Anonymizing Edge-Weighted Social Network Graphs

Sudipto Das Ömer Eğecioğlu Amr El Abbadi

Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106 - 5110, USA
{sudipto, omer, amr}@cs.ucsb.edu

ABSTRACT

The increasing popularity of social networks has initiated a fertile research area in information extraction and data mining. Although such analysis can facilitate better understanding of sociological, behavioral, and other interesting phenomena, there is growing concern about personal privacy being breached, thereby requiring effective anonymization techniques. If we consider the social graph to be a weighted graph, then the problem of anonymization can be of various types: *node identity anonymization*, *structural anonymization*, or *edge weight anonymization*. In this paper, we consider edge weight anonymization. Our approach builds a linear programming (LP) model which preserves properties of the graph that are expressible as linear functions of the edge weights. Such properties form the foundations of many important graph-theoretic algorithms such as *single source shortest paths tree, all-pairs shortest paths, k-nearest neighbors, minimum cost spanning tree*, etc. Off-the-shelf LP solvers can then be used to find solutions to the resulting model where the computed solution forms the weights of the anonymized graph. As a proof of concept, we choose the *shortest paths problem* and its extensions, prove the correctness of the constructed models, analyze their complexity, and experimentally evaluate the proposed techniques using real social network data sets. Our experiments demonstrate that not only does the proposed technique anonymize the weights, but it also improves the *k-anonymity* of the graphs while scrambling the relative ordering of the edge-weights, thereby providing robust and effective anonymization of the sensitive edge-weights.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and networks; G.1.6 [**Optimization**]: Linear programming; J.4 [**Social and Behavioral Sciences**]: Sociology

General Terms

Algorithms, Design, Security.

Keywords

Anonymization, Social Networks, Weighted network models, Shortest paths, Linear Programming.

1. INTRODUCTION

Social Networks have become increasingly popular applications in Web 2.0. Social networks such as MySpace (www.myspace.com), Facebook (www.facebook.com), LinkedIn (www.linkedin.com), and Orkut (www.orkut.com) have millions of registered users, and each user is associated with a number of others through friendship, professional association (being members of communities), common interests, and so on. The resulting graph structures have millions of vertices (users or social actors) and edges (social associations). Recent research has explored these social networks for understanding their structure [10, 4, 1, 22], for criminal intelligence [24], information discovery [2], advertising and marketing [13], and others [9]. As a result, companies (such as Facebook) hosting the data are interested in publishing portions of the graphs so that independent entities can mine the data. In order to protect the privacy of the users against different types of attacks [3, 15], graphs should be anonymized before they are published. Consequently, there has also been considerable interest in the anonymization of graph structured data [5, 6, 12, 19, 31]. But most of the existing research on anonymization techniques tend to focus on *unweighted* graphs for *node* and *structural anonymization*.

Are social graphs weighted? Recently, there has been considerable interest in the analysis of the *weighted* network model where the social networks are viewed as weighted graphs. The weighted graph model is used for analyzing the *formation of communities* within the network [17], *viral and targeted marketing and advertising* [13], *modeling the structure and dynamics* such as opinion formation [28], and for analysis of the network for *maximizing the spread of information* through the social links [14], in addition to the traditional applications on weighted graphs such as *shortest paths*, *spanning trees*, *k-Nearest Neighbors (kNN)* etc. The semantics of the edge weights depend on the application (such as users in a social network assigning weights based on "degree of friendship", "trustworthiness", "behavior", etc.), or the property being modeled (such as detection of communities [17] or modeling network dynamics [28]). For example, consider the **"Los Angeles"** community in Facebook. If we consider that edge weights are inverse of "trustworthiness" (smaller weights correspond to higher trust in the relation), then the *kNN* query at a particular vertex returns the k most trusted users associated to the queried user, and the *single source shortest paths tree* provides the most trusted paths within the community which might be used for communicating while minimizing chances of a leak. Similarly, if we consider a routing problem (for information spread and marketing) where edge weights correspond to cost of information propagation, then the shortest paths minimizes the cost of information transfer.

Edge-weight anonymization: why do we care? *First*, even though in most cases node identities are anonymized, there are a number of instances where they are public knowledge. For example, if we consider the **"Los Angeles"** community, the members and the link structures of the members (i.e., their connections) are known to any user who is also a member of the community. But the edge weights, such as "trustworthiness" of user A according to user B, is private

information (in this case for $B \rightarrow A$). Therefore, for publishing the graph, anonymization of the edge weights is critical, while node identity anonymization might not be needed. Similarly, in academic social networks [27, 6], node identities and link structure are public knowledge, but edge weights are sensitive. *Second*, even in the case where the node identities are anonymized, we assert that edge weight anonymization is still important. This is because as demonstrated by Backstrom et al. [3], if the adversary has prior access to the graph, then an attack can be devised for re-identification of nodes in the anonymized graph. Hence, if the edge weights are also not anonymized, re-identification of a node in the anonymized graph will reveal even more information. Therefore, unless effective anonymization techniques are devised, the wealth of information contained in the weighted social network graphs will remain buried inside the companies hosting the networks. In this paper, we provide a solution to this problem.

