THE PENNSYLVANIA STATE UNIVERSITY

SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DETERMINING OPTIMAL TRUNCATION PARAMETERS OF ALGORITHMS FOR RELEASING GRAPH STATISTICS UNDER NODE-DIFFERENTIAL PRIVACY

SUJEET BHANDARI

SPRING 2015

A thesis

submitted in partial fulfillment

of the requirements

for baccalaureate degrees

in Computer Science and Electrical Engineering

with honors in Computer Science

Reviewed and approved* by the following:

Adam Smith

Associate Professor of Computer Science and Engineering

Thesis Supervisor

John Hannan

Associate Professor of Computer Science and Engineering

Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

We investigate techniques for analyzing the structure of social network datasets while preserving the privacy of individuals' information. The techniques we consider satisfy node-differential privacy, which is a notion quantifying ”privacy” for graph-like datasets. More specifically, we implement and evaluate methods proposed in various literature for selecting an optimal threshold parameter for these algorithms to maintain privacy while still providing useful information about the graph.

We concentrate on two statistics for the purposes of this paper: the number of edges in a graph and the number of triangles. Using algorithms that compute the values of a statistic for varying truncation parameters, we analyze their results to determine a near-optimal threshold value of the statistic to release. We evaluate two methods for this threshold selection, one due to Kasiviswanathan et. al. and another due to Raskhodnikova and Smith. We find that, in most cases, the second method performs better than first.

Table of Contents

List of Figures

List of Tables

Acknowledgements

To my parents, for bringing me to to where I am today.

And to Dr. Adam Smith, for taking me under his wing
and helping me make this paper what it is.

Chapter 1

Introduction

1.1 The Problem

In an age where numerous petabytes of information are being exchanged over of the internet, many people are looking for methods to maintain their own privacy. There is a plethora of information regarding an individual's social presence, search histories, and other sources, and one can see why maintaining their privacy would be an increasing concern. On the other hand, this information can be extremely useful for researchers and companies to help better understand the world around us and to learn how individuals interact. One way of representing this information is by using a graph structure, or a system of nodes and edges. Here, the nodes could represent individual people and the edges could represent some sort of relationship between them. A number of these sources have been made available to the general public, stripped of any personal identifying information, but simply removing this information may not be enough.

Netflix offered a $1,000,000 prize for a 10% increase in their movie recommendation system. They also provided a training dataset, stripped of any personal information, so that the competitors could train their systems. It was found that by cross-referencing this data with the Internet Music Database (IMDB), some of the users' information could be reconstructed [1]. Another method of reconstructing information called a "composition attack" [2], works by obtaining two different statistics of the same dataset to determine a graph's structure or other properties.

In the context of social networks, Jernigan and Mistree [3] also found that your Facebook friends can reveal a fair bit of information about you. They state that public information about one's coworkers, friends, family and other associates can implicitly reveal private information. Using this public information, they were able to determine the sexual orientation of an individual by looking at declared orientation of that individual's neighbors in a Facebook graph. This shows that data about people from social networks can be extremely sensitive. [4, 5]

Since there may be many more ways of reconstructing supposedly anonymous datasets, a solution must be developed that provides a way to accurately obtain various statistics from datasets while provably maintaining a notion of privacy for the people involved. Simply stripping a dataset of any identifying information is not enough. The good news is there have been advancements in this field. In this thesis, we look at the application of one such advance, differential privacy.

1.2 Differential Privacy

Given a dataset, we'd like to learn about some of the features, or statistics. For example, in a social network graph, one might want to know the total number of relationships between individuals (the number of edges) or the number of individuals with a certain number of friends (k-stars). The values for these statistics are mandated by a so-called "curator" of the dataset. A curator is essentially an algorithm which approximates the value of the statistic in order to preserve privacy. This is usually done by adding some amount of noise. The curator must take care not to add too much noise to this value which would destroy the usefulness of the data to researchers.

To begin, we must define what we mean by privacy. *Differential privacy* [6], which emerged from a line of work started by Dinur and Nissim [7], is one definition. This definition essentially states that the data of a single individual in a dataset should not substantially affect the distribution of the information we release. In other words, we should not be able to significantly differentiate between a dataset when we include or exclude a single individual from it.

Various methods have been created to release the values of statistics under differential privacy. The global sensitivity framework [6] depends only on the statistic being released, independent of the input. An instance-based noise mechanism was introduced by Nissim, Raskhodnikova, and Smith [8]. While this technique adds much less noise than the global sensitivity framework, the time complexity is greater making it suitable for only a subset of datasets.

There are various kinds of datasets that exist. We will be focusing on data that deals with relationships portrayed as a graph-like structure. There are a number of different statistics one could request from a graph-like dataset such as the number of edges/relations, the number of triangles (where three vertices of an undirected graph form a triangle of edges), or the number of k-stars (nodes with k outgoing edges). We want to be able to release this statistic while preventing too much leakage of information regarding the individuals in the dataset. Such is the reasoning behind differential privacy. As stated before, the goal of differential privacy is to maximize the accuracy of these statistics while minimizing the probability of leaking information of the individuals or structures about the dataset. Hay et. al. [9] and Karwa et. al. [10] introduce what they call $edge$ and $node$-differential privacy. For the purposes of this paper, we will be utilizing node-differential privacy. Intuitively, node-differential privacy maintains that the removal or addition of a node (and any edges adjacent to it) will not drastically change the value of a statistic substantially. Similarly, one can infer that edge-differential privacy relates to the removal or addition of an edge.

1.3 Our goal

There exist state of the art algorithms for private analysis of graphs that take as input a "threshold" parameter Δ. Roughly Δ is used to truncate a graph based to the distribution of the statistic we would like to calculate. The problem is that selecting the best parameter may in itself reveal information. The goal of this thesis is to understand, implement, and evaluate two proposed methods that will aid us in selecting an optimal parameter while preserving privacy. We compare the methods by performing them several benchmark datasets to assess their performance and determine under which circumstances one method performs better than the other.

1.4 Outline

In section 2, we begin this thesis by giving a technical background on the notion of differential privacy, and definitions to go with it. In section 3, we then explain two reductions for obtaining two different statistics of graph: namely the number of edges and the number of triangles in a graph. Furthermore, we then describe the two methods which will help us determine the optimal threshold parameter for the graphs. In section 4, we first perform the reductions to calculate the values of the statistic for varying thresholds. We then use the proposed methods to choose a near-optimal threshold parameter for various datasets, and compare the results.

Chapter 2

Technical Background

This section has been adapted from [11], section 2 to provide a concise overview of the prerequisites for this paper.

2.1 Graph Metrics and Differential Privacy

We will use the following notation throughout the paper. Let $\mathbb{G}$ be the set of all graphs, let $\mathbb{G}_n$ denote be the set of all graphs with *n* vertices, and let $\mathbb{G}_{n,D}$ be the set of all graphs with *n* vertices with a maximum node degree of *D*.

Given two graphs *G* and *G'*, we define the distance *dist*(*G*, *G'*) between them as the number of nodes that must modified to convert graph *G* to *G'*. A modification of a node is defined as the addition or removal of a node, and any number of edges adjacent to it. We consider *G* and *G'* to be neighbors if *dist*(*G*, *G'*) = 1.

Definition 2.1: (ϵ, δ)-differential privacy[6, 12, 13] states that for $\epsilon, \delta \in \mathbb{R}^+$, any graph $G \in \mathbb{G}_n$, a randomized algorithm $\mathcal{A}$ which calculates some statistic about the graph, and for all values $S \in Range(\mathcal{A})$, the following holds true for all neighboring graphs *G'*:

$$Probability[\mathcal{A}(G) \in S] \leq e^{\epsilon} \times Probability[\mathcal{A}(G') \in S] + \delta$$

If $\delta = 0$, then we refer to this special case as ϵ-differential privacy.

The purpose of an algorithm guaranteeing node-differential privacy is that the algorithm will return roughly the same value of a certain statistic with the removal or addition of a node. The amount the values differ are dependent on ϵ. For smaller values of ϵ, the algorithm will leak less information than those with larger values of ϵ.

Additionally we can see that differential privacy ”composes” well.

Lemma 2.1: (Composition and Post-Processing [14, 15]) If an algorithm $\mathcal{A}$ runs t randomized algorithms $\mathcal{A}_1$,...,$\mathcal{A}_t$ each of which is (ϵ, δ)-differentially private and applies some (not necessarily node-differentially private) randomized algorithm g to their results, i.e., $\mathcal{A}(G) = g(\mathcal{A}_1(G),...,\mathcal{A}_t(G))$ then this resultant algorithm is $(t\epsilon, t\delta)$-differentially private.

