = \{lo, lo+1, \ldots, hi\} \) for \( lo, hi \in \mathbb{Z} \). For a sorted dataset \( D \in \mathbb{R}^n \), the median function \( \text{med} : \mathbb{R}^n \rightarrow \mathbb{R} \) returns the element with rank \( m = \lfloor n/2 \rfloor \) in \( D \) (for simplicity, we assume that \(|D|\) is odd). For an arbitrary \( r \in R \), let \( i \) denote the rank of \( r \) in \( D \). To compute \( d_{\text{med}}(D, r, \Phi) \), there are three cases to consider: (1) if \( i < m \), then the \( D' \) that minimizes equation (4.5) is the one derived from \( D \) by changing the \( m - i \) elements to the right of element \( i \) with the smallest privacy settings in \( \Phi \); thus \( d_{\text{med}}(D, r, \Phi) \) is just zero minus the sum of those privacy settings. For example, if \( D = \langle 3, 5, 6, 9, 11 \rangle \) and \( \Phi = \langle 0.1, 1, 1, 0.5, 1 \rangle \), then we have \( d_{\text{med}}(D, 3, \Phi) = d_{\text{med}}(D, 4, \Phi) = -1.5 \) and \( d_{\text{med}}(D, 5, \Phi) = -0.5 \), and so on. (2) Conversely, when \( i > m \), we must consider all elements to the left of \( i \) in \( D \). In this case, \( d_{\text{med}}(D, r, \Phi) \) will be zero minus the sum of the \( i - m \) smallest privacy parameters among those elements of \( D \) with rank \( < i \). For example, considering the \( D, \Phi \) given above, \( d_{\text{med}}(D, 11, \Phi) = -0.6 \). (3) When
\(i = m, \ d_{\text{med}}(D, r, \Phi) = 0\). Compared to finding a median with differential privacy, the PDP implementation additionally needs to sort \(\Phi\), leading to only a slight increase in computational overhead.

**Min:** As before, let \(R = \{lo, lo + 1, \ldots, hi\}\) for \(lo, hi \in \mathbb{Z}\). For a sorted dataset \(D \in R^n\), the \(\text{min}\) function \(\text{min} : R^n \rightarrow R\) returns the smallest element of input \(D\). For an arbitrary \(r \in R\), there are again three cases: (1) when \(r > \text{min}(D)\), observe that for \(r\) to become the minimum, all elements \(q \in D\) in which \(q < r\) would have to be changed so that their values are \(\geq r\). Thus \(d_{\text{min}}(D, r, \Phi)\) would be equal to the sum of the privacy settings of all elements in \(D\) with a value less than \(r\). (2) When \(r < \text{min}(D)\), for \(r\) to become the minimum, the value of any single element in \(D\) could be changed to \(r\); thus, \(d_{\text{min}}(D, r, \Phi)\) equals the minimum privacy setting among the elements in \(D\). (3) When \(r = \text{min}(D)\), \(d_{\text{min}}(D, r, \Phi) = 0\).

For large \(R\), we can improve efficiency for median and min by observing that all \(r \in R\) that fall between two consecutive elements in \(D\), say \(p\) and \(q\), have the same rank as either \(p\) or \(q\). Therefore, we can divide the output space into ranges and compute probabilities for each range (multiplied by the size of the range). When a range is selected by the mechanism, the returned value is sampled uniformly from within it.

### 4.5 Experimental Study

Next, we apply our PDP mechanisms to two common aggregate functions, \(\text{count}\) and \(\text{median}\), as well as to the more complex task of \(\text{multiple linear regression}\). Although count and median are relatively simple functions, they are important primitives for building more complex algorithms [12]. For the count and median functions, we compare the Sample mechanism and the exponential-like \(PE\) mechanism. For linear regression, we use the Sampling mechanism to transform a recent differentially private approach for linear regression, introduced by Zhang et al. [140], yielding a PDP version of the algorithm.

Our main goal for this experimental study is to demonstrate that by taking personal privacy
preferences into account, our proposed PDP mechanisms can often attain more accurate data analysis results, compared to traditional differential privacy, which provides only a uniform privacy guarantee. To that end, we compare to the baseline mechanisms Minimum and Threshold (see Section 4.4.1), in terms of root mean squared error (RMSE) on real and synthetic data, under different data distributions and privacy specifications.

**Datasets:** We evaluate mechanisms for count and median on synthetic data. For count, we generate datasets with 1,000 records, each with a single binary attribute. The fraction of records with a value of ‘1’ is controlled by a density parameter, $\delta$, in the range (0,1) (default $\delta = 0.15$). For the median function we generate datasets with 1,001 records, where the attribute values are randomly drawn from a normal distribution with mean $\mu$ and standard deviation $\sigma$ (defaults, $\mu = 500, \sigma = 200$), rounded to the nearest integer in the range [1,1000]. For the linear regression task, we use a dataset containing 100,000 records (representing 100,000 users) from the 2012 US Census [109], detailed in in Section 4.5.2.

**Privacy Specification:** To generate the privacy specifications for our experiments, we randomly divided the users (records) into three groups: conservative, representing users with high privacy concern; moderate, representing users with medium concern; and liberal, representing users with low concern\(^4\). The fraction of users in the conservative and moderate groups were determined by the parameters $f_C$ and $f_M$; the fraction of users in the liberal group is $1.0 - (f_C + f_M)$. The default values used in our experiments were $f_C = 0.54$ and $f_M = 0.37$ and were chosen based on findings reported in [2] in the context of a user survey about privacy concern. The privacy preferences for the users in the conservative and moderate groups were drawn uniformly at random from the ranges $[\epsilon_C, \epsilon_M]$ and $[\epsilon_M, \epsilon_L]$, respectively (and rounding to the nearest hundredth), with $\epsilon_C, \epsilon_M, \epsilon_L \in [0.01, 1.0]$; the users in the liberal group received a privacy preference of $\epsilon_L$, which was fixed at $\epsilon_L = 1.0$ for all of the experiments in this paper. The defaults for the other two parameters were $\epsilon_C = 0.01$ and $\epsilon_M = 0.2$, where a smaller value yields greater privacy. Table 4.1 lists the various parameters used in our experiments and the

\(^4\)The choice to partition users into low/medium/high privacy groups was based on findings from several studies by other researchers regarding user privacy attitudes (e.g., [6, 2, 1]).
ranges of values we tested for each, with the default values underlined. In the rest of the section, unless explicitly stated otherwise, the default values listed in the table were used.

Table 4.1 Experiment parameters (defaults underlined).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$ (count, median)</td>
<td>1000 (count); 1001 (med.)</td>
</tr>
<tr>
<td>$n$ (lin. regression)</td>
<td>10000, 20000, ..., 100000</td>
</tr>
<tr>
<td>$\delta$ (count only)</td>
<td>0.01, 0.05, 0.1, 0.15, ..., 0.5</td>
</tr>
<tr>
<td>$\epsilon_C$</td>
<td>0.01, 0.05, 0.1, 0.2, ..., 0.5</td>
</tr>
<tr>
<td>$\epsilon_M$</td>
<td>0.05, 0.1, 0.15, 0.2, ..., 0.5</td>
</tr>
<tr>
<td>$\epsilon_L$</td>
<td>1.0</td>
</tr>
<tr>
<td>$\sigma$ (median only)</td>
<td>100, 200, ..., 1000</td>
</tr>
<tr>
<td>$\mu$ (median only)</td>
<td>500</td>
</tr>
<tr>
<td>$f_C$</td>
<td>0.1, 0.2, ..., 0.6; 0.54</td>
</tr>
<tr>
<td>$f_M$</td>
<td>0.37</td>
</tr>
<tr>
<td>$f_L$</td>
<td>$1.0 - (f_C + f_M)$</td>
</tr>
</tbody>
</table>

### 4.5.1 PDP for Count and Median

In this section we apply our two PDP mechanisms to the count and median functions. We compared the RMSE of four main approaches: the Minimum and Threshold baselines ($M$ and $T$, respectively), the Sampling mechanism with threshold $t = \max_u \Phi^u = \epsilon_L$ (denoted $S$), and the exponential-like $\mathcal{PE}$ mechanism. Recall that $M$ invokes a standard differentially private mechanism (the Laplace mechanism for count and the exponential mechanism for median), using $\epsilon_C$ as the privacy parameter. $T$ works by first discarding all but the liberal user data and then invoking a differentially private mechanism with $\epsilon_L$ as the privacy parameter. Additionally, we investigated a variation of $S$ with the heuristic of setting the sampling threshold to $t = \frac{1}{n} \sum_u \Phi^u$ (i.e., the average privacy setting), as suggested in Section 4.4.2; we denote this approach $S$-avg. Finally, for the count task, we also considered the Stretching mechanism introduced by Alaggan et al. [4] in the context of their similar privacy model, HDP (it does not apply for the median task). Translated to our framework, the Stretching mechanism works by (1) multiplying the
data value of each tuple $i$ by a scaling factor $\frac{\Delta_i}{\epsilon_L}$, where $\epsilon_i$ is the privacy setting for tuple $i$, and

(2) releasing $f(D') + \text{Lap}(1/\epsilon_L)$, where $D'$ is the scaled data set. We write $\text{Alag.}$ to denote this approach in the results.

For a given configuration of the parameters in Table 4.1, we computed the RMSE for each mechanism (between the private count/median and the true count/median) over 1,000 runs, using a different randomly generated dataset and privacy specification for each run.

Figure 4.1 (Count) RMSE of each mechanism for the count task, as four parameters are varied.
4.5.1.1 Impact of Data Distribution

We first examine the impact of the data density $\delta$ (i.e., the fraction of 1’s in the input data) on the count function. Figure 4.1a shows the RMSE for count as a function of increasing $\delta$. The results indicate that when density is low (e.g., $\delta < 0.1$), $S$ and $S$-avg offer the lowest error, since most of the discarded tuples have a value of zero and the count is only affected when a 1 is discarded. However, we see that in general counts are highly sensitive to sampling, and as the density increases, the error of the sampling-based approaches quickly exceeds that.
of the $\mathcal{M}$ baseline. For denser datasets (e.g., $\delta > 0.1$), $\mathcal{PE}$ is the clear winner, outperforming all other approaches by a significant margin. For example, when $\delta = 0.3$, the error of $\mathcal{PE}$ is less than half that of the next best mechanism. An important observation is that, unlike the sampling-based mechanisms, $\mathcal{PE}$ is able to make use of all of the data in the input and, like the original exponential mechanism, is generally unaffected by the data density. This also means that, as the density (or the size of the dataset) increases, the relative error will tend toward zero, yielding highly accurate counts. For $Alag.$, we observed slightly lower error than for $\mathcal{S}$; however, $Alag.$ was still significantly outperformed by $\mathcal{PE}$. Finally, we observed that the simple threshold heuristic employed by $\mathcal{S}$-avg works surprisingly well here, offering a significant reduction in error compared to $\mathcal{S}$ (which uses $t = \max_u \Phi^u$).

For median, we vary the standard deviation $\sigma$. Figure 4.2a, shows that as $\sigma$ increases, and the data become more spread out, the errors of all approaches increase. When $\sigma$ is small, there are many values concentrated around the median and the output is therefore less affected by individual values that are discarded due to sampling. Likewise, for $\mathcal{M}$ and $\mathcal{PE}$, when the values are concentrated around the median, most of the probability mass will be concentrated on a small range of output values close to the true median. Compared to the count function, the median is far more resistant to sampling; consequently $\mathcal{S}$ and $\mathcal{S}$-avg offer a considerable reduction in error, relative to the baseline approaches. For the same reason, $\mathcal{T}$ fares much better in this task, compared to count; however, it is still significantly outperformed by the $\mathcal{S}$. Although $\mathcal{PE}$ significantly outperforms $\mathcal{M}$, it appears to be no better than $\mathcal{T}$ in this scenario. Finally, we note that, in contrast to the count task, $\mathcal{S}$-avg offers only a slight improvement over using the default threshold; again, this appears to be a consequence of the median’s stronger resistance to sampling.

### 4.5.1.2 Varying the Privacy Specification

In the previous experiments, a majority of the users (54%) were assumed to be in the conservative group; that is, most of the data records were assigned privacy preferences in the range
[\epsilon_C, \epsilon_M], while relatively few were assigned privacy preferences equal to \epsilon_L. In cases where a larger fraction of the users are liberal (i.e., when \(f_C\) is small) we would expect the PDP mechanisms to perform even better. We confirmed this by varying \(f_C\), while keeping the other parameters at their defaults. Note that for each setting of \(f_C\), the fraction of liberal users is equal to 1 - (\(f_C + f_M\)), so decreasing \(f_C\) increases the number of liberal users. The results are shown in Figures 4.1b (for count) and 4.2b (for median). The sharp increase in the error of \(T\) when \(f_C = 0.6\) (Figure 4.2b) is when the median is being computed over only 3% of the dataset (30 records). Note that \(M\) does not depend on the \(f_C\) parameter, which is why its \text{RMSE} appears unaffected.

Next, we vary \(\epsilon_C\), which controls the lower bound on the range of privacy settings generated for the conservative users. Figures 4.1c and 4.2c show the results for count and median, respectively. Note that we used \(\epsilon_M = 0.5\) (rather than the default of \(\epsilon_M = 0.2\)) for this experiment, to ensure \(\epsilon_C \leq \epsilon_M\) in all cases. The key observation here is that the benefits of PDP, in terms of reduced error, diminish as the privacy requirements of the conservative users become weaker (closer to \(\epsilon_L\)). This is because the error for the \(M\) decreases exponentially with increasing \(\epsilon_C\), while the reduction in error for the PDP mechanisms is much more subtle. For the count function, we see that when \(\epsilon_C\) is larger than about 0.08, \(M\) becomes the best choice. For median, on the other hand, \(S\)-avg remains the best choice until \(\epsilon_C\) is larger than about 0.25, at which point the benefits of using PDP diminish. We note that the \(\epsilon_C\) parameter is not used by \(T\), so observed variations are from the independent repetitions.

We now look at the impact of varying \(\epsilon_M\). Recall that increasing \(\epsilon_M\) has the effect of raising the upper (lower) bound on the range of conservative (moderate) privacy settings. Therefore, we would expect the error for the PDP approaches to be smaller with a higher \(\epsilon_M\). Figures 4.1d and 4.2d show the results for count and median, respectively. For the sampling-based approaches, the error reduction was much more pronounced for count than for median, due to count’s considerably lower resistance to sampling. The PE mechanism benefited greatly, with respect to both tasks, from the larger number of users with weaker privacy requirements.
4.5.2 PDP for Multiple Linear Regression

In this section, we demonstrate how the Sampling mechanism can be easily used to convert an existing differentially private algorithm into one that satisfies PDP. In particular, we focus on the task of linear regression, where the objective is to learn a linear model that can be used to predict the value of a response variable $y$ from one or more predictor variables $A$. That is, given training dataset $D_T$, with rows of the form $A_{i1}, \ldots, A_{ik}, y_i$, we wish to learn a model $y = A \times w + b$, where the parameter vector $w$ and the intercept term $b$ are the outputs of the training process. For the experiments in this section, our goal was to (privately) learn a model to accurately predict an individual’s income based on a set of other attributes (e.g., age, gender, number of children, etc.).

To do so, we adapt a differentially private linear regression algorithm, due to Zhang et al. [140], to satisfy PDP. The approach perturbs the coefficients of the objective function with Laplace noise, and then optimizes the perturbed objective function. The sensitivity—and hence the scale of the Laplace noise—depends on the number of attributes in the dataset. The algorithm can be easily extended to satisfy PDP by applying it in Definition 4.4.3. In other words, we choose a threshold $t$, sample the tuples in the input according to their privacy preferences and $t$, and then pass the sampled data directly into the differentially private algorithm described in [140], using $t$ as the privacy parameter. We modified a publicly available Matlab implementation\(^5\) of the original algorithm.

Dataset: We used a random sample of the 2012 US Census data from the Integrated Public Use Microdata Series [109] comprised of 100,000 records, each representing a unique individual living in the US. The dataset contained 12 attributes (five nominal and seven numeric): receivesFoodstamps, gender, maritalStatus, employmentStatus, ownsHouse, nBedrooms, nVehicles, nChildren, age, timesMarried, nHoursWorked, and income. We restricted the sample to only those individuals with a positive income. Of the nominal attributes, only maritalStatus had more than two values (i.e., married, single, divorced/widowed). Following [140], and oth-

\(^5\)http://sourceforge.net/projects/functionalmecha/
ers, we replace this attribute with two binary attributes *isMarried* and *isSingle*. Thus, the final dataset contained 13 attributes.

**Experiment Setup:** Our task is to model the income attribute based on the other attributes. We compared $S$ and $S$-avg to the two baseline mechanisms, $M$ and $T$ as well as the Stretching mechanism, *Alag*. [4]. Note that $PE$ is not applicable to this task. For each experiment, we performed 500 runs of five-fold cross-validation, using a different randomly generated privacy specification for each run. We computed the RMSE of each approach over the 500 runs. As a preprocessing step, the linear regression implementation normalizes all attributes to the range $[-1, 1]$; thus, the reported errors are interpreted relative to that range.

**4.5.2.1 Results**

As we did for count and median, we vary $\epsilon_C$, $\epsilon_M$ and $f_C$ to obtain different privacy specifications (Figures 4.3a, 4.3b and 4.3c, respectively). We also investigated the impact of the dataset cardinality $n$ (Figure 4.3d) by running the mechanisms (with default parameters) on different sized random subsets of the main dataset. We also plot the error of the non-private linear regression algorithm (denoted *non-private* in the plots).

Looking at Figure 4.3a, we see that $S$ significantly outperformed the baselines for $\epsilon_C < 0.3$. In contrast to the previous experiments, the threshold heuristic $S$-avg, performs less well here. This can be explained by the fact that the linear regression algorithm has a much higher sensitivity than count and median; consequently the effect of the Laplace noise is far more significant than the effect of sampling (up to a point), and so trading more noise for a higher sampling rate turns out to be a bad strategy. In contrast to what was seen for count, $S$ significantly outperformed the HDP mechanism *Alag*. This is perhaps not too surprising since, intuitively, rescaling the values causes them to lose much of their meaning.