But what is anonymized and what is preserved? As noted earlier, weights vastly increase the utility of the social graph structures, and it is these properties of edge weights that must be preserved across anonymization. For example, if we consider applications such as *shortest paths*, or *kNN*, then the anonymization should preserve some notion of *relative distances* between the nodes in the graph. On the other hand, if we consider applications such as maximizing spread of influence [14], then properties that can be formulated in the form $\sum_{v \text{ neighbor of } u} w_{u,v} \leq \theta_u$ [11] need to be preserved. What is ultimately desirable is *to have an anonymized graph which is as useful as the original graph in terms of the property being preserved, while revealing as little information as allowed by the semantics of the property being preserved* [23].

Privacy preserving modeling. Our solution to the problem of edge weight anonymization is to model the weighted graph based on the property to be preserved, and then reassign edge weights to obtain the anonymized graph satisfying the model. To be specific, we preserve *linear* properties of the graph:

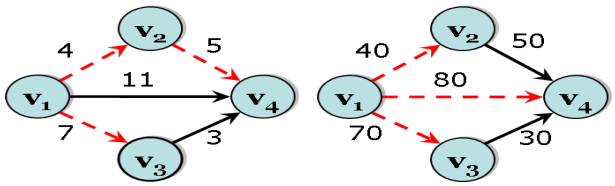
DEFINITION 1.1. *A **linear property** of a graph is a property expressible in terms of inequalities involving linear combinations of edge weights.*

If we consider that the anonymized graph preserves the structure of the original graph, the objective of the privacy preserving model can be formally stated as:

OBJECTIVE 1.1. *To construct a model that **correctly** captures the inequalities that must be obeyed by the edge weights for the **linear property** being modeled to be preserved. Any solution to such a model would ensure anonymization of edge weights, while preserving the linear property under consideration.*

As noted earlier, graph properties such as minimum spanning tree, shortest paths, nearest neighbors, graph clustering, maximizing information spread etc., are *linear properties*, and hence our approach is general enough to model and anonymize a large class of graph algorithms. In this paper, as a proof-of-concept, we consider the *shortest paths problem* since it is a problem of great interest in edge weighted graphs. It is also useful in modeling other properties such as *kNN* and community formation within complex network models.

Edge weights have been anonymized – now what? Once the model is created and a solution (edge weights) satisfying the model is obtained, the anonymized graph is guaranteed to preserve the property being modeled. For example, if we model shortest paths, the anonymized graph will have the same shortest paths as in the original graph. Consequently, the owner of the social network can publish the **"Los Angeles"** community graph without worrying about privacy breaches, while allowing applications to meaningfully process the anonymized graphs and extract useful information [17, 28]. If the node identities are not anonymized, then the shortest path in the anonymized graph can be used to spread information through entities in the original social graph. If node identities are anonymized, it still allows the extraction of useful information [17, 28] from the graphs without privacy breaches.



(a) Original graph (b) Naïve anonymization

Figure 1: Preserving only the order of edge-weights cannot model shortest paths tree. The dashed edges in the graph represent the shortest paths tree with v_1 as the root. In both the graphs, we have $w[3,4]<w[1,2]<w[2,4]<w[1,3]<w[1,4]$. This approach preserves the minimum cost spanning tree.

Contributions.

- We propose a framework for edge weight anonymization of graph structured data that preserves *linear properties*.
- As a proof-of-concept, we choose *shortest paths problem* which forms the basis of a number other graph properties. We use *single source shortest paths tree* as a stepping stone. The solutions to the individual *single source shortest paths trees* for all the nodes can be combined to model the *all pairs shortest paths problem*. The composability of individual solutions in this way demonstrates the extensibility of the proposed model to other *linear properties*, and preserve multiple properties in a single anonymized graph.
- We prove the correctness of the proposed models, provide a thorough analysis of the complexity of the proposed models, and present the results of experiments on real social network graphs that validate this analysis, while confirming that the anonymity of the sensitive information is preserved.

Organization. Section 2 introduces the abstract technique for privacy preserving modeling of weighted graphs while providing a simple model for *minimum spanning tree* as an example. Section 3 introduces the measures used to quantify the anonymity provided by the models. Section 4 explains the algorithm for modeling the graph for *single source shortest paths tree*. Section 5 extends these models to solve the *all pairs shortest paths* problem. Section 6 presents an experimental evaluation of the proposed techniques using real social network data sets, Section 7 provides a survey of related work in social network mining and anonymization, and Section 8 concludes the paper.