2.2 Calibrating Noise to Sensitivity

Now that we know what constitutes a $(t\epsilon, t\delta)$-differentially private algorithm, we now determine how to create such an algorithm. We want to add enough noise to the true value of a statistic to achieve differential privacy.

2.2.1 Noise Distribution

For our purposes, we use two distributions for adding controlled noise:

The Laplace distribution:

$$f(x; b, \mu) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$$

And the Exponential Distribution:

$$f(x; \lambda) = \lambda e^{-\lambda x}$$

2.2.2 Sensitivity

As stated before, differential privacy tries to ensure that the addition or removal of a node does not drastically affect the value of particular statistic. We define the global sensitivity of the function as the follows:

Definition 2.2: (Global Sensitivity [6]) The L^1-global node sensitivity (which uses the L^1 norm) of a function $f : \mathbb{G}_n \rightarrow \mathbb{R}^n$ is defined as:

$$\Delta f = \max_{G,G',dist(G,G')=1} \|f(G) - f(G')\|_1$$

Global Sensitivity is a way of describing the maximum that a statistic can change between a particular graph and all of its neighbors. We can see that the function which returns the number of nodes in a graph has a node sensitivity of 1 for all graphs (since node sensitivity is defined for neighboring graphs). Because we want a statistic's value to not change substantially with the addition or removal of a node, it makes sense to add some noise proportional to the global sensitivity. We use the Laplace mechanism to define how to add this nose:

Theorem 2.1: (Laplace Mechanism [6]) Given a graph statistic function f, an algorithm $\mathcal{A}(G) = f(G) + \text{Lap}(\Delta f/\epsilon)^p$, which takes a graph G as input, is ϵ-node-private.

So if we release the number of nodes in a graph, $|V|$, with noise of expected magnitude $\frac{1}{\epsilon}$, then that is node-differentially private.

2.2.3 Bounded Degree Graphs

A graph is D-bounded if it has a maximum degree of at most D. Kasiviswanathan et. al. [11] define a variant a differential privacy and sensitivity for these degree-bounded graphs:

Definition 2.4: (Degree-bounded $(\epsilon, \delta)_D$-node differentially privacy [11]) A randomized algorithm $\mathcal{A}$ is $(\epsilon, \delta)_D$-node differentially private if for all pairs of D-bounded graphs $G_1, G_2 \in \mathbb{G}_{n,D}$ such that G_1, G_2 are neighbors since we have:

$$\textit{Probability}[\mathcal{A}(G_1) \in S] \leq e^{\epsilon} \times \textit{Probability}[\mathcal{A}(G_2) \in S] + \delta$$

Thus we can extend the global sensitivity framework to work with degree-bounded graphs. Since we can now assume that the nodes of a graph have at most degree D, then many statistics' global sensitivity thus decrease. For example, the global sensitivity for the function that returns the number of edges in a graph is n, where n is the number of nodes in a graph. This is because an addition or removal of a node can add or remove at most n edges from the graph (from that node to every other node in the graph). When the degree is D-bounded and when $D < n$, then the global sensitivity becomes D since the addition or removal of a node can now only add or remove at most D edges.

Definition 2.5: (Global Sensitivity on degree-bounded graphs [11]) The L^1-global node sensitivity (which uses the L^1 norm) of a graph statistic function f is:

$$\Delta_D f = \max_{G,G' \in \mathbb{G}_{n,D} : d_{node}(G,G')=1} ||f(G) - f(G')||_1$$

Observation 2.1: (Laplace Mechanism on degree-bounded graphs [11]) The algorithm $\mathcal{A}G = f(G)$+Lap$(\Delta_D f/\epsilon)^p$ is ϵ_D-node differentially private.

As stated before, utilizing degree-bounded graphs, we can lower the global sensitivity. There are many techniques that make use design algorithms that are: (a) differentially private on all inputs, and (b) accurate on inputs that are $closetoD$-bounded. The question then becomes: how should we elect the value of D that these algorithms should use? This thesis considered two such methods, described in the next section, and evaluates them on a variety of datasets.

Chapter 3

Algorithm Implementations and Mechanisms

In this section, we describe two differentially private algorithms which take in a graph G, and values for ϵ and δ and compute a value for a chosen statistic. The algorithms require, as part of their input, a parameter D corresponding roughly to the maximum degree in the graph. We then introduce two methods for selecting a near-optimal value for the parameter D so as to ensure reasonably accurate values of the statistic, while ensuring differential privacy.

3.1 Projection Operators

Kasiviswanathan, Nissim, Raskhodnikova, and Smith [11] present various techniques to ”project” the value of a specified statistic to a bounded-degree graph. The chosen statistics have low sensitivity and thus can preserve information regarding the official value of the statistic. Our objective is to find an optimal threshold parameter, *D*, that will give us the most information about the statistic while losing the least amount of information when applying noise.

3.1.1 Reductions

Flow-based Algorithm for Releasing Edges

The first algorithm we utilize is from Kasiviswanathan et. al. [11]. This algorithm uses maximum flow to release the number of edges in a graph.

Definition 3.1: Given an (undirected) graph *G* = (*V*, *E*), we let $V_L = \{v_L | v \in V\}$ and $V_R = \{v_R | v \in V\}$ be two copies of the vertices *V* in *G*, called the left and right copies respectively. Let *D* be a natural number less than *n* = $|V|$. The flow graph of *G* with parameter *D*, a source *s* and a sink *t* is a directed graph with nodes $V_L \cup V_R \cup \{s, t\}$. The edges are added as follows: we add an edge of capacity, or weight, *D* from the source *s* to every node in V_L. Similarly, we add an edge of capacity *D* from every node from V_R to the sink *t*. We then add edges (u_L, v_R) of capacity 1 for all edges $\{u_L, vR\} \in E$. We then let v_{flow}(*G*) denote the value of the maximum flow in the flow graph of *G*. If the maximum degree in the graph is $< D$, then v_{flow}(*G*) = number of edges in G.

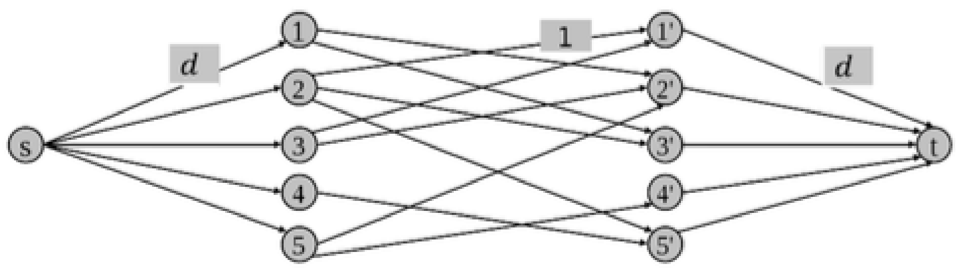

Figure 3.1: Visualization of a sample Flow Graph where *G* contains 5 nodes

Lemma 3.1: Given an undirected graph *G* = (V, E) and a degree bound *D*, the algorithm which returns v_{flow}(*G*) + Lap($\frac{2D}{\epsilon}$) as an approximate count for the number of edges in the graph, is ϵ-node differentially private. Moreover, when the max degree is at most D, v_{flow}(*G*) = $2 * |E|$.

LP-based Algorithm for Releasing Number of Triangles

The second algorithm we utilize is a linear programming based algorithm for releasing the counts of subgraphs in a graph *G*. In this thesis we will concentrate on releasing the count of the number of triples, also known as a 3-clique, in a graph. Other statistics such as *k*-stars are also possible with this algorithm.

Definition 3.1: Given an undirected graph *G* = (*V*, *E*) and a bound Δ such that Δ is a natural number less than *n* = $\binom{|V|}{2}$, we can construct a linear program to calculate the number of triangles

T of *G*. For every copy of a triangle *T* found in *G*, we introduce a variable x_C in the linear program. We let $\Delta(f)$ denote the global node sensitivity of a function f in Δ-bounded graphs. Then the LP corresponding to *G* is specified as follows:

maximize $\sum_{trianglesTinG} x_C$ subject to:
$0 \leq x_C \leq 1$ for all variables x_C and
$S_v \leq \Delta$ for all nodes $v \in V$, where $S_v = \sum_{C:v \in V(C)} x_C$

We let the maximal value of the linear program be v_{LP}(*G*). If Δ is chosen such that every vertex of the graph participates in Δ or fewer triangles, then v_{LP}(*G*) = number of triangles in G.

Lemma 3.2: Given an undirected graph *G* = (V, E) and a bound Δ, the algorithm which returns v_{LP}(*G*) + Lap($\frac{\Delta}{\epsilon}$) as an approximate count for the number of triangles in the graph, is ϵ-node differentially private. Moreover, when the bound is at most D, v_{LP}(*G*) = *num triangles in G*.