The observations regarding the impact of $\epsilon_M$ (Figure 4.3b) and $f_C$ (Figure 4.3c) are similar to those for the median task. As $\epsilon_M$ gets larger, the average privacy preference for the conservative and moderate users increases (i.e., becomes weaker), leading to fewer records being...
Figure 4.3 (Linear Regression) RMSE of each mechanism for linear regression, as four parameters are varied.

discarded due to sampling, and consequently a lower RMSE. When $f_C$ is small (i.e., there are fewer conservative users and more liberal users), the error for the sampling-based mechanisms is lower, but the improvement relative to $T$ is also quite small. On the other hand, when the conservative users make up half of the users in the dataset, the RMSE of $S$ is about 36% lower than that of $T$. We also note the spike in error for $T$ when $f_C$ approaches 0.6, which we also observed for the median task (Figure 4.2b); again, this appears to be due to the fact that the input to $T$ is only about 3% of the total records in that case. Since $M$ does not take the
parameters $\epsilon_M$ or $f_C$, we chose to focus on the other mechanisms in the respective plots. We note that the RMSE for $M$ was $> 0.77$ and generally unstable, due to the high noise variance associated with a privacy setting of 0.01.

Finally, Figure 4.3d shows that the size of the dataset $n$ has a big impact on all approaches, as expected. The larger the dataset, the more data that is left after sampling and the better the PDP approaches perform. Moreover, since the sensitivity of linear regression is independent of $n$, the signal to noise ratio improves with increasing input size. In Fig. 4.3d, the RMSE of $M$ remained above 1.4 until $n \geq 40,000$.

### 4.6 Summary

We have introduced PDP, a personalized privacy framework that combines the strength of differential privacy with the added flexibility of user-specific privacy guarantees. Mechanisms based on non-uniform sampling and extensions of the exponential mechanism can achieve PDP effectively and efficiently. There are many avenues for future work. With respect to the Sampling mechanism, although we have shown that simple threshold heuristics work well in practice, the error could likely be further reduced through a more careful tuning of the threshold for specific tasks. As with differential privacy, although the exponential mechanism is quite general, getting the best results requires a careful choice of quality function, and the use of the seemingly “obvious” quality function can be beaten by tailored approaches in terms of accuracy and scalability. It will also be of interest to extend notions of personalized privacy to social networks, where the individuals are nodes, and edges represent connections between pairs.
Chapter 5

Publishing Attributed Social Graphs with Formal Privacy Guarantees

5.1 Introduction

Social network analysis (SNA) is an important tool with diverse applications, from marketing to counter-terrorism to the prediction of disease outbreaks. Recent years have seen exponential growth in online social networking (e.g., Facebook, Google+), which has created unique opportunities for SNA at unprecedented scales. Although the vast stores of social data hold great potential for research, they also present significant concerns for privacy. Social networks encode complex relationships among individuals (e.g., friendships, acquaintances, sexual relationships, disease transmission), which may be sensitive. Moreover, the nodes (e.g., users) in real-world social networks may be associated with various sensitive attributes, such as age, location or sexual preference. In order for the full benefit of SNA to be realized, effective privacy-preserving analysis techniques for this type of data are critical.

Much of the prior research on privacy in the context of SNA has focused on graph anonymization techniques (e.g., [78, 141, 143]). Unfortunately, such approaches are now known to be vulnerable to deanonymization attacks (e.g., [5, 95]). Consequently, the past few years have seen
increasing interest around the idea of extending the rigorous privacy guarantees of differential privacy to graph analysis. Research in this area has generally followed one of two directions: (1) methods for privately computing specific statistics over sensitive graphs, such as degree distribution [44], subgraph counting [58, 10], clustering coefficient [126], and frequent subgraph mining [113]; and (2) private graph publication, which typically involves privately fitting a generative graph model to an input graph and then sampling a representative synthetic graph, which can be used in analyses as a proxy for a real input graph (e.g., [91, 110, 125, 19, 129, 80, 103]). Our contributions in this chapter are to that latter line of work. Specifically, we address the important challenge of privately modeling social graphs in which the vertices have sensitive attributes that are correlated with the structure of the graph. Our goal then is to be able to generate synthetic graphs that mimic an input graph in terms of these important characteristics and that can be published without compromising the privacy of the individuals and relationships described in the input graph.

The motivation for private graph learning and synthesis, in general, is manifold. Three important motivations were highlighted in [43]: (1) first, some analyses require running complex algorithms or simulations over actual network data, rather than computing statistics; using synthetic graphs in place of the true input allows an analyst to protect privacy without the need to determine the global sensitivity of the algorithm, which can be highly non-trivial in practice; (2) synthetic graphs facilitate “exploratory, open-ended, and iterative” analyses, which can be very difficult to accommodate directly on an input graph with limited privacy budget under differential privacy; and (3) it eliminates the need to share the details of proprietary analyses with the data owner.

Existing work related to differentially private graph learning and synthesis has focused on modeling network structure alone, without taking into account vertex attributes and their correlations with graph structure. However, real-world graphs that have vertex attributes and exhibit such correlations are common in practice. For example, in social graphs, user nodes often have many attributes such as age, gender, sexual preference, etc. and are well known to
exhibit homophily, or the tendency for nodes with similar attributes to form connections [85]. Existing differentially private graph models are unable to capture such properties.

5.1.1 Contributions

In this work, we develop a differentially private framework for synthesizing attributed social graphs that mimic the structure and attribute correlations observed in an input graph. Our solution prevents the disclosure of individual social relationships (edges) and node attribute values associated with the individuals in the input graph. To the best of our knowledge, this is the first differentially private approach that targets both structural properties and attribute correlations.

To that end, we adapted the recent Attributed Graph Model (AGM) of Pfeiffer et al. [101] to support strong differential privacy guarantees while preserving the utility of the synthesized graphs. AGM models a social network using three sets of parameters that describe (1) the distribution of attributes over the nodes, (2) the correlations between node attributes and edges, and (3) the modeling parameters for an underlying generative structural model. We show how to effectively compute each of these model parameters under differential privacy. While satisfying differential privacy for the attribute distribution is relatively straightforward, the other two sets of parameters present a significant challenge. Modeling the attribute correlations involves computing the fraction of edges that exist between nodes with every possible attribute configuration. Changing the attribute values associated with a single node can change many of these values, depending on the degree of the node. This translates to a high global sensitivity and standard techniques, such as the Laplace mechanism, will destroy utility. We explore three different techniques to overcome this problem: (1) smooth sensitivity, (2) an adaptation of the sample and aggregate framework [98] to the graph analysis domain, and (3) edge truncation.

To capture the structural properties of the input graph, AGM incorporates an underlying generative structural model. Although several prior works have focused on extending existing structural models to satisfy differential privacy, none of these are suitable for modeling large
social networks, which are characterized by heavy tailed degree sequences and large amounts of clustering. Moreover, we found that there is a lack of existing models that both capture these important structural characteristics and that are amenable to differential privacy. Therefore, we propose a new structural model, called TriCycLe, that captures the degree distribution and clustering coefficients of a social network, and is parameterized by statistics for which good differentially private estimators are known. Specifically, TriCycLe extends the simple Chung-Lu random graph model with a triangle construction phase, in which existing edges in the Chung-Lu graph are iteratively rewired to create the same number of triangles observed in the input graph. The model parameters are thus the triangle count and the degree sequence of the input graph, both of which can be accurately estimated under differential privacy. We show how to integrate this new model into AGM and outline the end-to-end workflow for generating differentially private synthetic graphs.

We evaluate the efficacy of the proposed framework on four real-world social network datasets and demonstrate that it is able to produce synthetic graphs that accurately mimic the structure and attribute correlations of an input graph, while also ensuring a strong level of privacy.

5.1.2 Organization

The remainder of this chapter is organized as follows. In the next section, we describe our problem and setting in more detail and introduce notation; we also provide an overview of AGM, the attributed graph model that we adapt to satisfy differential privacy. In Section 5.3 we present our approach for computing differentially private estimates of the three sets of modeling parameters used by AGM. Then, in Section 5.4 we put the pieces together and outline the end-to-end workflow for synthesizing differentially private attributed graphs. Section 5.5 presents our experimental study on four real-world datasets. Finally, we discuss related work in Section 5.6 and conclude with a summary of our approach and a discussion of future work in Section 5.7.
5.2 Preliminaries

5.2.1 Setting and Notation

We represent a social network (social graph) as an attributed simple graph (i.e., undirected, unweighted edges with no multiple or self-edges) denoted by the triple $G = (N, E, X)$. We use the terms network and graph interchangeably in the rest of the chapter. $N = \{v_1, \ldots, v_n\}$ is a set of vertices (nodes), where each node typically represents a person or user. $E = \{e_1, \ldots, e_m\}$ is a set of edges, where we write $e_{ij}$ to refer to an undirected edge between nodes $v_i$ and $v_j$ (i.e., $e_{ij} \equiv e_{ji}$ and $e_{ij} \in E$ iff $e_{ji} \in E$). We write $Nbr(v_i)$ to mean the set of $v_i$’s neighbors, i.e., $Nbr(v_i) = \{v_j \in N | e_{ij} \in E\}$, and we use $d_i = |Nbr(v_i)|$ to denote the degree of $v_i$.

$X = \{x_1, \ldots, x_n\}$ is a set of $w$-dimensional attribute vectors, where the vector $x_i = \langle x_{i1}, \ldots, x_{iw} \rangle$ contains the attributes associated with node $v_i$. In the rest of the chapter, we assume that the attributes are binary, i.e., $x_i \in \{0, 1\}^w$; however, the ideas could be extended to attributes with larger domains.

Table 5.1 summarizes the notation introduced above, as well as other notation frequently used throughout this chapter. We will sometimes augment the notation with a superscript to explicitly connect the notation with a specific graph, e.g., $N^{(G_1)}$ refers to the set of nodes of graph $G_1$ and $d_i^{(G_1)}$ to the degree of node $v_i \in N^{(G_1)}$.

5.2.2 Attributed Graph Model (AGM)

In this work, we choose to extend the recent Attributed Graph Model (AGM) of Pfeiffer et al.[101] to enable the generation of synthetic graphs with formal differential privacy guarantees. AGM is capable of capturing both the structure and attribute correlations of a given input graph and efficiently sampling synthetic graphs from the learned model.

We first define two functions that will be used in the explanation of AGM and throughout the rest of the chapter. These functions simply map a node attribute vector or an undirected edge to a unique integer based on the node’s attribute values or the combination of attribute values.
Table 5.1 Summary of the notation used in Chapter 5

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Set of nodes, ${v_i}$</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of nodes, i.e., $</td>
</tr>
<tr>
<td>Nbr($v_i$)</td>
<td>Set of nodes incident to node $v_i$</td>
</tr>
<tr>
<td>$d_i$</td>
<td>Degree of node $v_i$</td>
</tr>
<tr>
<td>$d_{\text{max}}$</td>
<td>Maximum degree among nodes in a graph</td>
</tr>
<tr>
<td>$E$</td>
<td>Set of edges, ${e_{ij}}$</td>
</tr>
<tr>
<td>$e_{ij}$</td>
<td>An (undirected) edge between $v_i$ and $v_j$</td>
</tr>
<tr>
<td>$E_{ij}$</td>
<td>Binary random variable, $E_{ij} = 1$ if $e_{ij} \in E$</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of edges, i.e., $</td>
</tr>
<tr>
<td>$X$</td>
<td>Set of attribute vectors, ${x_i}$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>Attribute vector associated with node $v_i$</td>
</tr>
<tr>
<td>$x_{ij}$</td>
<td>The $j$th attribute in $x_i \in X$</td>
</tr>
<tr>
<td>$w$</td>
<td>Number of attributes</td>
</tr>
<tr>
<td>$Y_w$</td>
<td>Set of elements representing the possible node attribute configurations</td>
</tr>
<tr>
<td>$f_w$</td>
<td>Function that maps an $x_i$ to an element of $Y_w$</td>
</tr>
<tr>
<td>$Y_w^F$</td>
<td>Set of elements representing the possible edge attribute configurations</td>
</tr>
<tr>
<td>$F_w$</td>
<td>Function that maps an $e_{ij}$ to an element of $Y_w^F$</td>
</tr>
<tr>
<td>$\Theta_X$</td>
<td>Set of attribute parameters used by AGM</td>
</tr>
<tr>
<td>$\Theta_F$</td>
<td>Set of attribute correlation parameters used by AGM</td>
</tr>
<tr>
<td>$M$</td>
<td>Underlying structural model used by AGM</td>
</tr>
<tr>
<td>$\Theta_M$</td>
<td>Set of structural model parameters</td>
</tr>
<tr>
<td>$k$</td>
<td>Truncation parameter</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Privacy parameter</td>
</tr>
<tr>
<td>$n_{\Delta}$</td>
<td>Number of triangles in a graph</td>
</tr>
<tr>
<td>$S$</td>
<td>Unordered degree sequence of a graph</td>
</tr>
</tbody>
</table>
associated with the edge’s incident nodes, respectively. The first function \( f_w : X \rightarrow Y_w \) maps a \( w \)-dimensional attribute vector \( x_i \) to an element of the finite set \( Y_w \), such that two attribute vectors map to the same value if and only if they are equivalent in every coordinate\(^1\). With binary attributes, we have \( |Y_w| = 2^w \). We can think of \( f_w \) as mapping the binary string formed from the attribute values to the corresponding decimal value. For example, with \( w = 2 \), we would have \( f_w((0,0)) = 0, f_w((0,1)) = 1, f_w((1,0)) = 2, \) and \( f_w((1,1)) = 3 \), with \( Y_w = \{0,1,2,3\} \). The second function \( F_w : X \times X \rightarrow Y_w^F \) maps a pair of \( w \)-dimensional attribute vectors associated with an edge to a unique element of \( Y_w^F \), ignoring the direction of the edge\(^2\). For binary attributes and undirected edges, we would have \( |Y_w^F| = \frac{(2^w+1)}{2} \). As an example, with \( w = 1 \), we have \( F_w((0),(0)) = 0, F_w((0),(1)) = F_w((1),(0)) = 1, \) and \( F_w((1),(1)) = 2 \), with \( Y_w^F = \{0,1,2\} \).

AGM models an input graph \( G \) using three sets of parameters, \( \Theta_X, \Theta_M, \) and \( \Theta_F \), which are described below.

1. A set of attribute parameters \( \Theta_X \) is used for modeling the distribution of attributes on the vertices, i.e., \( P(X|\Theta_X) \). \( \Theta_X \) can be computed from graph \( G \) as: \( \Theta_X(y_i) = \frac{\sum_{v \in V}[f_w(x_i) = y_i]}{n} \), \( \forall y_i \in Y_w \), where \([\cdot] = 1 \) if the statement within the brackets evaluates to true, and \( 0 \) otherwise. In other words, \( \Theta_X(y_i) \) is the fraction of total nodes whose attribute vectors evaluate to \( y_i \).

2. A set of edge parameters \( \Theta_M \) models the distribution of edges, \( P(E|\Theta_M) \). This is accomplished using an appropriate underlying generative structural model, \( M \), parameterized by \( \Theta_M \). In general, \( M \) can be any generative model in which a graph can be generated by repeatedly sampling edges from the conditional distribution \( P(E_{ij}|\Theta_M, M) \), where \( E_{ij} \in \{0,1\} \) denotes a binary random variable such that \( E_{ij} = 1 \) means \( e_{ij} \in E \). Note that graphs generated by \( M \) are independent of \( X \) (i.e., the node attribute vectors).

3. \( \Theta_F \) is the set of parameters for modeling the correlations between attributes and edges,

\(^1\)Formally, \( \forall x_i, x_j [f_w(x_i) = f_w(x_j)] \Leftrightarrow [x_{i1} = x_{j1} \land \ldots \land x_{iw} = x_{jw}] \).

\(^2\)Formally, \( F_w \) maps a pair of attribute vectors \( x_i, x_j \) to an element of \( Y_w^F \), such that \( \forall x_i, x_j, x_k [F_w(x_i, x_j) = F_w(x_k, x_i)] \Leftrightarrow [(f_w(x_i) = f_w(x_k)) \land (f_w(x_j) = f_w(x_i)) \lor (f_w(x_i) = f_w(x_i)) \land (f_w(x_j) = f_w(x_k))] \).
\[ P(F_w(x_i, x_j)|E_{ij} = 1, \Theta_M, \Theta_F). \]

We can construct \( \Theta_F \) from \( G \) as: 
\[
\Theta_F(y_j) = \frac{\sum_{e_{ij} \in E} F_w(x_i, x_j) = y_j}{m},
\]

\( \forall y_j \in Y_w^F \). In other words, \( \Theta_F(y_j) \) is the fraction of all edges that connect two nodes such that the associated pair of node attribute vectors evaluates to \( y_j \).

---

**Algorithm 2 SampleFromAGM (Algorithm 2 from [100])**

**Input:** \( M, \Theta_M, \Theta_X, \Theta_F, m, n, w \)

**Output:** \( G' = (N', E', X') \)

1: \( N' = N \)
2: Sample \( X' \) from \( P(X|\Theta_X) \) using \( \Theta_X \)
3: Create proposing distribution \( Q'_M \) from model \( M \) and parameters \( \Theta_M \)
4: Sample temporary edge set \( E' \) from \( M \) using \( Q'_M \)
5: // Compute the acceptance probabilities
6: Compute \( \Theta'_F \) from \( E' \) and \( X' \)
7: for \( y_i \in Y_w^F \) do
8: \( R(y_i) \leftarrow \frac{\Theta_F(y_i)}{\Theta_F'(y_i)} \)
9: for \( y_i \in Y_w^F \) do
10: \( A(y_i) \leftarrow \frac{R(y_i)}{\sup \{R(y)\}} \)
11: // Reinitialize \( E' \) and sample new edge set based on \( A \) and \( X' \)
12: \( E' = \emptyset \)
13: while \( |E'| < |E| \) do
14: \( e'_{ij} \leftarrow \text{SAMPLEMULTINOMIAL}(Q'_M) \)
15: \( u \leftarrow \text{SAMPLEUNIFORM}(0, 1) \)
16: if \( u \leq A(F_w(x'_i, x'_j)) \) then
17: \( E' \leftarrow E' \cup e'_{ij} \)
18: end while
19: return \( G' = (N', E', X') \)

Given the three sets of parameters, we can generate a synthetic graph using the procedure in Algorithm 2. In line 1, we simply let \( N' = N \) since the output graph will contain the same number of nodes as the input graph\(^3\). We sample a new set of attribute vectors \( X' \) for nodes \( N \) by repeatedly taking independent samples from \( P(X|\Theta_X) \). Then, the goal is to sample a synthetic graph \( G' \) by sampling independent edges from \( P_0(E_{ij} = 1, F_w(x'_i, x'_j), \Theta_M, \Theta_F) \). However, since it is not possible to efficiently sample from \( P_0 \) directly, AGM samples from it indirectly, using a form of Accept/Reject sampling, as shown in Algorithm 2. In lines 1–3, a new set of attribute

\(^3\) \( N \) is simply a list of arbitrary, insensitive node indices.
vectors for the nodes is sampled from $P(X|\Theta_X)$ using the attribute probabilities defined by $\Theta_X$ and then a proposing distribution $Q'_M(v'_i, v'_j)$ is created from model $M$ with parameters $\Theta_M$ learned from the input graph. That is, $Q'_M(v'_i, v'_j) = \frac{P_M(E_{ij}=1|\Theta_M)}{\sum_{k,l} P_M(E_{kl}=1|\Theta_M)}$.