3.2 Mechanisms for Determining Optimal Truncation Paramaters

Now that we have described how to compute the value of a statistic for degree-bounded graph, we try to determine an optimal cutoff threshold $\hat{\Delta}$. We define two methods that will try to compute an "optimal" value for Δ. The existing method is one described in Kasiviswanathan et. al. [11]. The second method was recently proposed by Raskhodnikova et. al. [16]. For the statistic that counts edges, we set Δ = D. For the statistic that counts triangles, we set Δ = $\binom{D}{2}$. In both cases, 0 $\leq D \leq |V|$. Suppose we have, for real values of Δ_i, a function that f_{Δ_i} that approximates f whose global sensitivity is at most Δ_i. Then for all values of Δ_i and a selected value of ϵ, we compute the errors as the following:

$$error(\Delta_i) = |f(G) - f_{\Delta_i}(G)| + \Delta_i/\epsilon.$$

We denote the value of Δ_i that minimizes this error as Δ^*. Δ^* is dependent on the chosen value of ϵ. The error function gives us an idea of the errors of the algorithms described in Lemma 3.1 and

3.2. The error function is an approximation to the error of each of those two algorithms, depending on which statistic f calculates. Our goal is to find a differentially-private algorithm that finds some Δ whose error is close to the error of Δ^*.

The methods described above take the graph G, and values for ϵ, β, and $\Delta_1, \Delta_2, ..., \Delta_k$. β is an upper bound on the probability that the algorithm makes a certain type of mistake. Lower values of β favor lower values of Δ. The Δ_i's are chosen to be powers of two to cover the range of possibilities for Δ.The methods return a value, which we denote as $\hat{\Delta}$, which is an optimal truncation parameter for each of the methods.

3.2.1 Method 1

We introduce the first mechanism to select a suitable projection degree. The method is described by Kasiviswanathan et. al. [11] and works as follows:

1. Set $\epsilon' = \frac{\epsilon}{k}$ where $k = \ln(|V|)$

2. Compute the values of $\hat{x}_i$ with $\Delta_1, \Delta_2, ..., \Delta_k$ such that:
   $\hat{x}_i = \hat{f}_{\Delta_i}(G) + \text{Lap}(\frac{\Delta_i}{\epsilon'})$
   Remember we stated in section 2 that we use the Laplace Distribution to add controlled noise. Furthermore, the Laplace noise used for different values of i is independent.

3. Compute the value of i that maximizes $\hat{x}_i - \frac{\Delta_i ln(\frac{k}{\beta})}{\epsilon'}$, where $0 \leq \beta \leq 1$
   We denote the value that maximizes the above value as $\hat{\Delta}$

The value that we compute above is an approximation of the optimal threshold value for Δ.

3.2.2 Method 2

The Exponential Mechanism of McSherry and Talwar [17] is one technique to help design differentially private algorithms. We try and determine the optimal threshold value using the method describe in [16] which works as follows:

1. Set t = log(k/β) where $0 \leq \beta \leq 1$

2. For all values of $\Delta_1, \Delta_2, ..., \Delta_k$, compute the errors as described earlier.

3. Next compute the scores where:
   $score(i) = \max_{i,j where i \neq j} \frac{(error(i)+t\frac{\Delta_i}{\epsilon})-(error(j)+t\frac{\Delta_j}{\epsilon})}{\Delta_i+\Delta_j}$

4. Compute the value of i that minimizes $score(i) - \frac{2}{\epsilon}E_i$ where E_i is an exponential random variable such that $E_i \sim Exp(1)$. Furthermore, the E_i's are independent.
   We denote the value that minimizes the above value as $\hat{\Delta}$

The value that we compute above is another approximation of the optimal threshold value for Δ.

3.3 Implementation

We utilize the resources implemented by Lu [18] to perform the reductions that provide us with the number of edges or triangles in various graph-like datasets. We use the Python programming language to use these values and perform the two methods proposed above. Finally, we used Microsoft Excel to plot the curves shown in section 4.

Chapter 4

Results

This section explains the results of our experiments. We first perform the reductions for computing the value of a given statistic on multiple graph-like datasets. The results of the reduction are shown later in this section. Using these results, we then apply the methods introduced in the previous section to determine how well those methods perform by comparing their respective errors.

4.1 Overview of our Results

We find that, in general, the second method almost always performs better than the first method. There are some cases where the relative errors are comparable as ϵ increases, but in general, the second method almost always has smaller relative errors for all values of ϵ. We must note that, in some cases, the relative error of even the second method might be too high for practical purposes. A relative error greater than even 0.1 is a cause for concern.

4.2 Graph Information

We make use of the graphs publicly available in the Stanford Large Network Dataset Collection [19]. We use the graphs named in the following table which contains relevant information regarding the statistics we'd like to extract.

Graph Name	**Vertices**	**Edges**	**Triangles**
CA-AstroPh.txt	18772	396220	1351441
CA-HepTh.txt	9877	-	28339
com-youtube.ungraph.txt	1134890	5975248	3056386
com-dblp.ungraph.txt	317080	2099732	-

Table 4.1: Graph Information

The number of edges in CA-HepTH.txt and the number of triangles in com-dblp.ungraph.txt are omitted because we don't perform those respective reductions for the purposes of this paper. CA-HepTH.txt is a fairly small graph so the number of triangles could be computed for even small values of Δ. Additionally, we split our experiments such that three of them would relate to edges while the other three would relate to triangles.

4.3 Performing the Reductions

The table below shows us the values obtained from the Flow-based projection when bounding the respective graphs with the specified degree. We can see that the value for the edge count increases as the degree bound increases up to a point after which it remains constant. This point occurs when $f_\Delta(G) = f(G)$. For convenience, this point has been emphasized in the table below.

D	**Edge count for CA-AstroPh.txt**	**Edge count for com-dblp.ungraph.txt**	**Edge count for com-youtube.ungraph.txt**
1	18438	283808	18438
2	35549	512579	35549
4	64785	835354	64785
8	110433	1209778	110433
16	175904	1573470	175904
32	256078	1862810	256078
64	329292	2028242	329292
128	375739	2089950	375739
256	392430	2099374	392430
512	**396220**	**2099732**	**396220**
1024	396220	2099732	396220
2048	396220	2099732	396220
4096	396220	2099732	396220

Table 4.2: Results from the Flow-Based Reduction for Edges

Similarly, the table below shows us the values obtained from the LP-based projection when bounding the respective graphs with the specified degree. We can see that the value for the triangle count also increases as the degree bound increases up to a point after which it remains constant. This point occurs when $f_{\Delta}(G) = f(G)$ For convenience, this point has been emphasized in the table below.

D	Triangle count for CA-AstroPh.txt	Triangle count for CA-HepTh.txt	Triangle count for com-youtube.ungraph.txt
1	-	0	-
2	-	2032	-
4	-	7975	-
8	-	16392	-
16	-	22740	-
32	711088	28330	992086
64	1186902	**28339**	1644475
128	1345951	28339	2383100
256	**1351441**	28339	2773615
512	1351441	28339	**3056386**
1024	1351441	28339	3056386
2048	1351441	28339	3056386
4096	1351441	28339	3056386

Table 4.3: Results from the LP-Based Reduction for Triangles

Because the linear program had started to take a longer time for Δ bounds < 32 for both CA-AstroPh.txt and com-youtube.ungraph.txt, those values have been omitted.

4.4 Analysis of the Mechanisms

The values computed above contain no noise. We implement the mechanisms described previously to select a good value for Δ. Because the amount of noise is random following some distribution, we run the mechanisms for 10,000 iterations to gain a better understanding of the performance of these mechanisms. The values received are then averaged out for the same reason. We test these mechanisms for varying values of ϵ and β to see how the values chosen affect the optimal chosen threshold value. Namely, 0.01 $\leq \epsilon \leq$ 0.1 in 0.01 increments and $\beta \in [0.01, 0.05, 0.03]$. The error bars shown in the figures below denote the 10th and 90th percentiles.

4.4.1 Error Comparisons

We compare the relative errors obtained for each of the methods to the errors of Δ^* for varying values of ϵ. We chose relatively spaced out values of β to show how β affects the amount of error. We also normalize the errors using the function below:

$$error_{relative}(i) = \frac{error(\hat{\Delta})}{f(G)}$$

In all cases, the results given by Method 1 have more error when compared to the results given by Method 2 for lower values of ϵ. This error decreases as ϵ increases, but in general doesn't reach the same amount of error for the results given by Method 2. On the other hand, the Method 2's values tend to approach the error of Δ^* as ϵ increases. This shows that Method 2 is almost always a better method to use when trying to determine an optimal projection degree, at least for the graphs tested on. Additionally, as β increases, we can see that the errors decrease. This seems to actually make a fairly substantial difference for Method 1, but for Method 2 the decrease in the amount of error isn't nearly as significant.

The figures that follow compare the relative errors of the two methods. We organize the figures with the statistic counting edges first, followed by the statistic counting triangles. We use three values of β to depict how its value affects the relative errors for each the methods.

Edges using Flow-based Reduction in CA-AstroPh.txt

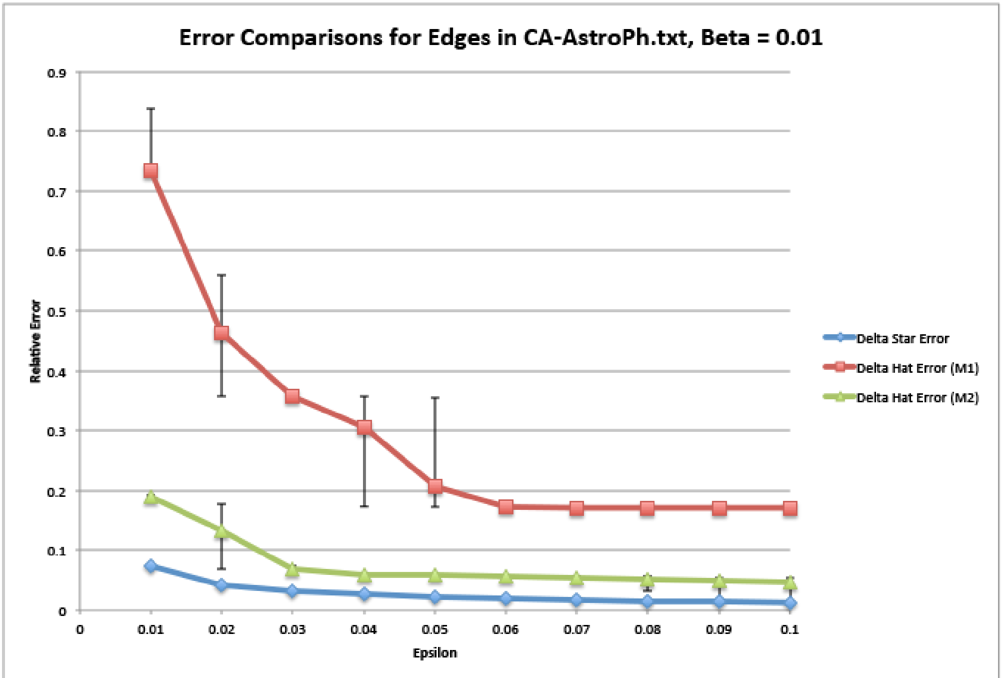

Figure 4.1: Error Comparisons for Edges in CA-AstroPh.txt with $\beta = 0.01$

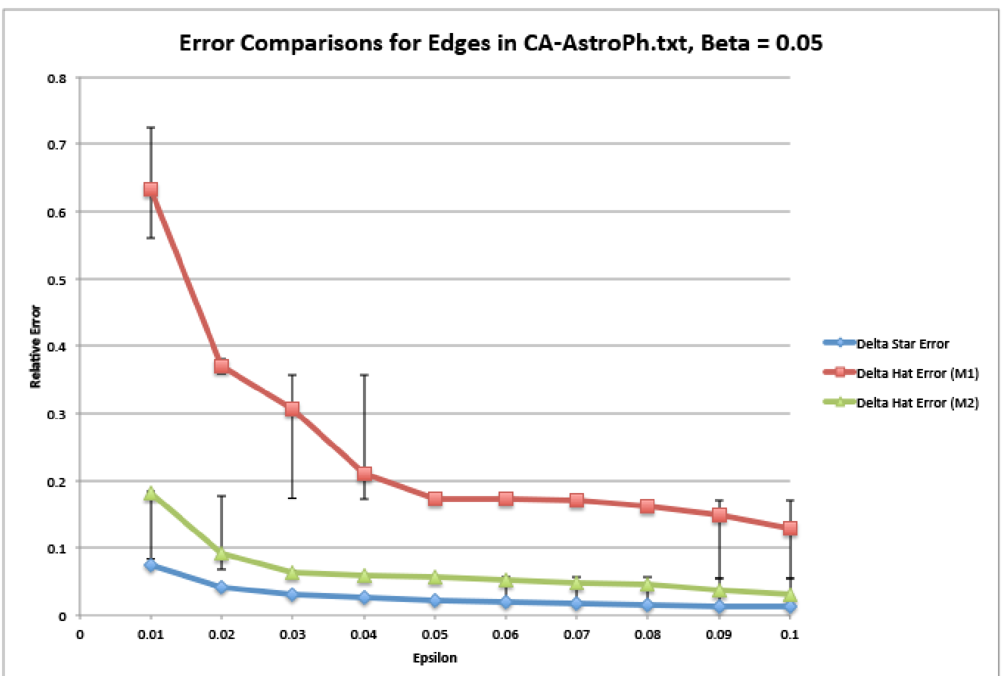

Figure 4.2: Error Comparisons for Edges in CA-AstroPh.txt with $\beta = 0.05$

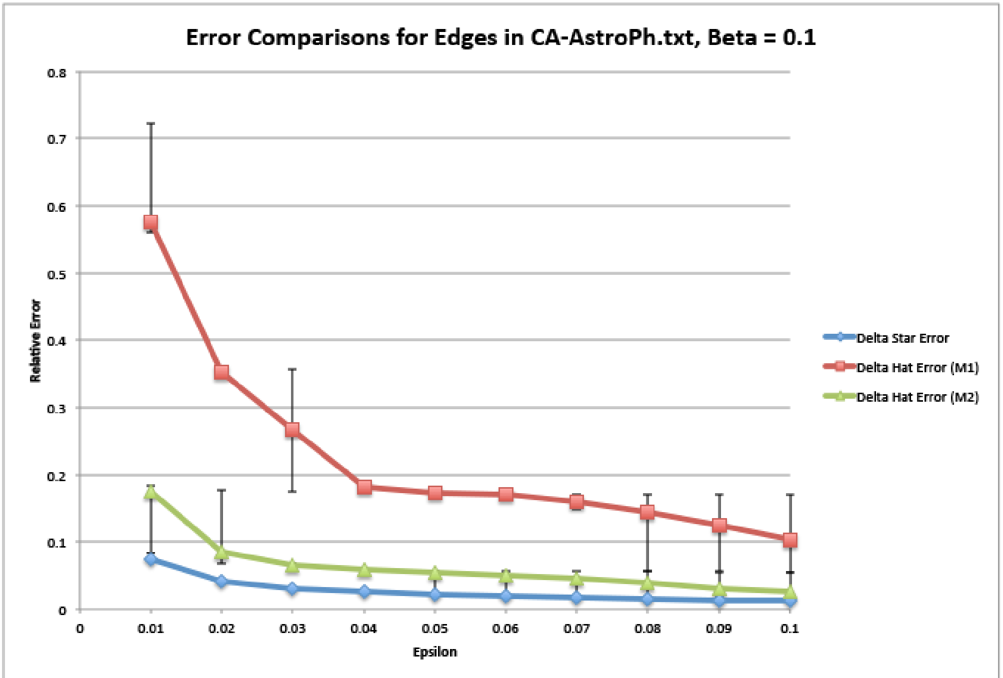

Figure 4.3: Error Comparisons for Edges in CA-AstroPh.txt with $\beta = 0.1$

Edges using Flow-based Reduction in com-dblp.ungraph.txt

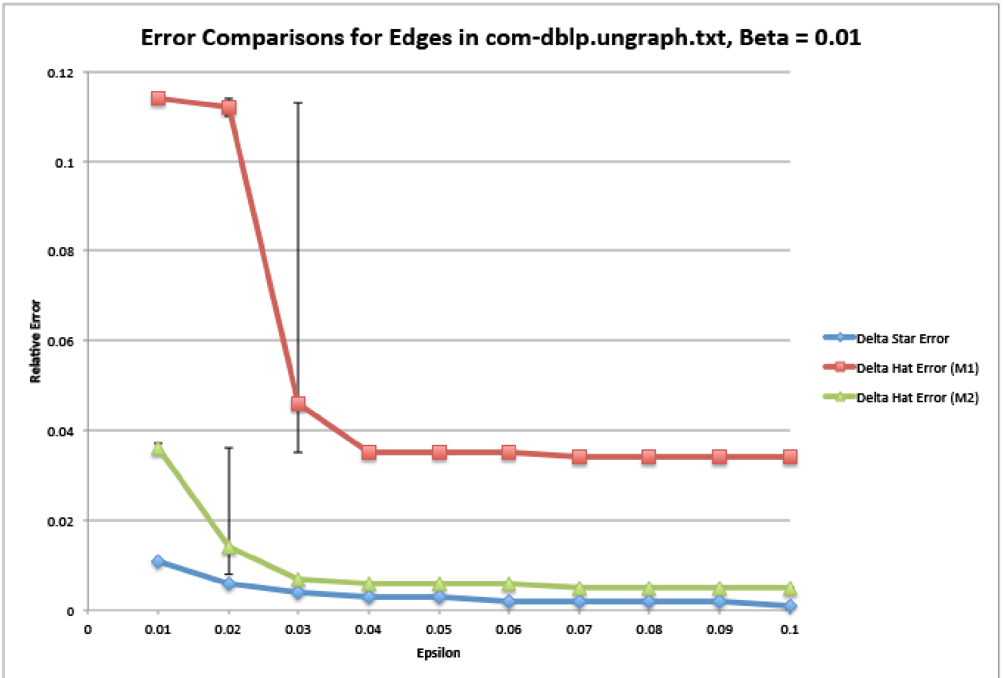

Figure 4.4: Error Comparisons for Edges in com-dblp.ungraph.txt with $\beta = 0.01$