The idea underlying lines 4–12 is to use the (randomly produced) attribute correlations observed in the temporary graph formed from $E'$ and $X'$, along with those observed in the input graph to derive acceptance probabilities $A$, such that edge-attribute configurations that are under-sampled by $M$ are given a higher probability of being accepted into the final graph, while those that $M$ over-samples are given a lower probability of acceptance. We then discard the temporary $E'$ and generate the new output graph by repeatedly sampling edges from $Q'_M$, accepting or rejecting an edge based on the attributes of its incident nodes and the acceptance probabilities $A$ (lines 12–17). The resulting graph approximates a joint sampling of the structural model and the attribute correlations.

### 5.2.3 Privacy Model

Differential privacy was originally defined in the context of tabular data, where the privacy guarantee depends on the notion of neighboring databases [26, 31]. Traditionally, two databases are considered neighboring if they differ in a single record. Translating differential privacy to the graph analysis domain requires defining what it means for two graphs to be neighboring. Most of the prior work in this area considers two graphs to be neighboring if they differ in the presence of a single edge; that is, given a graph $G$, the neighboring graph $G'$ is formed by adding (or deleting) any one edge to (from) $G$. This is commonly referred to as edge-differential privacy [44]. This definition protects individual relationships from disclosure—an adversary will not be able to determine with high probability whether an edge exists between any pair of nodes (in the input graph).

Unlike most of the prior work (with the exception of [10]), we specifically focus on graphs that have one or more attributes associated with each node. The notion of edge privacy stated above is only sufficient to prevent disclosure of edges; however, node attributes are typically
highly sensitive as well, and should be protected. Thus, we must alter the neighboring graph definition so that we can account for their impact when determining the sensitivity of the analysis task. We adopt the following notion for neighboring graphs, which to our knowledge first appeared in [10].

**Definition 5.2.1** (Edge-Adjacent Attributed Graphs). Two attributed graphs $G, G'$ are said to be edge-adjacent (or neighboring) if they differ in the presence of a single edge or in the attribute vector associated with a single node.

The principal challenge to applying differential privacy to graph data is that the sensitivities of many graph analyses are proportional to $n$, since the impact of a node often depends on its degree, which is bounded by $n - 1$. For example, in the subgraph counting task the goal is to count the number of subgraphs of a distinct shape (e.g., triangles) contained in a graph; in the worst case, it is possible to have a graph in which the addition of a single edge could change the triangle count from zero to $n - 2$, as illustrated in Fig. 5.1.

### 5.3 Privately Computing the Model Parameters

In this section, we extend AGM to enforce differential privacy for the synthesized graphs. To do so, we develop procedures for learning differentially private approximations for each of the three model parameters, based on the sensitive input graph. Having approximated the model...
parameters in a differentially private way, we can then use them with the sampling algorithm (Algorithm 2) to obtain a synthetic graph that closely approximates the input graph without loss of privacy.

5.3.1 Attribute-Edge Correlations

Recall that the attribute correlation parameters $\Theta_F$ describe the correlations observed between edges and node attributes in the input graph. Such correlations are a key characteristic of social graphs, which are well-known to exhibit phenomena such as homophily$^4$.

To compute $\Theta_F$, we need to compute the connection probabilities $\Theta_F(y_j) = \frac{\sum_{e_{ij} \in E} [F_w(x_i, x_j) = y_j]}{m}$, for each $y_i \in Y^F_w$. Since we are assuming undirected graphs with $w$ node attributes, we need to compute $y = \left(\frac{2^w+1}{2}\right)$ connection probabilities.

We can view the problem as computing a set of counting queries $Q_F = \{q_1, \ldots, q_y\}$ that count the number of edges between nodes with a particular attribute configuration $y_i$, i.e., $q_i = \sum_{e_{jk} \in E} [F_w(x_j, x_k) = y_i]$. If we can compute the counts privately, then dividing by their sum will give us the required probabilities. The challenge here is that, while adding or removing a single edge will only increase or decrease one count by at most one, changing the attributes of a single node could have a significant impact on one or more counts (depending on the degree of the node). In terms of differential privacy, this translates to a large global sensitivity. In fact, a naïve attempt to satisfy differential privacy through a direct application of the Laplace mechanism (Theorem 2.1.1), e.g. by computing $Q_F$ and adding independent Laplace noise to each $q_i$, would completely destroy the utility of the value. To understand why, imagine an input graph containing a degree $n-1$ hub node; changing the attributes of this hub node will cause some subset of the counts to decrease by a total of $n-1$ and another subset to increase by the same amount, which means the global sensitivity is $2n-2$. Therefore, achieving differential privacy with good utility in this setting will require either finding a way to exploit the fact

---

$^4$Homophily refers to the tendency for similar individuals to associate with one another, as expressed in the idiom *birds of a feather flock together*. Homophily can influence information flow in social networks and can have important implications for policy decisions [22]. A variety of different indices for measuring homophily have been proposed in the literature (e.g., [14, 22, 85]).
that such worst case nodes are extremely rare in real-world networks, or somehow limiting the worst-case impact of a single node by way of a transformation applied to the input graph. We explore several approaches that are based on these ideas.

The first two approaches that we describe below produce a private approximation of the set of connection counts $Q_{F}$ (i.e., noisy counts). Given these noisy counts, we can just divide each count by the sum of the counts to get an approximation of the probability distribution $\Theta_{F}$. The third approach produces an approximation of probability distribution directly.

5.3.1.1 Direct Approach with Smooth Sensitivity

In real-world social graphs, most nodes will only share edges with a small fraction of the other nodes in the graph. Smooth sensitivity, proposed by Nissim et al. [98], offers a general approach for getting around a high global sensitivity that is due to unlikely inputs. The idea is to add noise according to a smooth upper bound on the local sensitivity, which is based on the actual input rather than an improbable worst-case input. There is no automatic procedure for determining the smooth sensitivity of a computation, so we must derive it manually for our specific task.

We begin by observing that the local sensitivity $f_{Q_{F}}$ (that is, the function that computes the set $Q_{F}$) is twice the maximum node degree.

**Lemma 5.3.1.** The local sensitivity of $f_{Q_{F}}$ is $LS_{Q_{F}}(G) = 2d_{\text{max}}$, where $d_{\text{max}} = \max_{v_{i} \in V(G)} d_{v_{i}}$.

**Proof.** The greatest impact to $Q_{F}$ comes from changing the attribute vector associated with the highest degree node, which would have the effect of reducing some subset of the counts by an amount equal to $d_{\text{max}}$, and increase another subset of the counts by the same amount. The impact of adding or removing an edge is just an increasing or decrease of one to a single count, which is negligible by comparison. \qed

Recall that adding noise calibrated to the local sensitivity does not satisfy differential privacy, because the magnitude of the noise may itself leak information about the underlying
data. As explained in Section 2.1.1, the smooth sensitivity can be expressed in terms of the local sensitivity at distance $t$ from the input graph.

**Proposition 5.3.2.** The $\beta$-smooth sensitivity of $f_{Q_F}$ is

$$S^{*}_{Q_F,\beta}(G) = \max_{t \geq 0} e^{-t\beta} LS^t_{Q_F}(G)$$

$$= \max_{t \geq 0} e^{-t\beta} \max_{G' : d(G, G') \leq t} LS_{Q_F}(G')$$

$$= \max_{t \geq 0} e^{-t\beta} \min(2d_{\text{max}} + 2t, 2n - 2)$$

where, $d(G, G')$ is fewest number of edges that would need to be added/removed or the fewest number of nodes whose attributes would need to be changed to transform $G$ into $G'$.

For typical graphs and typical settings of $\epsilon$, the $\beta$-smooth sensitivity will be $2d_{\text{max}}$ (for negligible $\delta \approx 0$). Setting the derivative of the right hand side to zero and solving, we arrive at the following corollaries.

**Corollary 5.3.3.** For an input graph $G$ with maximum degree $d_{\text{max}},$

$$S^{*}_{Q_F,\beta}(G) = \begin{cases} 
2d_{\text{max}} & \text{if } \frac{1}{\beta} \leq 2d_{\text{max}} \\
2e^{(\beta d_{\text{max}} - 1) / \beta} & \text{otherwise}
\end{cases}$$

**Corollary 5.3.4.** For an input graph $G$ with maximum degree $d_{\text{max}},$ $\beta = \frac{\epsilon}{2 \ln(2/\delta)},$ and fixed $\epsilon,$ we have $S^{*}_{Q_F,\beta}(G) = 2d_{\text{max}},$ for any $\delta \geq 2e^{-(\epsilon d_{\text{max}})/2}.$

For many real-world social networks, we can expect the maximum degree to be a small fraction of the total number of nodes, and thus using smooth sensitivity should work reasonably well.
5.3.1.2 Edge Truncation

We now explore an alternative approach in which we essentially force a smaller global sensitivity by truncating high-degree nodes. The general idea of projecting an input graph of arbitrary degree onto the set of $k$-bounded graphs—that is, the set of graphs with degree at most $k$—was recently proposed in two independent works [59, 10]. Specifically, [10] proposed a general edge truncation operation for transforming graphs in the edge adjacency model, which we extended to our task of computing $Q_F$.

**Definition 5.3.1 (Edge Truncation [10]).** Given a graph $G$ and a truncation parameter $k > 0$, the truncation algorithm $\mu(G, k)$ starts by fixing a canonical ordering over all of the edges in $E$. Then, iterating through each edge $e_{ij} \in E$ in order, an edge is deleted if and only if $d_i > k$ or $d_j > k$.

The notion of restricted sensitivity, introduced in [10], restricts the global sensitivity to consider just the set $H_k$ of graphs with degree at most $k$. An important subtlety is that transformations applied to the input graph, such as $\mu$ from definition 5.3.1, may amplify the restricted sensitivity of a function that is executed on the resulting truncated graph. For a function $f$ with restricted sensitivity $RS_f(H_k)$, it is shown in [10] that the algorithm that applies the truncation operation from definition 5.3.1 to the input graph and then runs $f$ on the truncated graph, i.e., $A_f(G, k) = f(\mu(G, k))$, has a global sensitivity of $3 \cdot RS_f(H_k)$. The factor of 3 is because adding or removing a single edge, prior to truncation, can translate to a difference of up to three edges in the truncated graph [10].

Contrary to the general results in [10], we show that for the special case of $Q_F$, it turns out that we get the truncation operation “for free”, resulting in the global sensitivity being equal to the restricted sensitivity, which is $2k$. The intuition is that, although the truncation routine can amplify the effects of an edge addition/deletion in the input graph, the global sensitivity is still dominated by the impact of changing an attribute vector, which is not affected by the truncation routine.
Proposition 5.3.5. Let $\mu$ be the edge truncation algorithm defined in definition 5.3.1 and let $k > 1$. The global sensitivity $\text{GS}(A_{\mu,Q_F}(G,k))$ of the algorithm $A_{\mu,Q_F}(G,k) = f_{Q_F}(\mu(G,k))$, which truncates an input graph $G$ and then computes the $Q_F$ over the resulting $k$-bounded graph, is $2k$.

Proof. By definition 5.2.1, neighboring graphs may differ in the presence of a single edge or in the attributes associated with one node. Let $v_i, v_j$ be two nodes in an input graph $G$ such that $d_i = d_j = k$ and that no edge exists between $v_i$ and $v_j$. Let $G'$ be the neighboring graph formed by adding edge $e_{ij}$ to $G$. Let $O = e_0, e_1, \ldots, e_m$ be the canonical ordering of the edges in $G$ and let $O' = O + e_{ij}$ be the ordering for $G'$. Finally, assume that $e_0 = e_{rs}$ and $e_1 = e_{uv}$ such that $s \neq u, v \neq r$, $d_s \leq k$, $d_v \leq k$ and that $f_w(x_s) \neq f_w(x_v)$ (i.e., their attribute values are different). Observe that the graph resulting from applying the edge truncation routine, $\mu(G,k)$, to the input graph $G$, will contain edges $e_0, e_1$ since the nodes at both end points have degree $\leq k$, by our assumption. On the other hand, applying the truncation routine to $G'$ will result in both $e_0$ and $e_1$ being deleted, since at the time they are processed, both $v_r$ and $v_u$ have degree $k + 1$ in $G'$. Moreover, by the time edge $e_{ij}$ is considered, both $u$ and $v$ will have degree $\leq k$, so $e_{ij}$ will not be deleted. Also observe that all other edges that are deleted from $G$ are also deleted from $G'$ and vice versa. Consequently, two of the connection counts will decrease by a total of two, while one count will increase by one. Therefore the outputs of $\mu(G,k)$ and $\mu(G',k)$ will differ in exactly three edges.

Now let us consider the case in which $G$ and $G'$ are size $n$ graphs and are exactly the same except that the attribute vector associated with one node, say $v_i$, is changed from $x_i$ to $x'_i$, such that $f_w(x_i) \neq f_w(x'_i)$, in $G'$. Assume that $d_i \geq k$. First observe that the difference in the attribute vectors has no impact on the edges that are deleted by $\mu$, since $\mu$ only looks at the degrees, and after truncation, $v_i$ will have degree $k$ in both graphs. Since the attribute vector associated with $v_i$ is changed in $G'$, the $k$ edges incident to $v_i$ in $\mu(G',k)$ will now contribute to a completely different subset of the counts than the edges incident to $u$ in $\mu(G,k)$; that is, one subset of the counts will increase by $k$ and another disjoint subset will decrease by $k$, for a
total difference of $2k$. Finally, since $2k > 3$ (for $k > 1$), the impact of changing the attributes of a single node dominates that of adding/removing an edge and we have $GS(A_{\mu, Q_F}) = 2k$.

Compared to the smooth sensitivity approach, we could expect this approach to perform better when $k$ is close to the true maximum degree, since using smooth sensitivity with the Laplace mechanism requires using $\epsilon/2$, whereas we can use the full $\epsilon$ with this approach. However, the true maximum degree is a private quantity and cannot be used directly to set $k$. Moreover, for many social networks, which tend to have a power law degree distribution, the average degree may be significantly smaller than the maximum degree. Therefore, setting $k \ll d_{max}$, such that a few of the very high degree nodes are truncated while most nodes are untouched may give better performance due to the reduced sensitivity.

5.3.1.3 SA-Based Approach

In the context of traditional databases, the sample and aggregate framework (SA) was originally proposed by Nissim et al. [98] as a general procedure for satisfying differential privacy in situations where determining the global (or smooth) sensitivity of a function is difficult or inefficient. The idea is as follows: to compute a differentially private estimate of a function $f$ on dataset $D$, where $n = |D|$, one first partitions $D$ into $t = n/k$ smaller datasets of $k$ records each (for some $k$), denoted $D_1, \ldots, D_t$. Then, $f$ is computed on each of the $t$ smaller datasets, yielding the set $Z = \{z_1, \ldots, z_t\}$, where $z_i = f(D_i)$. Finally, $f(D)$ is estimated by applying an aggregation function $Agg$ (e.g., mean) to the set $Z$ and adding noise according to the global (or smooth) sensitivity of $Agg$. For example, if the range of $f$ is $[0, 1]$ and $A$ is the mean function, then we could add noise with sensitivity $1/t$ to the output of $Agg(Z)$.

We can extend this idea to the graph analysis domain to get around the high global sensitivity of $\Theta_F$, as follows. For a given input graph $G$ with $n$ nodes, we first randomly partition the nodes into $t = n/k$ disjoint groups of $k$ nodes each. Let $g_1, \ldots, g_m$ be the set of induced subgraphs, such that $g_i$ is the subgraph induced by the nodes in group $i$. Note that (a) partitioning the nodes randomly does not impact the sensitivity, and (b) by working with the induced
subgraphs, we ensure that a change to one node in the input graph $G$ will only impact a single subgraph. Thus, we can apply a function $f$ to each subgraph, aggregate the individual answers and add noise.

To compute $\Theta_F$, we would compute the $\left(\frac{2^w+1}{2}\right)$ connection probabilities (not the counts) for each subgraph, average the corresponding probabilities across all subgraphs and add Laplace noise to each average. This approach has a global sensitivity of $\frac{2}{t}$. To understand why, consider a pair of neighboring graphs $G, G'$, where $v_i$ is the node whose attributes were changed to get $G'$ from $G$, and observe that, in the worst case, $v_i$ is a member of a subgraph in which all of the internal edges are incident to $v_i$; if $v_i$'s attribute values are different in $G$ and $G'$, then the $L_1$ difference of the corresponding sets of probabilities from $G$ and $G'$ will be 2. Since we are averaging over $t$ subgraphs, the total impact on the set of average probabilities is $\frac{2}{t}$. Finally, after adding noise to the probabilities, we must divide them by their sum to ensure that they add up to one.