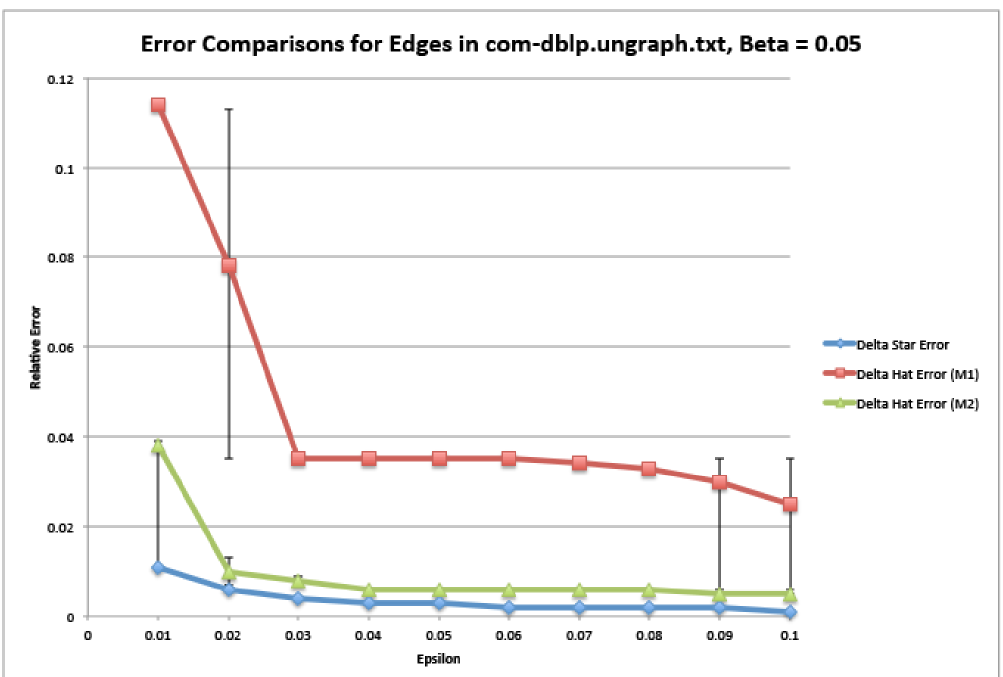

Figure 4.5: Error Comparisons for Edges in com-dblp.ungraph.txt with $\beta = 0.05$

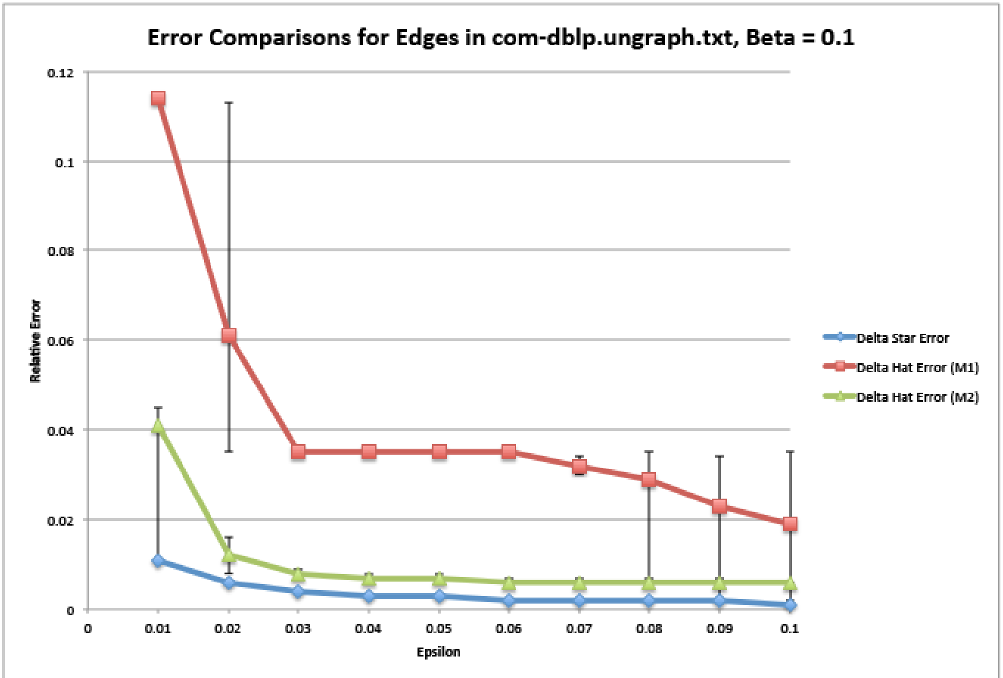

Figure 4.6: Error Comparisons for Edges in com-dblp.ungraph.txt with $\beta = 0.1$

Edges using Flow-based Reduction in com-youtube.ungraph.txt

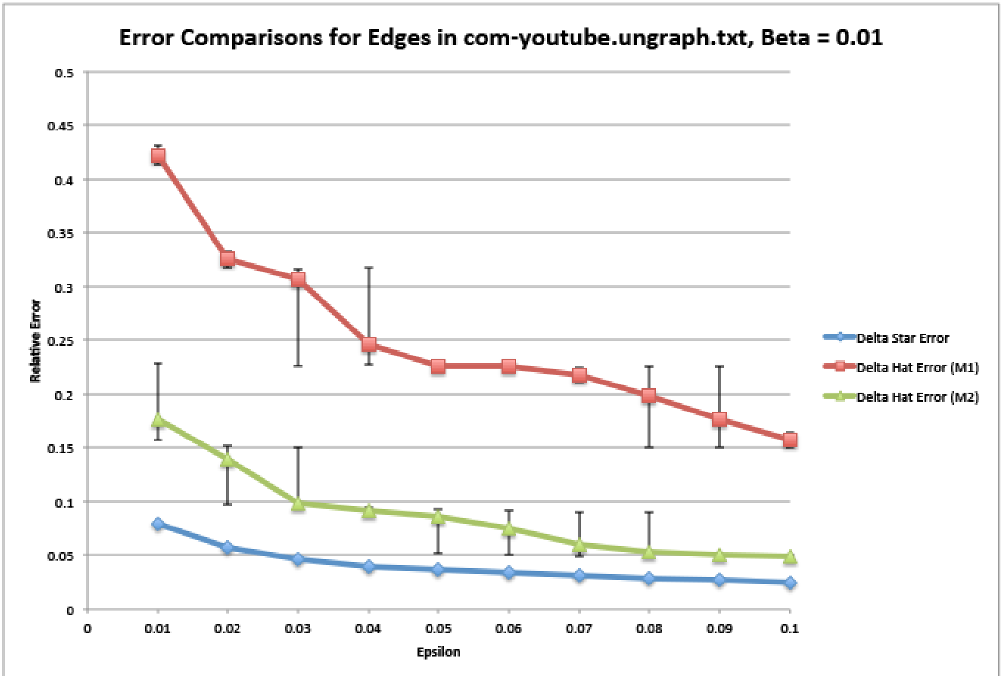

Figure 4.7: Error Comparisons for Edges in com-youtube.ungraph.txt with $\beta = 0.01$

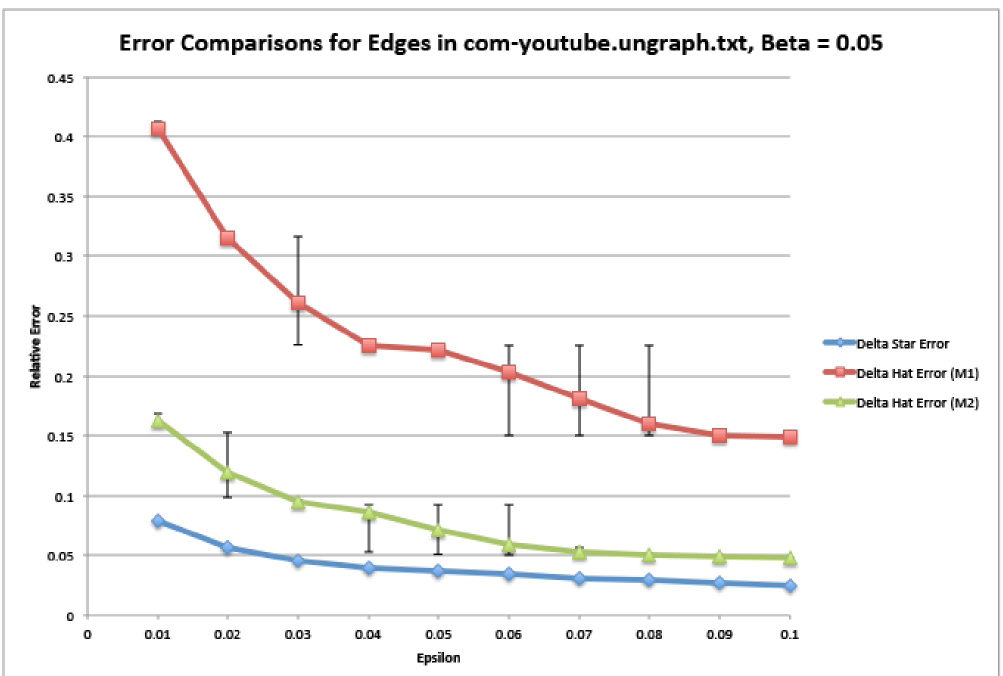

Figure 4.8: Error Comparisons for Edges in com-youtube.ungraph.txt with $\beta = 0.05$

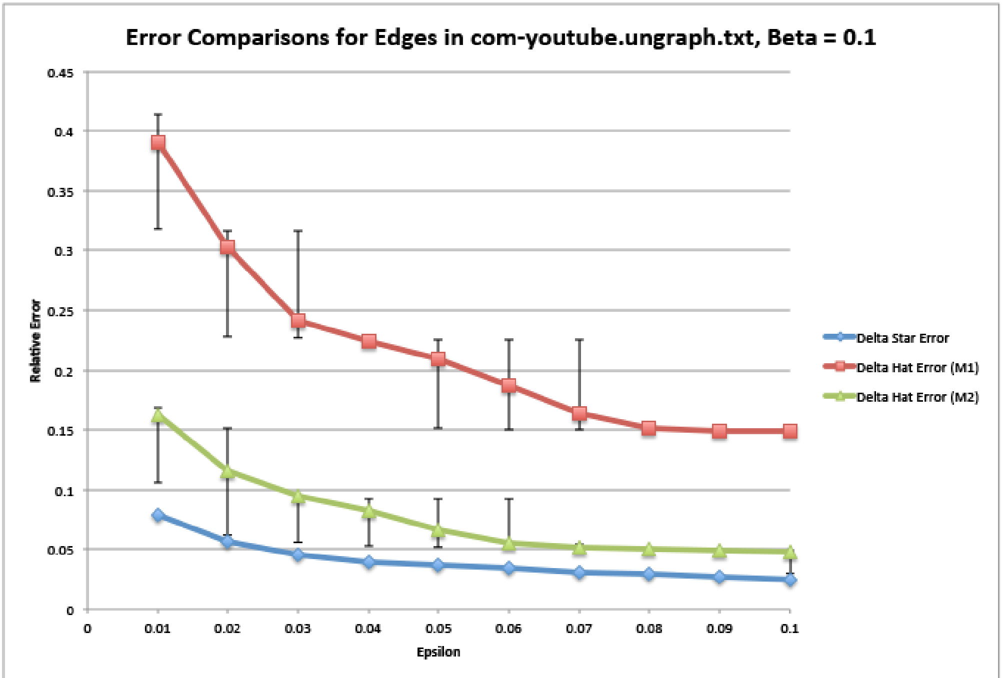

Figure 4.9: Error Comparisons for Edges in com-youtube.ungraph.txt with $\beta = 0.1$

Triangles using LP-based Reduction in CA-AstroPh.txt

Figure 4.10: Error Comparisons for Triangles in CA-AstroPh.txt with $\beta = 0.01$

Figure 4.11: Error Comparisons for Triangles in CA-AstroPh.txt with $\beta = 0.05$

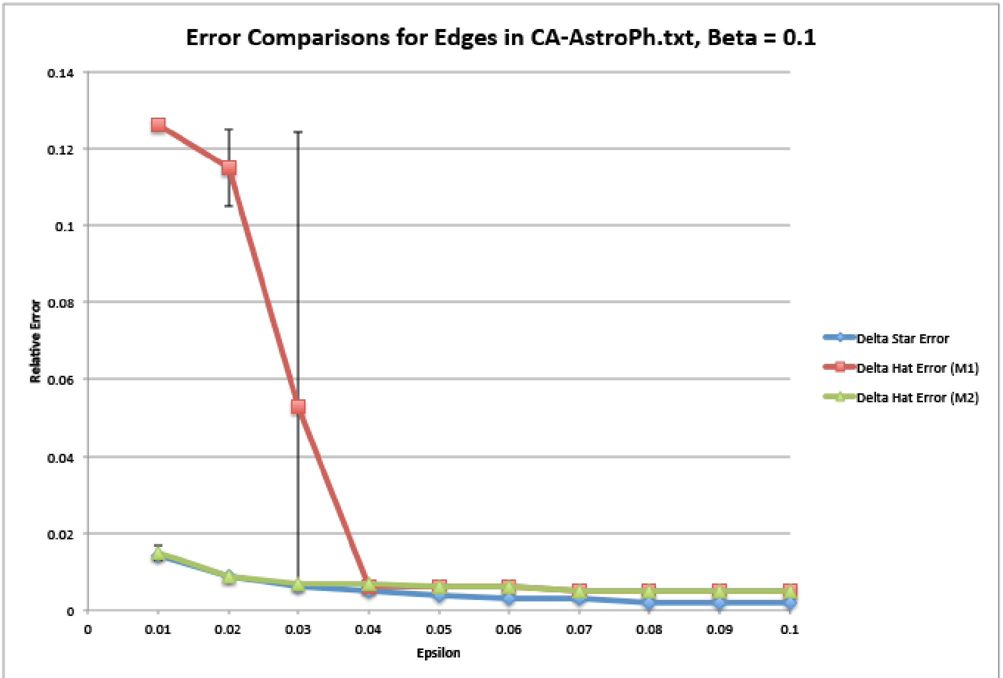

Figure 4.12: Error Comparisons for Triangles in CA-AstroPh.txt with $\beta = 0.1$