This approach introduces two types of error into the resulting probabilities: estimation error, due to averaging over outputs from the subgraphs, and perturbation error due to the Laplace noise. The accuracy of this approach will depend on balancing the two sources of error by selecting an appropriate size $k$ for the subgraphs. A larger $k$ gives fewer but bigger subgraphs, which reduces estimation error at the cost of increasing the perturbation error; conversely, a smaller $k$ gives a greater number of small subgraphs, which reduces perturbation error at the cost of more estimation error.

5.3.1.4 Comparison of Approaches

Figures 5.2(a)–5.2(d) compare the mean absolute errors$^5$ (MAE) of the three proposed approaches for different privacy settings, $\epsilon$, on four datasets (the Last.fm, Petster, Epinions and Pokec datasets are described later in Section 5.5.1). The input graphs contained $w = 2$ attributes. The error-rates are averages over over 10,000 trials (1,000 for the larger Epinions

$^5$That is, the mean error between the privately computed probabilities and the true probabilities computed directly from the input graph.
and Pokec datasets). In the plots, Smooth denotes the smooth sensitivity-based approach, EdgeTrunc is the edge truncation approach, and S&A is the sample and aggregate-based approach. The truncation parameter for EdgeTrunc and the group-size parameter for S&A were tuned empirically to minimize the MAE. Note that although tuning based on the input data could leak privacy in practice, it is useful for comparing the various approaches here. For the smooth sensitivity approach, we used negligible $\delta = 10^{-6}$.

As a baseline, we included results for the naïve approach (the dashed line in the figures) that simply adds noise from $\text{Lap}(\frac{2n-2}{\epsilon})$ to each connection count and then normalizes by the
sum to get $\Theta_F$. To be considered useful, an approach should have an MAE well below this line. The main takeaways from these results are that (1) all of the approaches work reasonably well at the weakest privacy setting tested ($\epsilon = 1$), but edge truncation appears to be the best choice across all datasets and privacy settings tested; (2) in general, the larger the input graph, the lower the error. On the large Pokec dataset ($m = 3,725,424$) all of the approaches perform well, even with the strongest privacy setting tested. For EdgeTrunc, the MAE comes very close to zero, even for $\epsilon = 0.1$.

Given the superior performance of EdgeTrunc, we chose to limit our focus to that approach for privately computing $\Theta_F$ in the remainder of this work. The next section summarizes the use of the truncation approach for computing $\Theta_F$ under differential privacy.

### 5.3.1.5 Algorithm: Using Edge Truncation to Compute Correlations

For completeness, we briefly summarize in Algorithm 3 the procedure for computing a differentially private version of $\Theta_F$, denoted $\tilde{\Theta}_F$, with edge truncation and provide a formal statement of its privacy guarantee (Thm. 5.3.6).

**Algorithm 3 LearnCorrelationsDP**

**Input:** $\epsilon, G, k, w$

**Output:** $\tilde{\Theta}_F$

1. $G' \leftarrow \mu(G, k)$  
   // Truncate G using the truncation operator $\mu$ from Def. 5.3.1
2. $Q_F \leftarrow f_{Q_F}(G')$  
   // Compute the set of counts on truncated graph $G'$
3. for $y_i \in Y^F$
   4. $\tilde{Q}_F(y_i) \leftarrow \min\left(n, \max\left(0, Q_F(y_i) + \text{LAP}(0, \frac{2k}{\epsilon})\right)\right)$  
      // Add Laplace noise to each count and clamp range
5. $q_{\text{sum}} \leftarrow \max_{y_i} \tilde{Q}(y_i)$
6. for $y_i \in Y^F$
   7. $\tilde{\Theta}_F(y_i) \leftarrow \frac{\tilde{Q}_F(y_i)}{q_{\text{sum}}}$
8. return $\tilde{\Theta}_F$
Theorem 5.3.6. Algorithm 3 satisfies $\epsilon$-differential privacy.

Proof. The global sensitivity of truncating $G$ to get $k$-bounded graph $G'$ followed by computing $Q_F$ on $G'$ (lines 1–2) was shown to be $2k$ by Proposition 5.3.5. In line 4, we add independent Laplace noise with mean zero and scale equal to $\frac{2k}{\epsilon}$, which gives $\epsilon$-differential privacy by Theorem 2.1.1 (Laplace mechanism). The remainder of the algorithm, including clamping the noisy counts to the range $(0,n)$, operates only on the noisy counts, and therefore has no impact on the privacy guarantee (by Theorem 2.1.5). Thus, Algorithm 3 satisfies $\epsilon$-differential privacy.

Setting the Truncation Parameter ($k$). In practice, we need to supply the truncation parameter $k$. The best choice for $k$ depends on $\epsilon$ and on the characteristics of the input graph; however, tuning $k$ directly on the input graph would violate differential privacy. Although it may be possible to allocate some portion of the privacy budget for tuning $k$ on the input, we find that using the data-independent heuristic $k = n^{\frac{1}{3}}$ works well in practice, as illustrated by Fig. 5.3. Since real-world social networks tend to have power-law degree distributions—meaning that only a small fraction of the nodes have very high degrees—this choice of $k$ significantly lowers the sensitivity without deleting too many edges. For sufficiently large graphs (e.g., Pokec), the
difference in error rates between the best \( k \) and the heuristically chosen \( k \) is negligible. Note that since \( n \) is assumed to be public in this work, setting \( k \) as a function of \( n \) does not affect the privacy guarantee of Thm. 5.3.6. The authors of [10] made a similar recommendation regarding the setting of \( k \).

5.3.2 Node Attribute Distribution

The distribution of attributes on the nodes is modeled by \( \Theta_X \). Recall that \( Y_w \) is the set representing the \( 2^w \) possible attribute configurations that nodes can take, on a dataset with \( w \) attributes. We compute the distribution from the input graph as:

\[
\Theta_X(y_i) = \sum_{v_i \in N} \frac{\overline{f_w(x_i) = y_i}}{n},
\]

for \( y_i \in Y_w \).

As with the previous task of computing the edge correlations, we can view the task of privately computing \( \Theta_X \) as answering a set of \( y = 2^w \) counting queries \( Q_X = \{q_1, \ldots, q_y\} \) (where \( q_i = \sum_{v_j \in N} [f_w(x_j) = y_i] \)) under differential privacy; however, in this case, we are counting disjoint sets of nodes, which is a more straightforward task, due to lower global sensitivity. Observe that changing the attributes of a single node may, in the worst case, reduce at most one count \( q_i \in Q_X \) by one and increase another count \( q_j \) by one, which means the global sensitivity is just two. We are only counting nodes, so adding/removing edges has no impact on the counts. Therefore, we can obtain a private approximation of \( \Theta_X \), denoted by \( \tilde{\Theta}_X \), by answering the counting queries on the input graph \( G \), adding noise from \( \text{Lap}(0, 2/\epsilon) \), and finally dividing the noisy counts by their sum. As with \( \Theta_F \), we clamp the noisy counts to the range \((0, n)\) before dividing by the sum. For completeness, this procedure is specified in Algorithm 4.

**Theorem 5.3.7.** Algorithm 4 satisfies \( \epsilon \)-differential privacy.

**Proof.** Let \( G = (N, E, X) \) be an arbitrary input graph and let \( G' = (N, E, X') \) be the neighboring graph created by changing the attributes of one node, say \( v_i \in N \), from \( x_i \) to \( x_i' \), where \( f_w(x_i) = y_i \) and \( f_w(x_i') = y_j \) where \( y_i, y_j \in Y_w, y_i \neq y_j \). Then we have that \( q'_i = q_i - 1 \) and \( q'_j = q_j + 1 \), and \( q'_k = q_k \), \( \forall k \neq i, k \neq j \). Thus, the global sensitivity of of \( f_{Q_X}(X) \) (line 1) has a global a sensitivity of two. In line 3, we add independent Laplace noise with mean zero and
Algorithm 4 LearnAttributesDP

**Input:** $\epsilon, X, w$

**Output:** $\tilde{\Theta}_X$

1: $Q_X \leftarrow f_{Q_X}(X)$ \hfill // Compute the set of counts from $X$

2: for $y_i \in Y_w$ do

3: $\tilde{Q}_X(y_i) \leftarrow \min\left(n, \max\left(0, Q_X(y_i) + \text{LAP}(0, \frac{2}{\epsilon})\right)\right)$ \hfill // Add Laplace noise to each count and clamp

4: $q_{\text{sum}} \leftarrow \sum_{y_i} \tilde{Q}_X(y_i)$

5: for $y_i \in Y_w$ do

6: $\tilde{\Theta}_X(y_i) \leftarrow \frac{\tilde{Q}_X(y_i)}{q_{\text{sum}}}$ \hfill // Normalize

7: return $\tilde{\Theta}_X$

Scale equal to $\frac{2}{\epsilon}$, which gives $\epsilon$-differential privacy by Thm. 2.1.1 (Laplace mechanism). The remainder of the algorithm, including clamping the noisy counts to the range $(0, n)$, operates only on the noisy counts, and therefore has no impact on the privacy guarantee (by Theorem 2.1.5). Thus, Algorithm 4 satisfies $\epsilon$-differential privacy.

5.3.3 Structural Model: TriCycLe

AGM requires an underlying generative graph model, $M$, to capture the structural properties of the input graph. $M$ is responsible for proposing new edges, which AGM then accepts or rejects to achieve the desired attribute correlations as it constructs a synthetic graph. A nice feature of AGM is that it supports the use of any generative structural model in which a graph can be generated by repeatedly sampling edges. Therefore, our objective in this section is to (1) identify an appropriate structural model, and (2) to devise a way to compute its required model parameters, $\Theta_M$, efficiently and accurately under differential privacy.

For the purposes of this work, the selected model should (1) be able to capture the unique structural features of social networks, including a heavy-tailed degree distribution and high clustering, (2) it should be reasonably scalable (i.e., able to handle hundreds of thousands of nodes and millions of edges), and (3) the model parameters should be amenable to differential privacy.

Some recent works have focused on extending popular structural models to support differ-
ential privacy, including exponential random graph models (ERGMs) [80] and the Kronecker graph model (KGM) [91]; however neither of these models is quite suitable for our purposes. While ERGMs are capable of reproducing the degree distribution and clustering coefficients of an input graph, they do not scale well beyond a few thousand nodes. Conversely, KGMs are more scalable but lack the ability to capture the high clustering that is characteristic of real-world social networks [112, 100].

From the broader graph analysis literature, two models that stand out as possible candidates are the Block Two-Phase Erdős-Rényi (BTER) model [111, 66], and the Transitive Chung-Lu (TCL) model [100]. Both are capable of efficiently generating synthetic graphs that mimic the degree distribution as well as the distribution of clustering coefficients in an input graph. The models both take as input the degree sequence measured from the input graph, as well as another parameter that captures the clustering in the input graph. The two models differ significantly in how they capture the clustering of the input graph and in how the synthetic graph is generated. At a high level, BTER models a graph as a collection of dense Erdős-Rényi (ER) subgraphs that are sparsely interconnected to form the global graph structure and aims to reproduce the degree-wise clustering coefficients, i.e., \( C_D = \{c_i\} \), where \( c_i \) is the ratio of the number of triangles involving nodes of degree \( i \) over the number of length-two paths centered at nodes of degree \( i \). To do that, it requires \( C_D \), along with the degree sequence \( S \), to be measured from the input graph and supplied as input to the algorithm. TCL, on the other hand, aims to reproduce the distribution of local clustering coefficients observed in the input graph. It extends the well-known Chung-Lu (CL) random graph model [21] by adding a transitive closure parameter, \( \rho \), which controls the probability by which new edges create triangles (by connecting a node to a random two-hop neighbor), versus connecting two random nodes across the graph. The \( \rho \) parameter is a single value that is efficiently learned from the input graph via an expectation maximization algorithm.

From the privacy perspective, although the degree sequence can be estimated from an input graph with reasonably high accuracy under differential privacy [44], the clustering parameters
used by these models are problematic. Informally, with respect to the clustering coefficients by degree $C_D$ required by BETR, the difficulty is that a single edge, say $e_{ij}$, may be part of many triangles that are counted in the $c_i$’s for many degrees $i$. Moreover, adding or removing edge $e_{ij}$ will change the degrees of nodes $v_i$ and $v_j$, which could cause a significant change in four different $c_i$’s (i.e., those of the old degrees $d_i, d_j$ and the new degrees $d_i \pm 1, d_j \pm 1$). This translates into a high global sensitivity, and it is unclear whether it is possible to circumvent this issue. With respect to TCL, the main difficulty comes from the fact that the $\rho$ parameter is tuned through multiple iterations of an expectation maximization algorithm. An edge addition or deletion may not only affect the first iteration but the effects may also propagate through to other iterations and it is difficult to determine what the ultimate impact will be on the final value of $\rho$.

**Proposed Approach.** Motivated by the difficulty of incorporating differential privacy guarantees into existing generative social network models, we chose to design a new model around an statistic for which accurate differentially private estimators are already known, namely the triangle count. Our model, which we call TriCycLe, was inspired by TCL. Briefly, rather than learning a transitive closure parameter, TriCycLe first estimates the number of triangles in the input graph and then rewire edges in a Chung-Lu seed graph until the desired number of triangles are created. In addition to being more amenable to differential privacy, we have found that TriCycLe is often able to attain a better fit to the local clustering coefficient distribution than TCL.

In the next section, we give a brief overview of TCL and the CL model that it extends. We then describe how TriCycLe modifies TCL to replace the transitive closure parameter with the triangle count, and how that statistic is used to introduce the expected amounts of clustering into the output graph. In Section 5.3.3.5 we explain how the required model parameters—the degree distribution and the number of triangles—can be estimated from the input graph privately and efficiently. In Section 5.4, we explain how TriCycLe can be integrated into AGM as the underlying structural model.
5.3.3.1 A Review of CL and TCL

In this section, we briefly describe the sampling algorithm of TCL, upon which our TriCycLe algorithm is based; however, we start with a brief review of the Chung Lu (CL) random graph model, which is used within both TCL and TriCycLe. In the CL model, a graph is generated by (1) assigning a desired degree to every node based on the degree sequence of the input graph, and (2) sampling edges with a probability proportional to the desired degrees of the endpoint nodes; specifically, an edge $e_{ij}$ is sampled with probability $\frac{d_i d_j}{2m}$. A graph matching a given degree sequence can be generated in $O(m)$ time by repeatedly sampling pairs of nodes $v_i, v_j$ independently from the distribution $\pi$, where $\pi(i) = \frac{d_i}{2m}$, and then adding edge $e_{ij}$ to the sample graph. [102] describes an efficient implementation of the above idea, called Fast Chung-Lu (FCL), in which nodes are sampled from $\pi$ in constant time by constructing a vector of length $2m$ in which the ID of a node $v_i$ is repeated $d_i$ times, and then choosing nodes uniformly at random from this vector. Although, CL is able to provably match the degree distribution of the input graph in expectation, it includes no mechanism to reproduce the large amounts of clustering found in real social networks.

TCL extends CL to incorporate transitive edge formation, i.e., when a node connects to a friend of a friend, which accounts for most of the clustering found in social networks [100]. In TCL, a transitive closure probability, $\rho$, controls the probability by which an edge is added between a randomly selected node and a two-hop neighbor (i.e., a friend of a friend), rather than to another randomly selected node, as in the CL model. More precisely, given a degree sequence $S$ and $\rho$ (both learned from the input graph), TCL generates a synthetic graph in two phases. First, a seed graph is created by sampling a total of $m$ edges from the CL model, parameterized by $S$, i.e., from $\pi$. The seed graph will have the same degree sequence, in expectation, as the input graph but will contain little to no clustering. Second, TCL refines the seed graph by introducing triangles in a way that maintains the expected degree sequence. To do so, a node $v_i$ is first sampled from the $\pi$ distribution; then with probability $\rho$, a node $v_k$ is selected uniformly
from among \( v_i \)'s neighbors and an edge is then added between \( v_i \) and a uniformly selected neighbor of \( k \), \( v_j \) (if it doesn't already exist), which results in at least one new triangle. With probability \( 1 - \rho \), the endpoint \( v_j \) is sampled from \( \pi \) instead. After each new edge addition, the oldest edge in the graph is deleted to ensure that the expected degree sequence is maintained. The process of replacing edges continues until all of the old (seed) edges have been replaced.

5.3.3.2 TriCycLe Random Graph Model

Similarly to TCL, TriCycLe is based on the intuition that the large amounts of clustering seen in social graphs is due to the triangles that form when a user befriends a two-hop neighbor (i.e., a friend-of-a-friend). Therefore, TriCycLe attempts to reproduce the total number of triangles seen in the input graph by connecting nodes in this way. Like TCL, TriCycLe starts with a Chung-Lu seed graph and iteratively replaces edges to create triangles. However, rather than using a transitive closure probability, we count the number of triangles in the input graph and then subsequently add transitive edges until the number of triangles in the resulting graph matches that of the input graph. As we will explain in the next section, the number of triangles in a graph can be accurately estimated under differential privacy using a recent technique [139]. Like TCL, each time we add a new transitive edge, we remove a seed edge to maintain the expected degree distribution; however, since the deleted edge may itself be part of one or more existing triangles, we reject proposed replacements that decrease the total triangle count. This check ensures that the algorithm terminates and that creating the desired number of triangles is achievable.