Triangles using LP-based Reduction in CA-HepTh.txt

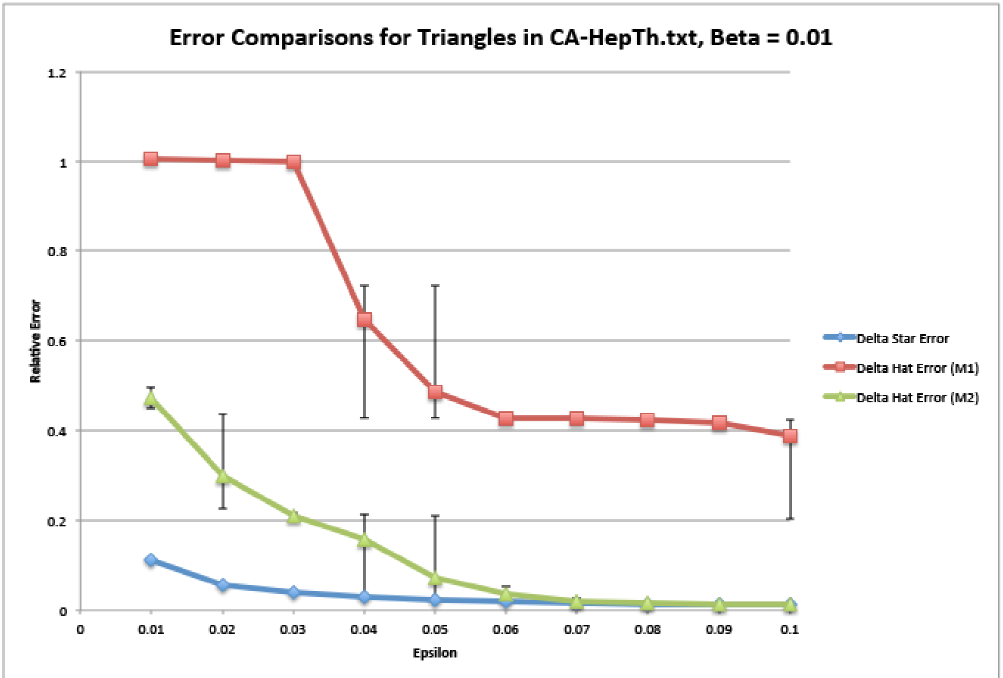

Figure 4.13: Error Comparisons for Triangles in CA-HepTh.txt with $\beta = 0.01$

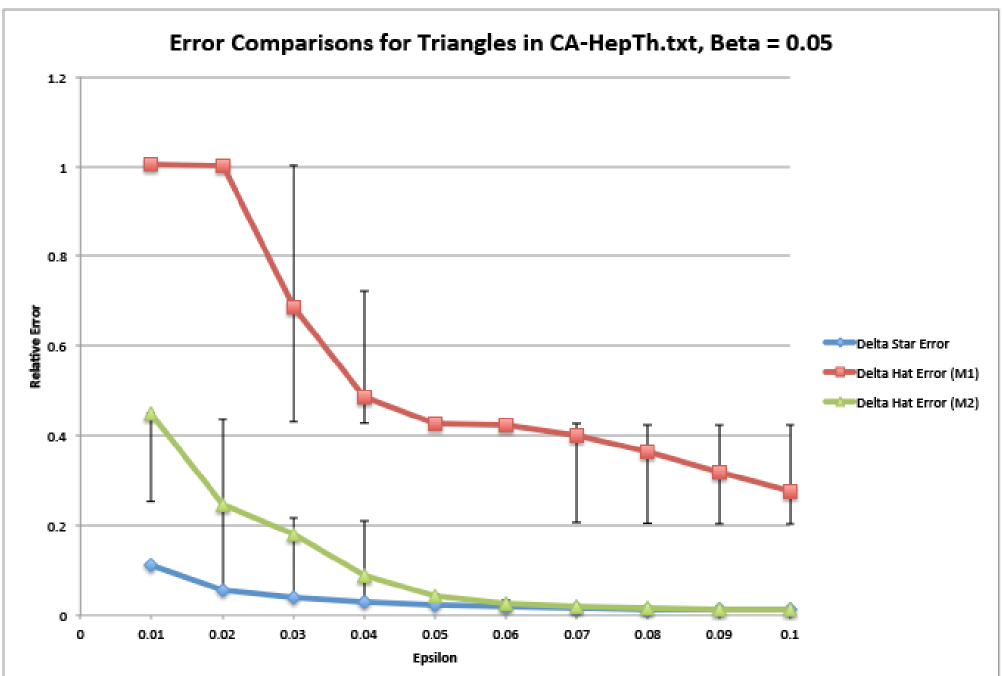

Figure 4.14: Error Comparisons for Triangles in CA-HepTh.txt with $\beta = 0.05$

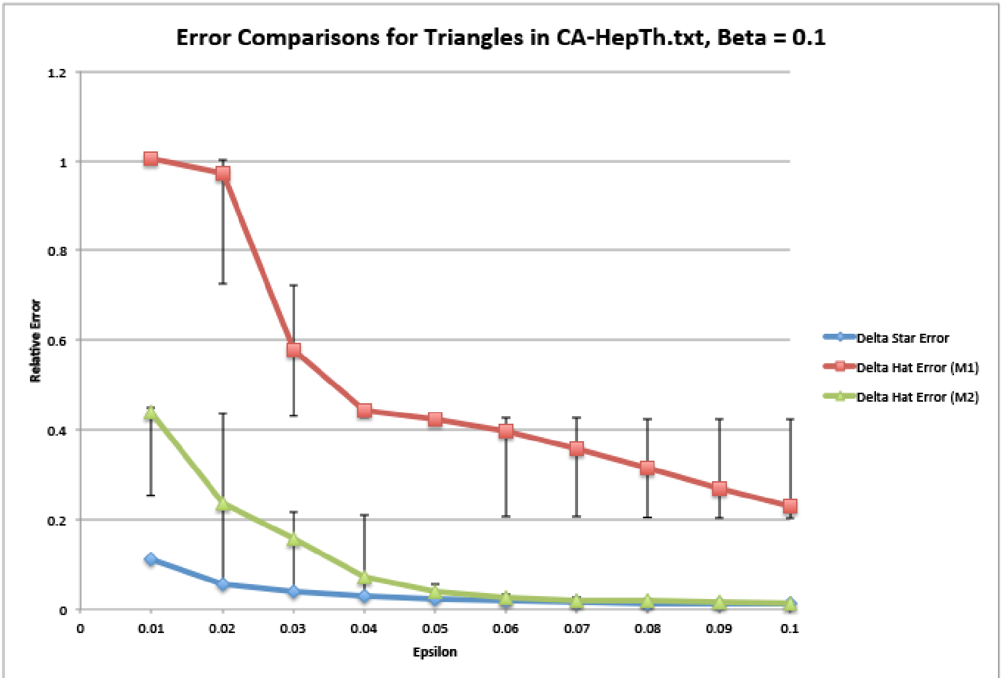

Figure 4.15: Error Comparisons for Triangles in CA-HepTh.txt with $\beta = 0.1$

Triangles using LP-based Reduction in com-youtube.ungraph.txt

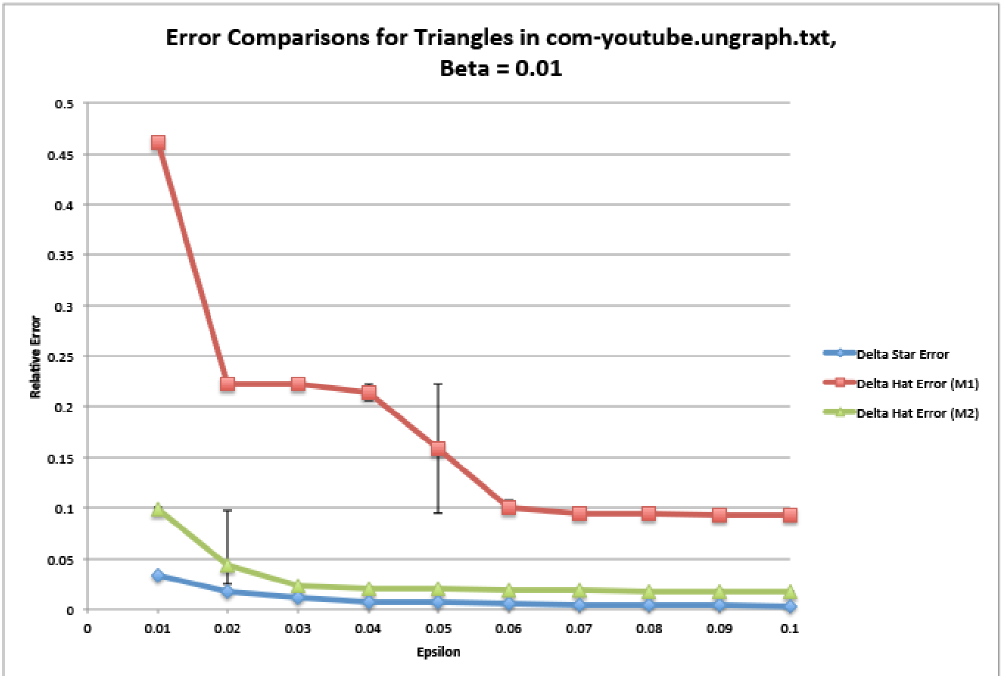

Figure 4.16: Error Comparisons for Triangles in com-youtube.ungraph.txt with $\beta = 0.01$

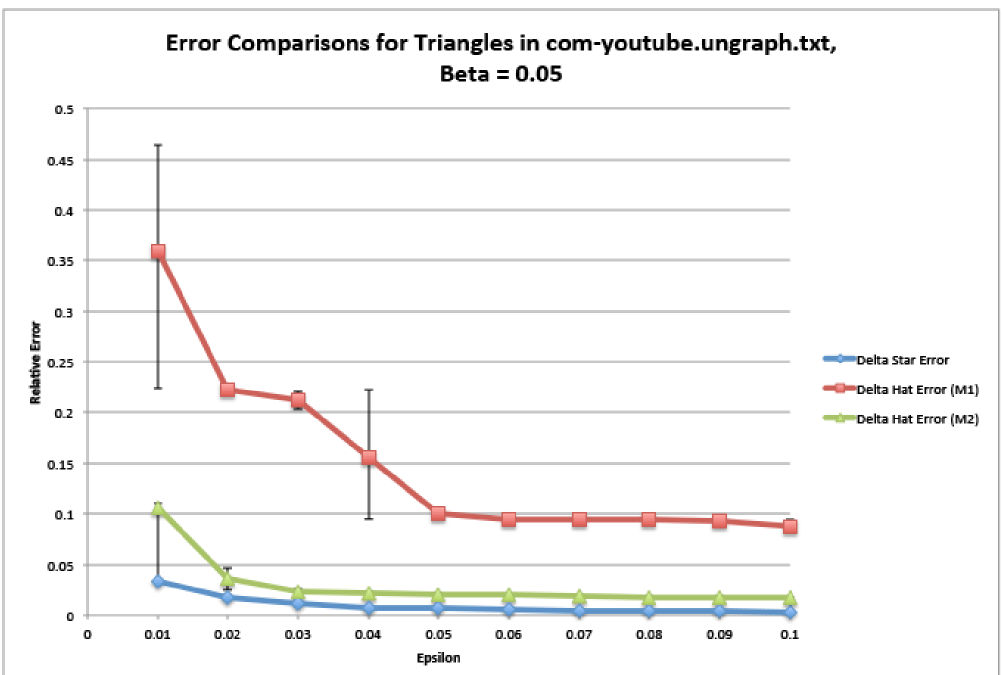

Figure 4.17: Error Comparisons for Triangles in com-youtube.ungraph.txt with $\beta = 0.05$

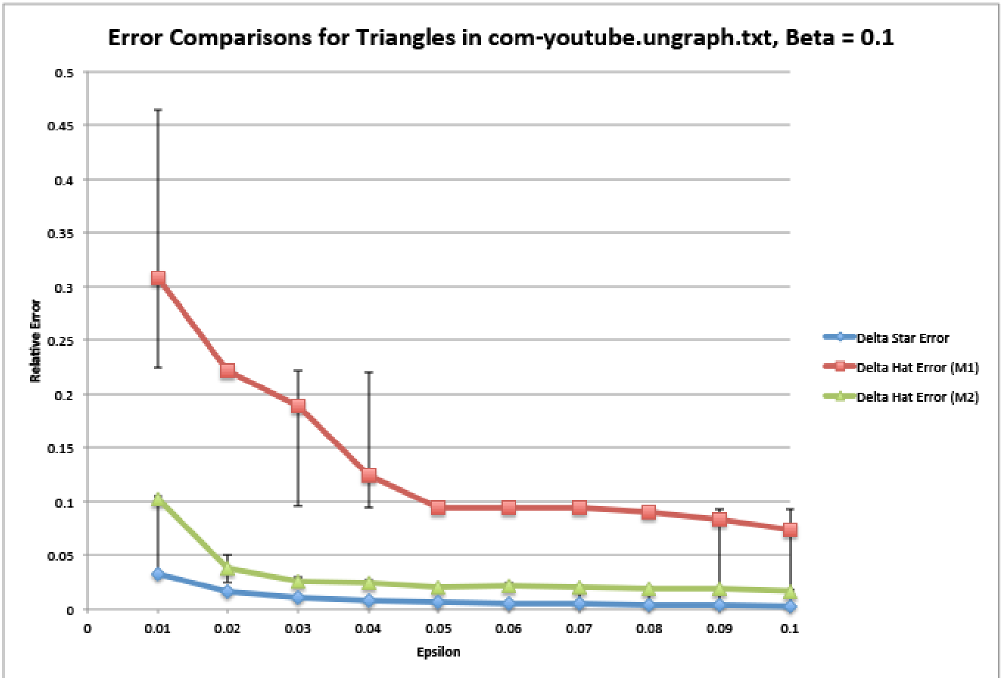

Figure 4.18: Error Comparisons for Triangles in com-youtube.ungraph.txt with $\beta = 0.1$

Chapter 5

Conclusions

5.1 Final Words

In this thesis, we describe two methods to select a near-optimal threshold parameter Δ for the algorithms presented be Kasiviswanathaon et. al. [11]. The algorithms themselves guarantee node differential privacy while at the same time produce less error than previous work. We work with various values of Δ, and try and see how varying values for ϵ and β affect the selection for the optimal cutoff threshold.

In general, it seems that Method 1 seems to be more erroneous than Method 2. Both Method 1's and Method 2's values seem to approach the error of Δ^* as ϵ increases, as expected. But we can see that Method 1 usually levels out at a higher error value than Method 2's and also usually has a much larger error for lower values of ϵ. Additionally, increasing the value of β tends to reduce the error, but the aforementioned trends still persist.

The methods described above are just two mechanisms utilized to select a suitable projection degree. We analyzed a few graphs to see how they respond to both the methods. We can analyze other methods or how other graphs react to these mechanisms. We leave this to future work.

Bibliography

[1] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. *IEEE Symposium on Security and Privacy (2008)*, Feb 2008.

[2] S. P. Kasiviswanathan, R. G. Srivatsava, and A. Smith. Composition attacks and auxiliary information in data privacy. *KDD (2008)*, 2008.

[3] C. Jernigan and B.F.T. Mistree. Gaydar: Facebook friendships expose sexual orientation. *First Monday 14(10) (2009)*, 2009.

[4] A. Narayanan and V. Shmatikov. De-anonymizing social networks. *The University of Texas at Austin (2009)*, 2009.

[5] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. *ACM Digital Library (2007)*, 2007.

[6] C. Dwork, F. McSherry, K Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *TCC (2006)*, 2006.

[7] I. Dinur and K. Nissim. Revealing information while preserving privacy. *PODS (2003)*, 2003.

[8] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. *STOC (2007)*, May 2011.

[9] M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate estimation of the degree distribution of private networks. *ICDM (2009)*, 2009.

[10] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structures. *PVLDB 4,11 (2011)*, 2011.

[11] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. *TCC (2013)*, Jul 2012.

[12] C. Dwork. Differential privacy. *ICALP (2) (2006)*, 2006.

[13] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. *EUROCRYPT (2006)*, 2006.

[14] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. *KDD (2009)*, 2009.

[15] C. Dwork and J. Lei. Differential privacy and robust statistics. *STOC (2009)*, 2009.

[16] S. Raskhodnikova and A. Smith. Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions. *UNPUBLISHED (2015)*, Apr 2015.

[17] F. McSherry and K. Talwar. Mechanism design via differential privacy. *FOCS (2007)*, 2007.

[18] E. Lu. Implementation and evaluation of algorithms for releasing graph statistics under node-differential privacy. *SHC (2013)*, 2013.

[19] J. Leskovec. Stanford large network dataset collection. `https://snap.stanford.edu/data/`. Last Accessed 2015-06-09.

ACADEMIC VITA

SUJEET BHANDARI

EDUCATION

The Pennsylvania State University, University Park, Pennsylvania
Bachelor of Science in *Computer Science* with Honors — **Schreyer's Honors Society**
Bachelor of Science in *Electrical Engineering* — *May 2015*
Minor in *Mathematics*

SKILLS

Programming Languages: C++, C, Python*, Java, Objective-C, Lua*, Assembly (MIPS ISA), Verilog/VHDL, LabVIEW
Software: NI Multisim, NI Ultiboard, ModelSim, SolidWorks, Mac OS X, Linux, Windows OS, MS Office, Xcode, Git
*Independently

PROFESSIONAL EXPERIENCE

Software Engineering Intern, Google Inc. – Summer 2014

- Expanded on an internal tool by implementing various useful features to aid in predicting pCTR
- Implemented these features using various programming languages such as Python, C++, and JavaScript

RESEARCH EXPERIENCE

Differential Privacy – 2014-2015

- Analyzed algorithms to determine optimal truncation degrees for releasing graph statistics under node-differential privacy

PROJECT EXPERIENCE

Circuit Card Review Application – 2014

- Project Leader for CMPSC Design Process (partnered with Lockheed Martin)
- Developed a command line application in C++ which analyzes circuits to detect errors in the wiring

Optical Theremin – *2013*

- Project Leader for EE Design Process
- Analyzed varying light intensities using photodiodes
- Varied frequency and amplitude of sine waves based on this analysis using a customized back-end created in LabVIEW
- Interpreted resultant sine waves to create a musical instrument

Tournament Handler – *2012*

- Implemented a Single-Elimination seeded tournament bracket in Java
- Utilized an SQL database to add players and to store the sequence of events

EVENTS

Microsoft Coding Competition (ACM/ICPC-Style Programming Competition) – *2015*

- Placed 2nd (Spring 2015)

CodePSU (ACM/ICPC-Style Programming Competition) – *2012-2014*

- Placed 1st in the Advanced Tier (Spring 2014)
- Placed 2nd in the Advanced Tier (Spring 2013)

PennApps – *2013*

- Created a program to generate a photo mosaic out of randomly generated YouTube thumbnails

INITIATIVES

Academic Tutor – *2007-Present*

- Helped over 50 students by explaining concepts in detail and improving their grades
- Subjects include Computer Science, Electrical/Computer Engineering, Math, Physics, and Chemistry