Algorithm 5 gives the procedure for generating a graph with TriCycLe. The inputs to the algorithm are the degree sequence \( S = \{d_i | v_i \in G\} \) and the desired number of triangles \( n_\Delta \). For now, we will assume that these two quantities have been directly measured from the input graph \( G \) (i.e., non-privately); in the next section, we explain how they can be accurately and efficiently estimated with differential privacy. In line 2, a seed graph is sampled from the CL\(^6\)

\(^6\)Specifically, we use the FCL variant with bias correction (cFCL) described in [100].
Algorithm 5 TriCycLe Graph Generation

Input: $N, \Theta_M = \{\mathcal{S}, n_\Delta\}$
Output: $E_T$

1: Compute $\pi$ distribution from $N$ and degree sequence $\mathcal{S}$
2: $E_T \leftarrow \text{CL}(\mathcal{S}, \pi)$
3: $nTri \leftarrow \text{COUNTTRIANGLES}(E_T)$
4: while $nTri < n_\Delta$ do
5: $v_i \leftarrow \text{SAMPLE}(\pi)$
6: $nbrs_i \leftarrow \text{NBR}(E_T, v_i)\backslash\{v_i\}$
7: $v_k \leftarrow \text{SAMPLEUNIFORM}(nbrs_i)$
8: $nbrs_k \leftarrow \text{NBR}(v_k)$
9: $v_j \leftarrow \text{SAMPLEUNIFORM}(nbrs_k)$
10: if $e_{ij} \notin E_T$ then
11: $e_{qr} \leftarrow \text{OLDESTEDGE}(E_T)$  
12: $CN_{qr} \leftarrow |\text{NBR}(v_q) \cap \text{NBR}(v_r)|$  
// Count number of common neighbors between $v_q$ and $v_r$
13: $E_T \leftarrow E_T \backslash e_{qr}$
14: $CN_{ij} \leftarrow |nbrs_i \cap \text{NBR}(v_j)|$
15: if $CN_{ij} \geq CN_{qr}$ then
16: $E_T \leftarrow E_T \cup e_{ij}$
17: $nTri \leftarrow nTri + CN_{ij} - CN_{qr}$  
// Update triangle count
18: else
19: $E_T \leftarrow E_T \cup e_{qr}$  
// Undo the edge removal, making $e_{qr}$ the youngest edge in $E_T$
20: end while
21: return $E_T$
model, yielding edge set $E_T$. Recall that in CL, $m = \frac{1}{2} \sum_{d_i \in S} d_i$ edges are sampled from the $\pi$ distribution that is created from $S$. In the main loop (line 4), we add transitive edges one at a time until $E_T$ contains the desired number of triangles. A new edge is proposed by sampling a node $v_i$ from the $\pi$ distribution (i.e., with probability proportional to node degree) and then selecting a random “friend of a friend”, $v_j$. Assuming the proposed edge $e_{ij}$ does not already exist in $E_T$, we select the oldest edge $e_{qr}$ (line 11) to be deleted in order to maintain the expected degree distribution. In lines 12–14 we calculate the number of triangles that would be created (destroyed) by adding (deleting) edge $e_{ij}$ ($e_{qr}$). Note that we delete $e_{qr}$ in line 13 before computing $CN_{ij}$, since its presence may affect the number of triangles created by connecting $v_i$ and $v_j$. As long as the proposed edge addition/deletion does not decrease the number of triangles (line 15), we proceed with adding $e_{ij}$ in line 16 and updating the triangle count in line 17. Otherwise, we undo the deletion of $e_{qr}$ in line 19 before proceeding to the next iteration. Note that if and when the deletion of $e_{qr}$ is reverted, its timestamp is reset so that it becomes the youngest edge in $E_T$; this is an important detail, since without it, the algorithm could get stuck, as there may not (at that moment) be any other edges in the graph that participate in more triangles than $e_{qr}$. It is worth pointing out that this algorithm implicitly assumes that the random graph generated by CL in line 1 will have fewer triangles than the input graph; in the rare event that this assumption does not hold (e.g., for very small input graphs), then the output graph may have more triangles than desired.

5.3.3.3 Extension to Handle Orphaned Nodes

A practical issue with the Chung-Lu model and those based upon it, such as TCL and TriCycLe, is that the graphs generated by these models tend to contain a much larger fraction of disconnected or orphaned nodes\(^7\), compared to the input graph. This is due to the probabilistic nature of the algorithm—although the low-degree nodes in the input graph (say, those with degree one or two) will, on average, have the desired degree in the generated graph, some frac-

\(^7\)We consider a node to be orphaned if it is not part of the main connected component of the graph. Recall that in this work, we assume that the input graph is connected, and therefore the output graph should be also.
Figure 5.4  This figure shows the number of orphaned nodes produced by FCL, TCL and TriCycLe on four datasets, each consisting of a single connected component.

As a concrete example, Fig. 5.4 shows the number of orphaned nodes produced by FCL, TCL and TriCycLe on four different datasets, all of which consisted of a single connected component. Observe that for the largest graph (Pokec), which has around 720K nodes (and many degree-one and degree-two nodes), around 100K of them ended up orphaned in the output graph.

We address the issue of orphaned nodes with an extension to TriCycLe$^8$. Recall that in this work we make the assumption that the input graph is connected (Section 5.2.1). We start by observing that (1) most of the nodes that end up orphaned are those with a degree of one in the input graph, and (2) since degree-one nodes cannot be part of any triangle, there is no reason to pick such nodes in line 5 of Algorithm 5. Let $N_1$ denote the set of degree-one nodes in the input graph. We make the following changes to Algorithm 5: (1) we exclude the degree-one nodes from the $\pi$ distribution computed on line 1; (2) when generating the CL graph in line 2, we generate $m - |N_1|$ edges, rather than $m$ edges; (3) we wire up the degree-one nodes, and

$^8$This extension could also be applied to FCL and TCL.
any other orphaned nodes, in a separate post-processing step, given by Algorithm 6. Note that
we apply the post-processing step to both the CL graph generated in line 2 (of Algorithm 5),
and to the final output graph, before it is returned in line 21.

**Algorithm 6 PostProcessGraph**

**Input:** $N, E, \pi, S$

**Output:** $E'$

1. $m \leftarrow \frac{1}{2} \sum_{v_i \in N} S_{v_i}$
2. $E' \leftarrow E$
3. $m' \leftarrow |E'|$
4. **while** graph $G \leftarrow (N, E')$ is disconnected **do**
5.   Select an orphaned node $v_i$
6.   **if** $d_i^{(G)} > 0$ **then**
7.     $m' \leftarrow m' - d_i^{(G)}$
8.     Delete edges incident to $v_i$
9.   **for** $j \leftarrow 1 \ldots S_{v_i}$ **do**
10.     **repeat**
11.         $v_k \leftarrow \text{SAMPLE}(\pi)$
12.     **until** $d_k^{(G)} < S_{v_k}$
13.     $E' \leftarrow E' \cup e_{ik}$
14.   **if** $m' = m$ **then**
15.       **Delete a random edge from** $E'$
16.   **else**
17.     $m' \leftarrow m' + 1$
18. **end while**
19. **return** $E'$

The post-processing algorithm takes as input: the set of nodes $N$, and edges $E$, of the graph
that is to be processed, along with the $\pi$ distribution and the degree sequence of the original
input graph, which corresponds to the desired degrees for the nodes\(^9\). In the main loop (lines
4–18), we process each orphaned node $v_i$ by first deleting any existing incident edges (which
could only be connected to other orphaned nodes) and then connecting $v_i$ to the main connected
component by adding edges to nodes in the main component (for which their desired degree
has not yet been met) until $v_i$’s desired degree is met. Upon adding each new edge, if we have
more than the desired number of edges overall, we delete a randomly selected edge (line 15).

\(^9\)Note that the degree-one nodes are included in $S$, but excluded from $\pi$. 

116
Note that deleting existing edges (line 15) may result in additional orphaned nodes, therefore, we repeat this process as long as the graph is still disconnected. Also note that, although some triangles may be destroyed by the edge removal in line 15, it typically does not significantly impact the clustering coefficients in practice.

5.3.3.4 Empirical Evaluation

We validated the proposed structural model by generating synthetic versions of four real-world social graphs (described in Sec. 5.5.1) and comparing the structural characteristics of the synthesized graphs to that of the original graphs, as well as to synthetic graphs generated by FCL and TCL. Figs. 5.5 (a)–(d) plot the degree distributions for the four datasets, while Figs. 5.6 (a)–(d) plot the local clustering coefficient distributions. The y-axis in each plot is the complementary cumulative degree distribution (CCDF), i.e., the y-value at each point is the fraction of nodes with a greater degree or clustering coefficient than the x-value. Note that the degree distributions are plotted on a log-log scale.

Looking at Figs. 5.6(a)–(d), we see that, as expected, the clustering coefficient distributions of the graphs generated by TCL and TriCycLe were much closer to that of the original graph than for FCL, on all four datasets; recall that FCL does not explicitly model clustering. Compared to TCL, TriCycLe was able to more closely approximate the clustering coefficients of the Last.fm and Epinions graphs, while its performance was comparable to that of TCL on the Petster graph. On the Pokec graph, TriCycLe appeared to over-estimate the clustering coefficients somewhat. All three of the models approximated the degree distributions reasonably well (Figs. 5.5(a)–(d)), though we note that on two of the datasets, the graphs synthesized by TriCycLe exhibited a slight bias in the high degree nodes. The bias appears to be a consequence of the condition that requires edge replacements to increase the triangle count; removing edges from high degree nodes tends to remove more triangles, so fewer such edges are removed. In general, there appears to be an inherent tension between the fidelity of the degree distribution and that of the clustering characteristics. Nevertheless, TriCycLe captures both properties suf-
Figure 5.5 Comparison of structural models in terms of their ability to reproduce the degree distribution on four datasets.

5.3.3.5 TriCycLe-DP: Privately Fitting TriCycLe

The two inputs required by the algorithm are the degree sequence $S$ and the number of triangles $n_\Delta$, measured from the input graph $G$. Here we discuss how to compute differentially private estimates of these statistics using existing techniques. Using the private statistics as the inputs...
to Algorithm 5, we can generate a graph that, by extension, also satisfies DP.

**Degree Sequence.** The degree sequence of input graph $G$ is the set $S = \{d_i | v_i \in N\}$. A straightforward way to satisfy differential privacy for $S$ is to apply the Laplace mechanism (Thm. 2.1.1), adding independent noise drawn from Lap($2/\epsilon$) to each $d_i \in S$; the global sensitivity is two because adding or removing a single edge of $G$ changes the degrees of exactly two nodes by one. However, this approach introduces too much noise, especially in the low degree nodes, which are typically abundant in real social graphs. Instead, it is possible to get a
much better estimate of the degree sequence by taking advantage of the fact that the mapping between a specific node and its degree is unimportant for how the degree sequence is used in TriCycLe—we only need the sequence of degrees, the order is unimportant. [44] proposed an approach for accurately estimating the degree sequence of a graph based on constrained inference, where the high-level idea is to sort the degree sequence prior to adding noise and then to post-process the noisy sequence to ensure that the ordering condition is still met, thereby canceling out a significant portion of the noise without affecting the privacy guarantee\textsuperscript{10}. More precisely, the algorithm in [44] starts by computing the actual degrees for each node, forming degree sequence \( S \). It then sorts \( S \) in non-decreasing order and adds independent noise drawn from \( \text{Lap}(2/\epsilon) \) to each degree, yielding noisy degree sequence \( \tilde{S} \); the addition of noise may cause the ordering constraint to be violated. A constrained inference procedure is applied to \( \tilde{S} \) to find the \( S \) that is “closest” to \( \tilde{S} \) (i.e., the minimum \( L_2 \) distance) that also satisfies the ordering constraint. A dynamic programming solution given in [44] performs the constrained inference operation in linear time.

\textbf{Triangle Count.} The second input expected by TriCycLe is the number of triangles in the input graph. The challenge here, from the privacy perspective, is that the global sensitivity of triangle counting is prohibitively high, since in the worst case there could be a single edge that is shared by every triangle in the graph—for a graph with \( n \) nodes, adding or removing a single edge could change the triangle count by up to \( n - 2 \), as illustrated earlier in Fig. 5.1. Consequently, a direct application of the Laplace mechanism (Theorem 2.1.1) yields a highly inaccurate count. Fortunately, real-world social graphs typically do not resemble the worst case structure depicted in Fig. 5.1 and tend to have local sensitivities that are much lower. Triangle counting has been extensively studied in the context of differential privacy (e.g., [98, 139, 58, 20]). The most effective approaches take advantage of the above observation in some way to reduce the amount of noise required to satisfy differential privacy.

\textsuperscript{10}Recall that \( \epsilon \)-differential privacy is invariant under post-processing (Thm. 2.1.5).
Ladder framework, introduced by Zhang et al. [139], which effectively combines the concept of “local sensitivity at distance \( t \)”, from the smooth sensitivity framework [98], with the exponential mechanism [88]. In the context of triangle counting, it has been shown to provide better accuracy and has the added benefit that it provides pure differential privacy, rather than the weaker notion of \((\epsilon, \delta)\)-differential privacy provided by the smooth sensitivity framework.

The local sensitivity of a function \( f \) at distance \( t \) (from the input graph) is denoted \( LS_f^t(G) \), and quantifies the maximum local sensitivity among all graphs that can be formed from \( G \) by adding or deleting up to \( t \) edges. For the triangle counting query \( f_{n\Delta} \), this quantity is given by the following lemma, restated from [98, 139].

**Lemma 5.3.8** (Claim 3.13 from extended version of [98]). The local sensitivity of \( f_{n\Delta} \) at distance \( t \) is

\[
LS_{f_{n\Delta}}^t(G) = \max_{v_i, v_j \in N(G), v_i \neq v_j} \min \left(a_{ij} + \left\lfloor \frac{t + \min(t, b_{ij})}{2} \right\rfloor, n - 2 \right)
\]

where \( a_{ij} \) is the total number of triangles incident to potential edge \( e_{ij} \) (i.e., \( |\text{Nbr}(v_i) \cap \text{Nbr}(v_j)| \)), and \( b_{ij} = d_i + d_j - 2a_{ij} - E_{ij} \), where \( E_{ij} \) is one if \( e_{ij} \) exists and 0 otherwise, is the number of nodes that are connected to either \( v_i \) or \( v_j \), but not both.

In the ladder framework, \( LS_{f_{n\Delta}}^t(G) \) is used to directly derive a quality function that can be used to instantiate the exponential mechanism\(^{11}\). This mechanism allows us to sample an approximate value for the triangle count, such that better approximations are exponentially more likely to be sampled. An efficient algorithm for constructing and sampling from this distribution is given in [139], and we do not restate it here.

### 5.3.3.6 Algorithm for Fitting TriCycLe-DP

Algorithm 7 summarizes the procedure for fitting the parameters used by TriCycLe from an input graph \( G \), while Thm. 5.3.9 shows that the algorithm satisfies \( \epsilon \)-differential privacy.

\(^{11}\)See Section 2.1.1 for a review of the exponential mechanism.
Algorithm 7 FitTriCycLeDP

Input: $\epsilon, E, n$
Output: $\tilde{\Theta}_M = \{\tilde{S}, \tilde{n}_\Delta\}$

1: // Split the privacy budget (evenly) between the degree sequence and triangle count
2: $\epsilon_S \leftarrow \frac{\epsilon}{2}$
3: $\epsilon_\Delta \leftarrow \frac{\epsilon}{2}$
4: $S \leftarrow (d_1, \ldots, d_n)$
5: $S \leftarrow \text{sortAscending}(S)$
6: // Add independent Laplace noise to each coordinate of degree sequence $S$
7: $\tilde{S} \leftarrow S + \text{Lap}(0, \frac{2}{\epsilon_S})^n$
8: Apply constrained inference (Algorithm 1 from [44]) to $\tilde{S}$ to get $\bar{S}$
9: for $\bar{d}_i \in \bar{S}$ do
10: Round $\bar{d}_i$ to the nearest integer in $\{0, \ldots, n - 1\}$
11: // Get a differentially private estimate of the triangle count using Algorithm 1 from [139]
12: $\tilde{n}_\Delta \leftarrow \text{NoiseSample}(f_{n_\Delta}, E, \epsilon_\Delta)$
13: return $\tilde{\Theta}_M = \{\tilde{S}, \tilde{n}_\Delta\}$

Theorem 5.3.9. Algorithm 7 satisfies $\epsilon$-differential privacy.

Proof. Let $G = (N, E, X)$ be an arbitrary input graph and, without loss of generality, suppose we form a neighboring graph $G'$ by adding an arbitrary edge $e_{ij}$, such that the degrees of the two incident nodes in $G$ are $d, d'$ respectively. Let $S$ and $S'$ denote the degree sequence vectors for $G$ and $G'$ sorted in non-descending order, respectively. Thus, relative to $S$, two values in $S'$ are different: the right-most $d$ has changed to $d + 1$ and the right-most $d'$ has become $d' + 1$. Thus, the $L_1$ difference $|S' - S| = 2$, which gives a global sensitivity of 2 (Def. 2.1.3). In line 7 we add Laplace noise to each coordinate of the sorted degree sequence vector by invoking the Laplace mechanism. Since the global sensitivity is 2 and we add Laplace noise with a scale of $\frac{2}{\epsilon_S}$, we have $\epsilon_S$-differential privacy for the degree sequence (by Thm. 2.1.1). Performing the constrained inference procedure on the noisy degree sequence (line 8) has no effect on the privacy guarantee, since it does not look at the original degree sequence (Thm. 2.1.5); likewise for the subsequent rounding of the post-processed degrees. Line 11 computes an $\epsilon_\Delta$-differentially private estimate of the number of triangles in $G$ using the Ladder framework of [139]. By Thm. 2.1.3 (sequential composition), these two differentially private computations together satisfy $\epsilon$-differential privacy, where $\epsilon = \epsilon_S + \epsilon_\Delta$. \qed
5.4 AGM-DP: Putting It All Together

In the previous section we described how to compute differentially private estimates of the modeling parameters used by AGM. We also introduced a new structural model that can be used as the underlying edge generating model $M$ and showed how its parameters $\Theta_M$ can be learned privately. In this section, we put the pieces together and outline the end-to-end workflow for synthesizing differentially private graphs that mimic the attribute correlations and structural properties of a given social network. For concreteness, the presented algorithm assumes that TriCycLe is the underlying structural model; however, other generative structural models that can be made to satisfy differential privacy could be used in this workflow. In Section 5.4.2 we provide a formal statement of the privacy guarantee provided by AGM-DP.

5.4.1 End-to-End Workflow

In the original AGM algorithm (Algorithm 2 in Section 5.2.2), a synthetic graph is generated for an input graph $G$ by repeatedly sampling edges from an underlying structural model $M$ and then either accepting or rejecting them into the output graph based on the acceptance probabilities $A$; and this process repeats until the desired number of edges (i.e., $m^{(G)}$) has been generated. However, the TriCycLe model (Section 5.3.3.2) does not fit this pattern directly, as it continues to refine an initial seed graph by replacing edges to create new triangles until the desired number of triangles exists in the output graph. Therefore, to accommodate models such as TriCycLe, we move the accept/reject step into the sampling algorithm for the structural model—that is, we still compute the acceptance probabilities as in the original AGM algorithm (aside from the alterations required for satisfying differential privacy), but we then pass the set of probabilities $A$ into $M$’s specific sampling algorithm when generating the final output graph. Specifically, to use TriCycLe with the algorithm below, we need to change the condition in the if statement on line 10 of Algorithm 5 to “$e_{ij} \notin E_T \land \text{SampleUniform}(0, 1) \leq A(F_w(\tilde{x}_i, \tilde{x}_j))$”, where $\tilde{x}_i$ and $\tilde{x}_j$ are the attribute vectors generated for nodes $v_i$ and $v_j$ in Algorithm 4. Additionally,
since TriCycLe only continues to replace edges until the desired number of triangles exist, there may be a large subset of edges that are not replaced and thus are not subjected to the acceptance probabilities. Therefore, we also pass the acceptance probabilities into the CL seed graph generator on line 2 of Algorithm 5.

Algorithm 8 AGM-DP-TriCycLe

Input: \( G = (N, E, X), w, \epsilon, k \)

Output: An \( \epsilon \)-DP graph \( \tilde{G} = (\tilde{N}, \tilde{E}, \tilde{X}) \)

1: \( \tilde{N} \leftarrow N \)
2: \( \epsilon_X, \epsilon_F, \epsilon_M \leftarrow \text{SPLITBUDGET}(\epsilon) \) \quad \text{// Split privacy budget \( \epsilon \) among the parameters to be learned}
3: // Learn DP approximations for each parameter from input graph \( G \)
4: \( \tilde{\Theta}_X \leftarrow \text{LEARNATTRIBUTESDP}(\epsilon_X, X, w) \)
5: \( \tilde{\Theta}_M \leftarrow \text{FITTRICYCLEDP}(\epsilon_M, E) \)
6: \( \tilde{\Theta}_F \leftarrow \text{LEARNCORRELATIONS}(\epsilon_F, G, k) \)
7: Sample new attribute vectors \( \tilde{X} \) using \( \tilde{\Theta}_X \)
8: \( \tilde{E} \leftarrow \text{TRICYCLE}(\tilde{N}, \tilde{\Theta}_M) \)
9: \( A \leftarrow \emptyset \)
10: // Compute the acceptance probabilities
11: \textbf{for} \text{ Iterations } \textbf{do}
12: \hspace{1em} Compute \( \tilde{\Theta}'_F \) from the graph \( \tilde{G} = \{\tilde{N}, \tilde{E}, \tilde{X}\} \)
13: \hspace{1em} \textbf{for} \( y_i \in Y_F \) \textbf{do}
14: \hspace{2em} \( R(y_i) \leftarrow \frac{\tilde{\Theta}_F(y_i)}{\tilde{\Theta}'_F(y_i)} \)
15: \hspace{2em} \textbf{if} \( A_{old} \neq \emptyset \) \textbf{then}
16: \hspace{3em} \( R(y_i) \leftarrow R(y_i) \times A_{old}(y_i) \)
17: \hspace{1em} \textbf{for} \( y_i \in Y_F \) \textbf{do}
18: \hspace{2em} \( A(y_i) \leftarrow \frac{R(y_i)}{\sup(R)} \)
19: \hspace{1em} \( \tilde{E} \leftarrow \text{TRICYCLE}(\tilde{N}, \tilde{\Theta}_M, \tilde{X}, A) \) \quad \text{// Sample a new edge set using acceptance probabilities}
20: \hspace{1em} \( A_{old} \leftarrow A \)
21: \textbf{return} \( \tilde{G} = (\tilde{N}, \tilde{E}, \tilde{X}) \)

The end-to-end differentially private workflow, which we refer to as AGM-DP, is given in Algorithm 8 and illustrated in Fig. 5.7. It takes an input graph \( G \), the number of attributes \( w \), a global privacy budget \( \epsilon \), and a truncation parameter \( k \) (see Section 5.3.1.2). The algorithm begins by splitting the overall privacy budget \( \epsilon \) among the differentially private learning pro-
Figure 5.7 Fig. 5.7a provides a graphical overview of our differentially private adaptation of AGM, with Fig. 5.7b depicting the sampling step (corresponding to the right-most block in 5.7a).
cedures for the three model parameters. We chose to split the budget evenly in our empirical analysis (Section 5.5), i.e., $\epsilon_X = \epsilon_F = \frac{1}{4} \epsilon$, $\epsilon_M = \epsilon_S + \epsilon_\Delta = \frac{1}{4} \epsilon + \frac{1}{4} \epsilon$, which seems to work well in practice; though, other strategies could also be used. In lines 4–6, we learn the three modeling parameters from the input graph using their respective differentially private learning procedures. Note that, after line 6, we never look back at the raw input graph. Next we assign attribute vectors to all of the nodes by drawing samples randomly from the distribution defined by $\tilde{\Theta}_X$. We then generate a temporary edge set $E'$, where the edges are sampled independently of the node attributes. The loop beginning on line 11 computes the edge acceptance probabilities. We start by measuring the attribute correlations $\Theta'_F$ in the temporary graph defined by $\tilde{G} = \{\tilde{N}, \tilde{E}, \tilde{X}\}$, and computing the ratios between $\Theta'_F$ and the $\tilde{\Theta}_F$ measured (privately) from the input graph (lines 12–14). The acceptance probabilities are then obtained by dividing the ratios by the supremum. Finally, a new graph $\tilde{G}$ is sampled using the modified structural model (as discussed above), which proposes edges based on the parameters $\tilde{\Theta}_M$ and accepts or rejects them according to the acceptance probabilities $A$ (line 19). We loop until the acceptance probabilities converge, updating $A$ at each iteration (line 16) to adjust for edge-attribute combinations that were over-sampled or under-sampled in the previous iteration. It is only necessary to iterate more than once for structural models like TriCycLe, where the edges are not drawn independently of one another. Recall that in TriCycLe, an initial seed graph is created (using FCL) and then edges are incrementally rewired to create the desired number of triangles; thus, the edges inserted in the triangle creation phase are dependent on those laid in the seed graph, and by extension, are dependent on the acceptance probabilities $A$. For simple models like $FCL$, we would stop after the initial iteration and return $\tilde{G}$. We note that in our experimental study, we found that $A$ tended to converge after just a few iterations. The output graph $\tilde{G}$ approximates a joint sampling of the structural model and the attribute correlations, and satisfies differential privacy (as the next section shows).
5.4.2 Privacy Analysis

**Theorem 5.4.1.** Algorithm 8 satisfies $\epsilon$-differential privacy.

*Proof.* The privacy budget $\epsilon$ is split into three parts, $\epsilon_X, \epsilon_F, \epsilon_M$, which are used in the learning procedures for the three sets of model parameters (lines 4–6). The fact that the learning procedures satisfy differential privacy with privacy parameters $\epsilon_F, \epsilon_X, \text{and } \epsilon_M$, respectively, follows from Thm. 5.3.7, Thm. 5.3.6, and Thm. 5.3.9, respectively. After line 6, the algorithm never looks back at the raw input data again. Thus, by Thm. 2.1.3 (sequential composition) and Thm. 2.1.5 (post-processing invariance) we have that the algorithm as a whole satisfies $\epsilon$-DP. \qed

5.4.3 Extensions

The framework presented in this work could be extended in a number of interesting ways. We will briefly discuss a few of the more important extensions now.

5.4.3.1 Directed Edges

In terms of computing $\Theta_F$, directed edges could be supported by changing the $F_w$ function that maps edges to unique integers, so that it takes the edge direction into account. Accommodating directed edges increases the number of counts that must be computed for $\Theta_F$. Specifically, for directed graphs with $w$ attributes, we need to compute $4^w$ counts (rather than the $\left(\frac{2^w+1}{2}\right)$). However, for small $w$ and reasonably large input graphs, supporting directed edges should not be a problem. Another challenge that arises with directed edges is how to satisfy differential privacy for structural models like TriCycLe and FCL that require knowing the (approximate) degree sequence of the input graph. In this work, we used the constrained inference approach of [44] to obtain a private estimate of the degree sequence of an undirected graph; however, for directed graphs, we would need to capture both the in-degree and out-degree sequences, which removes the possibility of using constrained inference to boost the accuracy. The problem is
that constrained inference exploits the fact that the order of the sequence does not matter; however, to correctly model the joint degree sequence in directed graphs, we need to maintain the mapping between corresponding nodes in the in-degree and out-degree sequences. Addressing this challenge would be an interesting direction for future work.

5.4.3.2 Non-Binary Attributes

The framework could be made to support non-binary categorical or continuous attributes by simply converting each attribute to a series of binary attributes, one per category or range\textsuperscript{12}. Although increasing the number of attributes, \( w \), has no impact on the sensitivity of our approach, it does increase the number of counts that need to be computed for \( \Theta_X \) and \( \Theta_F \); thus we can expect the accuracy of the generated graphs to degrade as \( w \) increases, due to a higher noise-to-signal ratio for the counts. In general, the larger the graph, the more attributes that could be supported accurately, since the amount of noise is also independent of the size of the graph.

5.4.3.3 Node Privacy

In this work we focused on satisfying edge-differential privacy, in which neighboring graphs differ in a single edge or the attributes of a single node. A stronger notion of adjacency which has been studied in a few recent works is node-differential privacy, in which neighboring graphs differ in a single node and all of its adjacent edges (and/or the attributes associated with one node). This is a more difficult definition to satisfy, as it translates to a much higher global sensitivity for many tasks. For example, the simple task counting the number of edges in a graph has a sensitivity of \( n \), compared to a sensitivity of one under edge-differential privacy. We have developed a preliminary approach for computing \( \Theta_F \) under node-privacy with reasonably high accuracy. The general idea is to use the same edge truncation approach but then to add noise according to its smooth sensitivity in the node-adjacency model. In a preliminary experiment

\textsuperscript{12}For example, marital status could be represented by the following three binary attributes: isMarried, isDivorced, isSingleOrWidowed.
(similar to the one in Section 5.3.1.4), we found that using this approach, the Hellinger distance
between the original and noisy correlation probabilities was still better than the baseline when
\( \epsilon \geq \ln(2) \) on Last.fm, when \( \epsilon \geq 0.3 \) on Petster, when \( \epsilon \geq 0.2 \) on Epinions, and when \( \epsilon \geq 0.05 \) on
Pokec; \( \delta \) was fixed at 0.01 for the experiment. Developing a node-private approach for fitting a
suitable structural model is non-trivial and would be a worthwhile direction for future work.

5.4.3.4 Other Graph Types

We chose to focus on attributed social networks in this work for concreteness; however the
proposed framework could be extended to support other types of attributed graphs by plugging
in an appropriate underlying structural model for \( M \). As long as the structural model satisfies
differential privacy, the framework as a whole will still satisfy differential privacy.

5.5 Empirical Analysis

In this section, we present an experimental study of our differentially private adaptation of
AGM on several real-world social network datasets. Our goal for this study is to demonstrate
that the synthetic graphs produced by AGM-DP reasonably preserve the attribute correlations
and structural properties of an input social graph, while simultaneously attaining a strong level
of privacy. We instantiated AGM-DP with the differentially private version of TriCycLe (abbrevi-
ated as AGMDP-TriCL henceforth), as well as a differentially private version of the simple
FCL model (abbreviated as AGMDP-FCL henceforth) as the underlying structural models. For
the latter, we modified the bias-corrected FCL algorithm (cFCL) described in [100] to take a
noisy degree sequence as input, which was generated using the constrained inference approach
of [44], just as was done for TriCycLe (see Section 5.3.3.5). We generated synthetic versions
of four social network datasets (discussed in the next section) under several different privacy
settings and compared the synthesized graphs to the original input graphs with respect to a
number of graph statistics (described in Section 5.5.3).
5.5.1 Datasets

We conducted experiments on four real-world social network datasets:

- **Last.fm**\(^\text{13}\) is a social music platform where users listen to and rate music, as well as share their favorite music with friends. This publicly available dataset [15] contains undirected friendship edges between user nodes, as well as a set of “listened to” relations indicating which artists a user has listened to. Although the dataset does not contain node attributes, we created two attributes by selecting the two most popular artists and creating \textit{listenedToArtistX} attributes, with a value of 1 if a user had listened to artist X at least once.

- The **Epinions dataset** [84] was crawled from Epinions.com, a website where users review and rate products. The dataset contains directed edges between users that represent positive trust statements, e.g., that user \(A\) trusts the reviews/ratings produced by user \(B\). We converted the directed edges to undirected edges by keeping only the \textit{mutual} trust relationships, i.e., we keep edge \(e_{ij}\) only if both \(e_{ij}\) and \(e_{ji}\) were present. Similarly to the Last.fm dataset, we created binary node attributes by selecting the two most frequently-rated products in the dataset and creating attributes with a value of 1 if a user had rated the product.

- **Petster** is a website where users create web pages for their pets and form friendships links with other pet owners\(^\text{14}\). The dataset\(^\text{15}\) is a crawl of the undirected friendship network of pet hamster owners and includes node attributes such as the hamster’s name, sex, age, etc. We used two binary attributes: sex and \textit{is-living}, where is-living was set to 0 if the value of the age attribute was “gone to hamster heaven”, or 1 otherwise.

- **Pokec** is a popular Slovakian online social network. This dataset [119] is publicly available

\(^{13}\)http://www.last.fm  
\(^{14}\)www.petster.com.  
\(^{15}\)http://konect.uni-koblenz.de/networks/petster-friendships-hamster
as part of the Stanford Large Network Dataset Collection\textsuperscript{16} and contains directed friendship edges and node attributes such as gender, age, hobbies, etc. As with the Epinions dataset, we converted the edges to undirected by keeping mutual friendships. We used two attributes: sex and age. We converted age into a binary attribute by assigning a value of 1 for ages $\leq 30$ and 0 for ages greater than 30. We ignored nodes with missing values for gender or age.

For all datasets, we considered only the main connected component (after converting to undirected edges, where applicable). Table 5.2 summarizes the four (pre-processed) datasets, with respect to various statistics.

Table 5.2 Datasets used in empirical analysis. The columns are number of nodes $n$, number of edges $m$, maximum degree $d_{\text{max}}$, average degree $d_{\text{avg}}$, number of triangles $n_\Delta$, and average local clustering coefficient $C$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$m$</th>
<th>$d_{\text{max}}$</th>
<th>$d_{\text{avg}}$</th>
<th>$n_\Delta$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last.fm</td>
<td>1,843</td>
<td>12,668</td>
<td>119</td>
<td>6.9</td>
<td>19,651</td>
<td>0.183</td>
</tr>
<tr>
<td>Petster</td>
<td>1,788</td>
<td>12,476</td>
<td>272</td>
<td>7.0</td>
<td>16,741</td>
<td>0.143</td>
</tr>
<tr>
<td>Epinions</td>
<td>26,427</td>
<td>104,075</td>
<td>625</td>
<td>3.9</td>
<td>231,645</td>
<td>0.138</td>
</tr>
<tr>
<td>Pokec</td>
<td>592,627</td>
<td>3,725,424</td>
<td>1,274</td>
<td>6.3</td>
<td>2,492,216</td>
<td>0.104</td>
</tr>
</tbody>
</table>

5.5.2 Experiment Setup

We experimented with four different privacy budgets, $\epsilon \in \{0.2, 0.3, \ln(2), \ln(3)\}$; for the larger Pokec dataset, we used $\epsilon \in \{0.01, 0.05, 0.1, 0.2\}$. For AGM-DPTriCL, we divided the overall privacy budget $\epsilon$ evenly among the four model parameters: attribute distribution $\Theta_X$, attribute correlations $\Theta_F$, degree sequence $\mathcal{S}$ and number of triangles $n_\Delta$. For AGMDP-FCL, which does not use $n_\Delta$, we allocated half of the budget to $\mathcal{S}$ and split the other half evenly between $\Theta_F$ and $\Theta_X$. The truncation parameter, $k$, used in computing the private attribute correlations, was set to $n^{\frac{1}{3}}$ (see Section 5.3.1.5) for all experiments. For both AGMDP-TriCL and AGMDP-FCL,

\textsuperscript{16}http://snap.stanford.edu/data/
on each setting of $\epsilon$, we report results average over 1,000 synthetic graphs (100 for Epinions and Pokec). For reference, we also report the results of AGM instantiated with the non-private versions of FCL and TriCycLe, denoted AGM-FCL and AGM-TriCL, respectively.

### 5.5.3 Statistics Evaluated

We compared the synthetic graphs to the corresponding input graphs in terms of the following statistics:

**Clustering Coefficients.** As described earlier, social graphs typically exhibit large amounts of clustering among nodes (compared to chance), and TriCycLe was designed to capture this clustering via a triangle creation phase. We consider two global measures of clustering. The *global clustering coefficient* (also called *transitivity*) of a graph $G$ is defined as $C(G) = \frac{3 \times n_{\Delta}}{n_{W}}$ where $n_{\Delta}$ is the number of triangles in $G$ and $n_{W}$ is the number of wedges (i.e., length two paths) in $G$. An alternate characterization of the amount of clustering in a graph is the average of the *local clustering coefficients*. The local clustering coefficient of a node $v_i \in N$ is given by $C_i = \frac{2|\{(e,j,k)\in E|v_j,v_k\in Nbr(v_i)\}|}{|Nbr(v_i)| \times ([|Nbr(v_i)|]-1)}$, and the average of the local clustering coefficients is denoted $\overline{C} = \frac{1}{n} \sum_{v_i \in N} C_i$. The reason for considering both of these measures is that the former tends to emphasize the low-degree nodes while the latter emphasizes the high-degree nodes. We report the *mean relative error* (MRE) of $C$ and $\overline{C}$ on the synthetic graphs relative to the original graphs (denoted as $C$ and $\overline{C}$ in the tables).

**Degree Distribution.** To evaluate how well a synthetic graph captures the degree distribution of the input graph, we use the *Kolmogorov-Smirnov* (KS) statistic, which quantifies the maximum distance between the two degree distributions. Let $F_S$ and $\overline{F_S}$ denote the cumulative distribution functions estimated from the sorted degree sequences of the original and synthetic graphs, respectively. Then $\text{KS}(S, \overline{S}) = \max_d |F_S(d) - \overline{F_S}(d)|$. Additionally, since the KS statistic is known to be less sensitive to differences in the tails of the distributions, we also report the *Hellinger distance* between the two degree distributions (denoted $D_S, D_S$), defined
as \( H_S = \frac{1}{\sqrt{2}} \sqrt{\sum_d \left( \sqrt{D_S(d)} - \sqrt{\tilde{D}_S(d)} \right)^2} \). The smaller the values of both statistics, the closer (more similar) the degree distributions of the synthetic and original graphs.

**Edge Count and Triangle Count.** We report the mean relative error (MRE) for the number of edges (denoted simply as \( m \) in the tables) and the number of triangles (denoted as \( n_{\Delta} \) in the tables) in the synthetic graphs relative to the original input graphs.

**Attribute Correlations.** To quantify the error in the attribute correlations in the synthetic graph, relative to the original graph, we report the MRE (denoted as simply \( \Theta_F \) in the tables), as well as the Hellinger distance, defined as \( H_{\Theta_F} = \frac{1}{\sqrt{2}} \sqrt{\sum_{y_i \in Y_w^F} \left( \sqrt{\Theta_F(y_i)} - \sqrt{\tilde{\Theta}_F(y_i)} \right)^2} \), where \( \Theta_F \) and \( \tilde{\Theta}_F \) are the discrete probability distributions for the original and synthetic graphs, respectively. \( H_{\Theta_F} \) lies in the range \([0, 1]\), with a value near 0 meaning that the correlations in the synthetic graph closely approximate those in the input graph.

**5.5.4 Results**

Tables 5.3–5.6 summarize the experimental results on the four datasets. For each setting of the overall privacy budget (\( \epsilon \)) we report the average error rates for the graphs synthesized by both AGMDP-TriCL and AGMDP-FCL. Recall that a smaller \( \epsilon \) translates to a stronger privacy guarantee; so, the level of privacy (and, in general, the amount of error) increases as we move down from top to bottom in the tables. To make the impact of the privacy mechanism clear, we include, at the top of each table, the error rates for the non-private version of AGM instantiated with FCL and TriCycLe (denoted AGM-FCL and AGM-TriCL, respectively).

**Clustering.** Recall that the FCL structural model includes no mechanism for capturing and reproducing the clustering in the input graph, so the error rates for the triangle count (\( n_{\Delta} \)), average clustering coefficient (\( \bar{C} \)) and the global clustering coefficient (\( C \)) are larger for the FCL-based models than for the TriCycLe-based models, as expected. For that reason, the error rates for AGMDP-FCL on those three statistics serve as a good reference point for interpreting the corresponding results of AGMDP-TriCL. For instance, looking at the results on the Last.fm and
Table 5.3 Results for AGM-DP with the FCL (AGMDP-FCL) and TriCycLe (AGMDP-TriCL) structural models on the **Last.fm** dataset for different privacy settings, $\epsilon$.

<table>
<thead>
<tr>
<th>Last.fm</th>
<th>Model</th>
<th>$\Theta_F$</th>
<th>$H_{\Theta_F}$</th>
<th>$KS_S$</th>
<th>$H_S$</th>
<th>$n_\Delta$</th>
<th>$\overline{C}$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-private</td>
<td>AGM-FCL</td>
<td>0.00</td>
<td>0.01</td>
<td>0.05</td>
<td>0.15</td>
<td>0.59</td>
<td>0.75</td>
<td>0.61</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.00</td>
<td>0.02</td>
<td>0.08</td>
<td>0.16</td>
<td>0.05</td>
<td>0.04</td>
<td>0.24</td>
<td>0.0001</td>
</tr>
<tr>
<td>$\epsilon = \ln 3$</td>
<td>AGMDP-FCL</td>
<td>0.02</td>
<td>0.14</td>
<td>0.06</td>
<td>0.16</td>
<td>0.59</td>
<td>0.75</td>
<td>0.60</td>
<td>0.0076</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.02</td>
<td>0.14</td>
<td>0.09</td>
<td>0.17</td>
<td>0.05</td>
<td>0.07</td>
<td>0.23</td>
<td>0.0147</td>
</tr>
<tr>
<td>$\epsilon = \ln 2$</td>
<td>AGMDP-FCL</td>
<td>0.03</td>
<td>0.18</td>
<td>0.07</td>
<td>0.17</td>
<td>0.56</td>
<td>0.74</td>
<td>0.58</td>
<td>0.0120</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.03</td>
<td>0.18</td>
<td>0.10</td>
<td>0.18</td>
<td>0.06</td>
<td>0.10</td>
<td>0.23</td>
<td>0.0222</td>
</tr>
<tr>
<td>$\epsilon = 0.3$</td>
<td>AGMDP-FCL</td>
<td>0.05</td>
<td>0.27</td>
<td>0.09</td>
<td>0.19</td>
<td>0.42</td>
<td>0.68</td>
<td>0.48</td>
<td>0.0248</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.05</td>
<td>0.28</td>
<td>0.12</td>
<td>0.21</td>
<td>0.18</td>
<td>0.30</td>
<td>0.24</td>
<td>0.0499</td>
</tr>
<tr>
<td>$\epsilon = 0.2$</td>
<td>AGMDP-FCL</td>
<td>0.06</td>
<td>0.32</td>
<td>0.11</td>
<td>0.20</td>
<td>0.39</td>
<td>0.65</td>
<td>0.43</td>
<td>0.0374</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.06</td>
<td>0.33</td>
<td>0.16</td>
<td>0.24</td>
<td>0.35</td>
<td>0.38</td>
<td>0.27</td>
<td>0.0769</td>
</tr>
</tbody>
</table>

Table 5.4 Results for AGMDP-FCL AGMDP-TriCL on the **Petster** dataset for different privacy settings, $\epsilon$.

<table>
<thead>
<tr>
<th>Petster</th>
<th>Model</th>
<th>$\Theta_F$</th>
<th>$H_{\Theta_F}$</th>
<th>$KS_S$</th>
<th>$H_S$</th>
<th>$n_\Delta$</th>
<th>$\overline{C}$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-private</td>
<td>AGM-FCL</td>
<td>0.00</td>
<td>0.02</td>
<td>0.04</td>
<td>0.17</td>
<td>0.13</td>
<td>0.51</td>
<td>0.15</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.00</td>
<td>0.01</td>
<td>0.05</td>
<td>0.17</td>
<td>0.00</td>
<td>0.35</td>
<td>0.07</td>
<td>0.0000</td>
</tr>
<tr>
<td>$\epsilon = \ln 3$</td>
<td>AGMDP-FCL</td>
<td>0.03</td>
<td>0.16</td>
<td>0.05</td>
<td>0.17</td>
<td>0.18</td>
<td>0.51</td>
<td>0.19</td>
<td>0.0075</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.03</td>
<td>0.16</td>
<td>0.06</td>
<td>0.18</td>
<td>0.09</td>
<td>0.26</td>
<td>0.08</td>
<td>0.0136</td>
</tr>
<tr>
<td>$\epsilon = \ln 2$</td>
<td>AGMDP-FCL</td>
<td>0.03</td>
<td>0.20</td>
<td>0.06</td>
<td>0.18</td>
<td>0.16</td>
<td>0.49</td>
<td>0.17</td>
<td>0.0117</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.04</td>
<td>0.20</td>
<td>0.07</td>
<td>0.19</td>
<td>0.13</td>
<td>0.29</td>
<td>0.10</td>
<td>0.0221</td>
</tr>
<tr>
<td>$\epsilon = 0.3$</td>
<td>AGMDP-FCL</td>
<td>0.05</td>
<td>0.29</td>
<td>0.08</td>
<td>0.20</td>
<td>0.21</td>
<td>0.44</td>
<td>0.18</td>
<td>0.0268</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.06</td>
<td>0.30</td>
<td>0.12</td>
<td>0.22</td>
<td>0.39</td>
<td>0.32</td>
<td>0.24</td>
<td>0.0486</td>
</tr>
<tr>
<td>$\epsilon = 0.2$</td>
<td>AGMDP-FCL</td>
<td>0.07</td>
<td>0.35</td>
<td>0.10</td>
<td>0.21</td>
<td>0.32</td>
<td>0.41</td>
<td>0.24</td>
<td>0.0388</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.07</td>
<td>0.36</td>
<td>0.14</td>
<td>0.24</td>
<td>0.72</td>
<td>0.32</td>
<td>0.40</td>
<td>0.0766</td>
</tr>
</tbody>
</table>
Table 5.5 Results for AGMDP-FCL AGMDP-TriCL on the Epinions dataset for different privacy settings, $\epsilon$.

<table>
<thead>
<tr>
<th>Epinions</th>
<th>Model</th>
<th>$\Theta_F$</th>
<th>$H_{\Theta_F}$</th>
<th>$KS_S$</th>
<th>$H_S$</th>
<th>$n_\Delta$</th>
<th>$\overline{C}$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-private</td>
<td>AGM-FCL</td>
<td>0.01</td>
<td>0.03</td>
<td>0.06</td>
<td>0.08</td>
<td>0.82</td>
<td>0.92</td>
<td>0.83</td>
<td>0.0008</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
<td>0.08</td>
<td>0.27</td>
<td>0.06</td>
<td>0.53</td>
<td>0.0008</td>
</tr>
<tr>
<td>$\epsilon = \ln 3$</td>
<td>AGM-FCL</td>
<td>0.01</td>
<td>0.06</td>
<td>0.06</td>
<td>0.08</td>
<td>0.83</td>
<td>0.92</td>
<td>0.84</td>
<td>0.0019</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.01</td>
<td>0.06</td>
<td>0.04</td>
<td>0.08</td>
<td>0.28</td>
<td>0.10</td>
<td>0.53</td>
<td>0.0057</td>
</tr>
<tr>
<td>$\epsilon = \ln 2$</td>
<td>AGM-FCL</td>
<td>0.01</td>
<td>0.08</td>
<td>0.05</td>
<td>0.08</td>
<td>0.82</td>
<td>0.92</td>
<td>0.84</td>
<td>0.0042</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.01</td>
<td>0.09</td>
<td>0.04</td>
<td>0.09</td>
<td>0.28</td>
<td>0.11</td>
<td>0.52</td>
<td>0.0084</td>
</tr>
<tr>
<td>$\epsilon = 0.3$</td>
<td>AGM-FCL</td>
<td>0.01</td>
<td>0.12</td>
<td>0.06</td>
<td>0.10</td>
<td>0.81</td>
<td>0.92</td>
<td>0.83</td>
<td>0.0128</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.02</td>
<td>0.14</td>
<td>0.07</td>
<td>0.11</td>
<td>0.23</td>
<td>0.14</td>
<td>0.48</td>
<td>0.0218</td>
</tr>
<tr>
<td>$\epsilon = 0.2$</td>
<td>AGM-FCL</td>
<td>0.02</td>
<td>0.13</td>
<td>0.08</td>
<td>0.11</td>
<td>0.78</td>
<td>0.91</td>
<td>0.80</td>
<td>0.0157</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.02</td>
<td>0.17</td>
<td>0.09</td>
<td>0.12</td>
<td>0.20</td>
<td>0.19</td>
<td>0.46</td>
<td>0.0345</td>
</tr>
</tbody>
</table>

Epinions datasets (Tables 5.3 and 5.5, respectively) we can see that even with the reasonably strong privacy budget of $\epsilon = 0.2$, AGMDP-TriCL was able to reproduce the clustering observed in the input graph significantly more accurately than AGMDP-FCL, which places edges solely according to the (noisy) degree sequence. For the large Pokec dataset, the error rates remained significantly below those of AGMDP-FCL, even with the extremely strong privacy guarantee provided by $\epsilon = 0.01$, which is typically the smallest $\epsilon$ used in the differential privacy literature. In fact, the error rates for those statistics remained nearly unchanged (compared to the non-private counterpart) until $\epsilon$ was set $\leq 0.05$. This is due to the fact that the amount of noise added to preserve privacy is independent of the size of the input graph; thus, larger graphs achieve a more favorable signal-to-noise ratio for the model parameters. On the other hand, we observe that on the smaller Petster dataset, the clustering produced by AGMDP-TriCL was no better than that produced by AGMDP-FCL, when $\epsilon < \ln(2)$ ($\approx 0.69$). It is worth noting, however, that the error rates seen for the FCL-based models, with respect to the clustering related statistics, on the small Last.fm and Petster datasets, were far better than those seen for AGMDP-FCL on the larger Epinions and Pokec datasets—on the latter two datasets, the clustering coefficients were close to zero, leading to error rates near 1.0; in contrast, the smaller
size of the Last.fm and Petster datasets made it possible for a significant number of triangles to arise naturally. Consequently, the synthetic graphs generated for those datasets with small $\epsilon$’s are still likely to be useful.

**Attribute-Edge Correlations.** As another reference point for interpreting the error observed in the attribute-edge correlations ($H_{\Theta_F}$ and $\Theta_F$ in the tables), we note that a baseline approach that simply sets all of the correlation probabilities to be equal$^{17}$, rather than actually estimating them from the true probabilities, yields an average Hellinger distance of 0.37, 0.45, 0.55 and 0.5 on Last.fm, Petster, Epinions and Pokec, respectively. Similarly, we note that the MAE of the baseline probabilities relative to the true probabilities was 0.09, 0.11, 0.13 and 0.12, on the four datasets respectively. Comparing against $H_{\Theta_F}$ and $\Theta_F$ in the tables, we see that the error rates for the synthetic graphs generated by AGMDP-FCL and AGM-TriCL were significantly lower than the baseline error rates on all datasets, even under the strongest privacy settings tested. The difference was most dramatic on the larger datasets (Epinions and Pokec).

**Degree Statistics.** Regarding the errors in the degree statistics ($H_S$ and $K_S$), we first note that the reason that the error appears to grow more slowly for AGMDP-FCL than for AGMDP-TriCL (as $\epsilon$ decreases) is that we allocated one-half of the privacy budget for computing the noisy degree sequence with AGMDP-FCL, while for AGMDP-TriCL we allocated only one-fourth (the other one-fourth was used to estimate the triangle count, which is not used by FCL). Thus, the degree sequence received more noise in AGMDP-TriCL for the same $\epsilon$. Next, to make the error-rates more meaningful, we note that the KS statistic for a baseline model that assigns edges to nodes uniformly at random is 0.51 on Last.fm and Petster, and 0.61 and 0.43 on Epinions and Pokec, respectively. Similarly, the baseline Hellinger distance is 0.64, 0.63, 0.64 and 0.56 on the four datasets, respectively. We see from the tables that for both AGMDP-FCL and AGMDP-TriCL, with the strongest privacy settings tested, the KS statistic was substantially below that of the baseline: the KS was $\leq 0.16$ on Last.fm and Petster (Tables 5.3 and 5.4), and $\leq 0.09$ on Epinions and Pokec (Tables 5.5 and 5.6). Similarly, the Hellinger

---

$^{17}$That is, for $w = 2$ attributes, we set each of the ten probabilities to 0.1.
distance for both models was less than half that of the baseline on all datasets: on Last.fm and Petster, we had $H_S \leq 0.24$, and on Epinions and Pokec we had $H_S \leq 0.12$.

The number of edges in the synthetic graphs ($m$) also remained reasonably close to that of the original graphs under the smallest privacy budgets tested ($\epsilon = 0.2$ for Last.fm, Petster and Epinions, and $\epsilon = 0.01$ for Pokec) with the MRE below 0.08 on the small Last.fm and Petster datasets, below 0.04 on Epinions and below 0.085 on Pokec.

**Other Observations.** In some cases (particularly the smaller datasets), the error rates observed for the triangle count ($n_\Delta$) and clustering statistics ($C$ and $\bar{C}$) were somewhat counter-intuitive, but can be explained. With AGMDP-FCL, we observed that on the smaller datasets, the error for the triangle count, and in some cases the clustering statistics, appeared to decrease with $\epsilon$ (i.e., with increasing privacy), which is not what we might expect. This appears to be a side-effect of the post-processing procedure. We noted earlier (Section 5.3.3.3) that the post-processing routine introduces a slight negative bias into the triangle count because it rewire a subset of the existing edges in order to reattach orphaned nodes to the main connected component, which inevitably destroys some of the existing triangles while creating very few new ones, since most of the orphaned nodes have a desired degree of one. When noise is injected into the connection counts to satisfy differential privacy for the correlation parameters $\Theta_F$, the small probability values become larger, spreading the triangles out over more of the nodes; with less overlap among the triangles, fewer triangles are destroyed by the edge removals during post-processing, which in-turn counteracts some of that bias.

A similar phenomenon was also observed for AGMDP-TriCL in some instances, but to a lesser extent, due to the independent noise that is introduced into the triangle count statistic (which is one of the inputs to TriCycLe). In particular, on Epinions the error rates for $n_\Delta$ and $C$ decreased slightly when $\epsilon < \ln(2)$. 

137
Table 5.6 Results for AGMDP-FCL AGMDP-TriCL on the Pokec dataset for different privacy settings, $\epsilon$.

<table>
<thead>
<tr>
<th>Pokec</th>
<th>Model</th>
<th>$\Theta_F$</th>
<th>$H_{\Theta_F}$</th>
<th>$KS_S$</th>
<th>$H_S$</th>
<th>$n_\Delta$</th>
<th>$\overline{C}$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-private</td>
<td>AGM-FCL</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
<td>0.05</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>AGM-TriCL</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
<td>0.04</td>
<td>0.24</td>
<td>0.37</td>
<td>0.30</td>
<td>0.0000</td>
</tr>
<tr>
<td>$\epsilon = 0.2$</td>
<td>AGMDP-FCL</td>
<td>0.00</td>
<td>0.02</td>
<td>0.05</td>
<td>0.05</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.0020</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.00</td>
<td>0.03</td>
<td>0.03</td>
<td>0.05</td>
<td>0.24</td>
<td>0.39</td>
<td>0.30</td>
<td>0.0052</td>
</tr>
<tr>
<td>$\epsilon = 0.1$</td>
<td>AGMDP-FCL</td>
<td>0.00</td>
<td>0.03</td>
<td>0.05</td>
<td>0.06</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.00</td>
<td>0.04</td>
<td>0.05</td>
<td>0.07</td>
<td>0.24</td>
<td>0.37</td>
<td>0.30</td>
<td>0.0101</td>
</tr>
<tr>
<td>$\epsilon = 0.05$</td>
<td>AGMDP-FCL</td>
<td>0.00</td>
<td>0.06</td>
<td>0.05</td>
<td>0.07</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.0108</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.01</td>
<td>0.09</td>
<td>0.07</td>
<td>0.10</td>
<td>0.25</td>
<td>0.35</td>
<td>0.33</td>
<td>0.0189</td>
</tr>
<tr>
<td>$\epsilon = 0.01$</td>
<td>AGMDP-FCL</td>
<td>0.02</td>
<td>0.14</td>
<td>0.11</td>
<td>0.15</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.0510</td>
</tr>
<tr>
<td></td>
<td>AGMDP-TriCL</td>
<td>0.03</td>
<td>0.20</td>
<td>0.15</td>
<td>0.19</td>
<td>0.46</td>
<td>0.37</td>
<td>0.50</td>
<td>0.0833</td>
</tr>
</tbody>
</table>

5.5.5 Discussion

Our empirical results presented in the previous section demonstrate that it is possible to generate synthetic attributed social graphs with reasonably high fidelity, without sacrificing privacy. An important takeaway from the results is that, while the proposed framework performs well even on very small graphs, its power really becomes evident when applied on moderate to large input graphs. In particular, we saw that on the reasonably large Pokec social network, we were able to generate very accurate synthetic graphs even under the strongest privacy regime typically used to evaluate differentially private systems in the literature (i.e., $\epsilon = 0.01$).

We focused our empirical analysis on graphs with $w = 2$ node attributes, however our framework is not limited to two-dimensional attribute vectors. Although the sensitivity of the approach is not affected by the number of attributes, we can expect the error rates to increase as $w$ increases, since the number of counts that must be computed for $\Theta_X$ and $\Theta_F$ increase exponentially with $w$, leading to smaller counts and hence a higher noise-to-signal ratio. Likewise, we would expect the error rates to decrease when using $w = 1$.  

138
5.6 Related Work

Prior work on differential privacy for network data has generally pursued one of two directions: methods for privately computing specific statistics over sensitive graphs, and private graph publication. We provide an overview of the recent work along both directions below. We also summarize some very recent work that attempt to address some of the hurdles to achieving node differential privacy. A review of differential privacy and a thorough summary of its application in other areas was provided in Chapter 2.

Hay et al. [44] proposed an approach for releasing the degree distribution of a graph under edge-differential privacy using a novel constrained inference technique. The idea is to impose an ordering constraint on the pre-noise degree sequence, add Laplace noise to the sorted sequence and then post-process the noisy sequence so that it also satisfies the order constraint. They provide an efficient implementation based on dynamic programming and show that the approach leads to a very accurate degree distribution when the degree sequence contains long subsequences of uniform degrees. Another graph statistic that has recently been studied in the context of differential privacy is subgraph counting (e.g., [58]). A subgraph counting query over an input graph $G$ consists of a query graph $Q$ (e.g., triangle), and the goal is to return a count of the number of “edge-induced isomorphic copies” of $Q$ that are present in $G$ [58]. Although subgraph counting suffers from unbounded global sensitivity in general, Karwa et al. show that the smooth sensitivity can be much lower and present an approach that achieves edge-differential privacy for triangle counting [58]. They also present an approach for $k$-triangle ($k$ triangles that share a common edge) counting that achieves edge privacy under the weaker notion of $(\epsilon, \delta)$-differential privacy; however, this algorithm is NP-Complete in general [58]. Blocki et al. introduced the notion of restricted sensitivity, which enables statistics such as subgraph counting to be answered under differential privacy with less noise [10]. The idea is to apply a low-sensitivity transformation to a graph (e.g., restrict the maximum node degree) leading to a lower global sensitivity for the transformed graph. Recently, Zhang et al. [139] introduced the
Ladder framework for producing highly accurate, differentially private estimates of subgraph counting queries, including triangles and $k$-stars. The Ladder framework is a novel combination of the concept of “local sensitivity at distance $t$”, from the smooth sensitivity framework [98], with the exponential mechanism [88]. Compared to smooth sensitivity, it is often more accurate and additionally has the advantage of satisfying pure differential privacy, as opposed to the weaker $(\epsilon, \delta)$-differential privacy. Machanavajjhala et al. studied edge-differential privacy for the task of personalized social recommendation, where the input graph consists of users and items, and edges between users denote social relationships, while edges between users and items represent item preferences [82]. They showed that generating recommendations that are both private and accurate is not possible, in general. Our work in Chapter 3 and [56], relaxes the assumption that the social edges are private and presents an approach that is able to achieve edge privacy for item preferences without significantly harming utility. Recently, Wang et al. outlined a general, divide and conquer approach utilizing smooth sensitivity to achieve better utility for graph analysis tasks that have low local sensitivity [126]. They demonstrated their approach by providing an algorithm for computing the clustering coefficient under edge $(\epsilon, \delta)$-differential privacy. Shen and Yu [113] study the problem of finding frequently occurring subgraphs under differential privacy. That is, given a multiset of graphs, the basic task is to find subgraph patterns that frequently occur within the graphs of the multiset. They consider neighboring datasets to be graph multisets that differ in the presence of a single graph and propose a technique based on Markov Chain Monte Carlo (MCMC) sampling that achieves privacy while maintaining high precision.

Another active line of work focuses on publishing graph data under differential privacy with the goal of retaining enough structural information to enable useful analyses on the published graph [91, 110, 125, 19, 129, 103, 80]. Mir and Wright [91] focused on the problem of generating representative synthetic graphs for input graphs that can be modeled well by the Kronecker graph model. Their approach estimates the model parameter from the input graph under edge-differential privacy, which then defines a probability distribution on graphs, from
which a representative synthetic graph can be sampled. Sala et al. introduced an edge-private graph model called *Pygmalion* that constructs a synthetic graph by introducing noise into the $dK$-series of the input graph [110]. They use a degree-based clustering approach to reduce the amount of noise required to achieve differential privacy. Unfortunately, the noise injected by their approach is scaled according to the local sensitivity, rather than global sensitivity; consequently their approach may leak information about the structure of the input graph and does not strictly satisfy differential privacy. Wang and Wu also use the $dK$-series model to generate synthetic graphs; however, in contrast to [110] they add noise according to the smooth sensitivity, thereby achieving $(\epsilon, \delta)$-differential privacy [125]. Very recently, Xiao et al. proposed to encode the structure of a graph as a series of private edge-counting queries, under the hierarchical random graph (HRG) model [129]. They used the Markov Monte Carlo Method (MMCM) to efficiently sample an HRG under differential privacy, which can then be used to generate a synthetic graph similar in structure to the input graph. This approach is shown empirically to significantly out-perform the dK-series based approach of [125]. Chen et al. [19] proposed to privately publish an input graph by first relabeling the vertices, in a private way, such that the corresponding adjacency matrix contains dense clusters, which are identified through a data-dependent partitioning process. A noisy version of the adjacency matrix is then reconstructed using the exponential mechanism. Proserpio et al. [103] proposed non-uniformly down-weighting the edges of a graph in order to get around the high global sensitivity due to the possibility of very high degree nodes. They demonstrate the approach, combined with MCMC-based sampling, to generate private synthetic graphs. Our work in this chapter differs from all of the above prior works in that we focus on the problem of generating synthetic graphs that mimic both the structure and the attribute correlations observed in a social network under edge-differential privacy.

Although the bulk of prior work has focused on edge-differential privacy, there have been some recent advances toward satisfying node-differential privacy while retaining a non-trivial level of utility. In node-differential privacy, neighboring graphs differ in a single node and all of
its incident edges. Smooth sensitivity has proven to be useful for dealing with the high global sensitivity of many tasks in the context of edge-differential privacy; however, in the context of node differential privacy, smooth sensitivity is often ineffective. The difficulty stems from the fact that local sensitivity, like global sensitivity, depends in part on the worst-case effect of adding an arbitrary node to the input graph; in the worst-case, the added node could be incident to all of the edges in the graph. For example, under node privacy, the simple task of counting the number of edges in a graph has a local sensitivity equal to the number of nodes in the input, which is also the same as the global sensitivity. Chen and Zhou proposed the recursive mechanism that is based on the notion of local empirical sensitivity, which considers only the effects of the deletion of an existing node from the input graph [20]. Kasiviswanathan et al. [59] and Blocki et al. [10] independently proposed the idea of using low-sensitivity transforms to project an input graph onto the set of degree-bounded graphs. By bounding the maximum degree of the input graph, the sensitivity becomes bounded and can be significantly reduced. When the degree bound, $D$, is set appropriately, so that most of the nodes in the graph have degree $\leq D$, very little error is introduced by the projection operation; however, choosing an appropriate bound may be difficult in practice. We plan to investigate the possibility of generating attributed social networks under node-differential privacy in a future work.

5.7 Summary

This work proposed a differentially private adaptation of the Attributed Graph Model (AGM) of [101]. Our framework is capable of modeling and synthesizing attributed social networks that mimic the structural properties and attribute correlations of an input network, without disclosing individual relationships and attributes. We explored three different approaches for accurately modeling the attribute correlations under differential privacy, identifying edge truncation to be the most effective choice. We introduced a new structural model capable of reproducing the important structural properties of social networks, including the degree distribution.
and the distribution of local clustering coefficients, and showed that the parameters required by the model can be accurately estimated under differential privacy. Finally, we presented an end-to-end workflow for generating private, synthetic social graphs with our framework and demonstrated its efficacy through experiments on four real-world social network datasets.

We believe this work lays a good foundation for providing differential privacy guarantees for attributed graph analysis; however, there remain many avenues for future work. We limited our focus to undirected graphs with a relatively small number of binary node attributes. Addressing the hurdles associated with supporting directed edges and larger numbers of attributes (which were discussed in Section 5.4.3) would be a worthwhile direction. Second, although we have made some progress in extending the framework to support the stronger notion of node-differential privacy, more work is needed to develop an appropriate structural model that supports node privacy and that can be plugged into the framework. Other possible directions include incorporating support for edge attributes (e.g., trust or friendship strength) and improvements to the TriCycLe structural model.
Chapter 6

Conclusions and Future Research

In the decade since it was first proposed, differential privacy has become the gold standard for privacy-preserving data analysis and an active research area in its own right. Despite many advances, achieving strong differential privacy guarantees for many types of analysis, while preserving the utility of the output, remains a significant challenge. Our work in this dissertation was aimed at making differentially private data analysis more practical in terms of the trade-off between privacy and utility.

6.1 Review of Contributions

We began by tackling the problem of providing useful social recommendations to users based of the item preferences of their social network neighbors, while simultaneously protecting the privacy of the individuals in the social network. Our work was motivated by an earlier study [82] that provided rather negative results for incorporating differential privacy into this task. In particular, the authors of that study concluded that it was not possible to generate useful social recommendations while providing reasonably strong differential privacy guarantees; however, the model considered in the study made no distinction between item preferences and social relations. By contrast, our work presented a new formalization of the social recommendation
problem that explicitly differentiates user nodes from item nodes and social relations from item preferences; this distinction enabled us to decouple the computations on the typically insensitive social graph from those on the often sensitive item preferences. Despite relaxing the privacy requirements for the social relations, satisfying differential privacy for the item preferences remains a significant challenge. Depending on the underlying social similarity measure used, a single user’s item preferences may affect the recommendations received by many users. Moreover, the recommendation system was required to generate recommendations for every user in the network, with nothing preventing those users from colluding to uncover the preferences of a mutual neighbor. These factors contribute to a high global sensitivity, which would require an enormous amount of noise to be injected using existing differentially private mechanisms. We developed a differentially private framework that achieves strong privacy and high utility for a natural class of social recommendation algorithms by incorporating a novel user clustering phase that groups users according to the social graph structure, significantly reducing sensitivity, and hence the amount of noise required to satisfy differential privacy. Our empirical results on two real-world data sets, show that it is possible to make accurate social recommendations while simultaneously providing a high degree of privacy for item preferences.

Our second major contribution was the development of a novel personalized variant of differential privacy that enables per-user privacy guarantees for aggregate analyses and boosts utility by providing only the level of privacy required by each individual in a dataset. Personalized differential privacy (PDP) was motivated by the observation that, in practice, data privacy is a personal and multifaceted concept, and that different individuals often have very different expectations for the privacy of their personal data. In contrast to the single privacy parameter provided by standard differential privacy (i.e., $\epsilon$), PDP allows every user to configure a personal privacy parameter for their own data. In addition to formalizing PDP and its properties, we developed two mechanisms for satisfying PDP. Our primary mechanism takes advantage of the privacy amplification effects of random sampling and has the advantage of being very general—it can be used to automatically transform any existing differentially private computation into
one that satisfies PDP. Our second mechanism is a more direct approach inspired by the exponential mechanism [88]. It is applicable to common aggregates such as counts, medians, and min/max and is shown to outperform the sampling-based mechanism for certain aggregates that are especially sensitive to the effects of sampling. Our extensive empirical study showed that taking personal privacy preferences into account enables significantly more accurate outputs in many cases, since we are not “paying” for more privacy than we need.

Our final work aimed to make privacy-preserving analysis of sensitive social networks more practical. One of the challenges of adapting differential privacy to graph data is that determining the correct global sensitivity for a complex graph analysis task can often be challenging, error-prone and time consuming. Moreover, some tasks involve running simulations over a graph or conducting exploratory analyses that are iterative in nature; these can be very difficult to accomplish in a differentially private way, given a limited privacy budget. Motivated by these difficulties, we explored an alternate approach to enabling differentially private graph analysis. We developed an end-to-end framework capable of privately modeling an input social graph and generating a synthetic graph that preserves differential privacy and that can be used as a proxy for the original input graph in a variety of analyses. Unlike related prior works, which focused on generating private synthetic graphs that mimic only the structural properties of an input graph, our framework is capable of capturing and reproducing not only the structure but also the distribution of nodal attributes and their correlations with the graph structure. Consequently, our framework produces more realistic output graphs that can be used in a wider variety of analyses and simulations. We demonstrated the effectiveness of our approach by generating synthetic versions of four real-world social network datasets under a variety of privacy regimes. We quantified the error in the synthetic graphs by comparing a variety of relevant graph statistics measured on both the synthetic and original graphs and we found that our framework was able to produce synthetic graphs with high fidelity, even under very strong privacy regimes. Moreover, we were able to show that the error of our approach decreases with the size of the input graph.
6.2 Future Work

We have already provided some discussion of possible extensions to the projects presented in this dissertation, in their respective chapters. Here we take a step back and provide some recommendations for future work around differential privacy in general.

6.2.1 Graph and Network Analysis

Social network analysis (SNA) has become a hot area for differential privacy research, and it will likely continue to receive attention for some time, given the abundance of social data and its inherent value. Although great strides have been made in recent years toward making differential privacy effective in this domain, there is still much work to be done. First, much of the existing work, including our work in this dissertation, has focused on providing edge-differential privacy. Although edge-differential privacy is generally believed to provide reasonably strong privacy, a perhaps more desirable notion of privacy for graphs is node-differential privacy. Until very recently, it was unclear whether achieving node-differential privacy while maintaining a non-trivial level of utility was feasible; however, several recent works [59, 10] have provided a glimmer of hope. These works center around the idea of projecting an input graph onto the space of degree-bounded graphs that can be analyzed with fixed sensitivity. It remains to be seen whether there are other types of low-sensitivity transformations that can be applied to further reduce the noise required to satisfy node privacy.

Second, the tasks of community detection and clustering are important in SNA but very difficult to accommodate privately. Although approaches for releasing synthetic graphs (such as our work in Chapter 5) could allow us to learn some high-level information about the community structure of a graph, it would not allow us to determine which specific nodes are part of the same community, for example. It is not clear whether this is achievable under differential privacy or if a more relaxed privacy definition would be necessary.

Third, the approach of releasing synthetic graphs that preserve differential privacy is a
promising approach to supporting complex analyses on sensitive graphs. There has been some recent work focusing on learning various structural models under differential privacy, but there is room for improvement. In Chapter 5, we pointed out the lack of effective differentially private models for generating realistic social networks. We proposed a new model, TriCycLe, that takes a positive step toward that goal but leaves room for improvement. In particular, many modern social networks support directed relationships among users (e.g., followers in Twitter) and being able to model direct edges privately would be desirable.

Finally, it has been argued [64] that differential privacy may not be the most appropriate definition for preserving privacy in graphs, particularly social networks, which are highly dynamic. The argument is that there may be correlations between edges that are not apparent from a static snapshot of a dynamic graph. For example, in a social network, the formation of an edge between two individuals may lead to the subsequent formation of edges between neighbors of those individuals over time. In other words, an edge at one time step can have a long term influence on the future evolution of the graph. [64] argues that to fully protect privacy in social networks, we must be able to mask the participation of a node or edge in the graph’s evolution. To the best of our knowledge, concrete attacks successfully exploiting this observation have yet to be demonstrated, nor have any solutions been proposed. It is worth noting that this issue is not limited to graph data, but the dynamic nature of social graphs makes these types of correlations between edges more likely.

6.2.2 Differential Privacy in Practice

A long-standing hurdle to applying differential privacy in practice is that it can be difficult to know what is an appropriate choice for $\epsilon$; justifying that choice to non-experts can be harder still. So far, there has been very little discussion of this issue in the literature. Can we develop new intuition and insights for guiding this parameter choice within important data domains and classes of analyses? Can the privacy provided by differential privacy be understood within the existing legal frameworks that govern acceptable privacy risks (e.g., HIPPA)?
6.2.3 Weaker Notions of Privacy

It could be argued that for some tasks, differential privacy is unnecessarily strong. For example, in practice it seems unlikely that an adversary would have full knowledge of all but one record in a dataset. Can we relax this assumption in some way to achieve a more favorable privacy-utility trade-off? A very recent work by Tramèr et al. [121] takes a step in this direction by proposing a weaker characterization of differential privacy that provides better utility by weakening the adversary’s assumed prior belief about the contents of the input database. Specifically, the adversary’s prior knowledge regarding the presence of any tuple $t$ in a database is assumed to be in the range $[a, b]$ for $0 < a \leq b < 1$ (or that the adversary already has full knowledge of $t$’s presence or absence).
REFERENCES