2. ABSTRACT MODEL

Naïve modeling. Unlike node anonymization, where a random assignment of identifiers is a possible option for anonymization, with edge weight anonymization, a simple random assignment of weights would be useless as it would not preserve any properties

$G = (V, E, W)$	Weighted graph to be anonymized
$G' = (V, E, W')$	Anonymized graph, W' satisfies the model
n, $\|V\|$	Number of vertices in the graph
m, $\|E\|$	Number of edges in the graph
d	Average degree of the vertices of the graph
$w[u, v]$	Original weight of edge (u, v), in G
$w'[u, v]$	Anonymized weight of edge (u, v) in G'
$P[u, v]$	Path from vertex u to v in the graph G
$D[u, v]$	Cost of $P[u, v]$, $\sum_{(u',v') \in P[u,v]} w[u', v']$
$\Pi[v]$	Predecessor of v in the shortest paths tree
T_i	Shortest paths tree with v_i as the source
$x_{(u,v)}$	Variable corresponding to edge $(u, v) \in E$
$f(u, \ldots, v)$	$\sum_{(u',v') \in P[u,v]} x_{(u',v')}$
μ	Indistinguishability threshold for *k-anonymity*
N_u	Edge neighborhood of a vertex

Table 1: Notational Conventions.

of the original graph. An alternative but still naïve anonymization attempt is to randomly reassign edge weights compatible with the given linear ordering of the weights. But this naïve approach is restrictive in terms of both preserving the properties, as well as the extent of anonymization it provides. In terms of restrictiveness, preserving only the order of edge-weights will work for simple applications like *minimum spanning tree*, but cannot be extended to *shortest paths tree*, *kNN* etc. As an example, consider the graph in Figure 1; the order of the edge weights in the original graph (Figure 1(a)) is preserved in the anonymized graph (Figure 1(b)), but the shortest paths tree in the two graphs are not identical. This method is also deficient in terms of the anonymity it provides since the relative order is preserved. In terms of the extent of anonymization provided, unintended information is seen to be revealed in the anonymized graph: the ordering of the neighbors of each node, or the relative "trustworthiness" among a set of friends, for example. These shortcomings call for stronger models. In the rest of this section we introduce, in abstract, a stronger modeling technique.

Abstract model formulation. As noted in Section 1, our proposed model is based on the observation that a gamut of interesting properties are expressible in terms of linear combinations of edge weights. For example, applications involving maximization of the spread of influence [14, 11], use properties of linear combination of edge weights (such as $\sum_{v \text{ neighbor of } u} w[u, v] \leq \theta_u$). Similarly, applications such as *shortest paths*, *kNN*, *minimum cost spanning tree*, etc., are expressable in terms of linear properties of graphs. In this section, we introduce in abstract the technique used for modeling *linear properties* and use Kruskal's algorithm for *minimum spanning tree (MST)* [16] as an example algorithm being modeled. The goal of the model is to capture the dynamic behavior of the algorithm using a system of linear inequalities. Given the original weighted graph $G = (V, E, W)$ with positive edge weights represented by variables $x_1, x_2, \ldots, x_m$ (where each x_i corresponds to an edge $i = (u, v) \in E$; refer to Table 1 for notational conventions), our goal is to model the system of linear inequalities in terms of these variables. For example at every step of the Kruskal's algorithm [16] for the MST, the edge with the minimum weight amongst the set of remaining edges is selected, and if this edge does not result in a cycle, it is added to the MST. Let (u, v) be the edge selected at the i^{th} iteration, and (u', v') be the edge selected in the $(i + 1)^{th}$ iteration, then this implies that $w[u, v] \leq w[u', v']$. If $x_{(u,v)}$ and $x_{(u',v')}$ are the variables representing these edges in the model, then this outcome is modeled by the inequality $x_{(u,v)} \leq x_{(u',v')}$. Therefore, for the MST, any solution to the system of inequalities constructed by taking for every pair of edges selected in consecutive iterations (u, v) and (u', v'), the inequality $x_{(u,v)} \leq x_{(u',v')}$ whenever the given weights satisfy $w[u, v] \leq w[u', v']$, preserves the MST.

The algorithm makes decisions based on the actual numerical values of the edge weights (or $w[u, v]$'s) and we model this decision in terms of the variables $x_{(u,v)}$. Decisions made at each step of the algorithm can similarly be expressed as inequalities involving the edge-weights. Thus, the execution of the algorithm processing the graph can be modeled as a set of linear inequalities involving the edge weights as *variables*, and this results in a system of linear inequalities:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{km} \end{pmatrix}}_{\mathbb{A}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}}_{\mathbb{X}} \leq \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}}_{\mathbb{B}} \quad (1)$$

If the edge weights are reassigned as any solution of the system of inequalities in (1), this would ensure that the properties of the graph remain unchanged w.r.t the algorithm being modeled. The model can therefore be formulated as a Linear Programming (LP) problem

$$\begin{aligned} \text{Minimize (or Maximize)} \quad & F(x_1, x_2, \ldots, x_m) \\ \text{subject to} \quad & \mathbb{A}\mathbb{X} \leq \mathbb{B} \end{aligned}$$

where F is a linear objective function. Any application that can be expressed as a function of a linear combination of edge weights can be expressed as a Linear Optimization problem, and hence this abstract modeling technique can be used for any such application. Once the model has been developed, any off-the-shelf LP solver package can be used to find a solution to the set of inequalities (constraints) that optimizes F. The model is said to be **correct** if the property being modeled is preserved across anonymization, i.e., any solution to the model ensures that the property being modeled is the same in the original graph as well as the anonymized graph. The **complexity** of the model is the number of inequalities necessary to define the model. Columns in the matrix $\mathbb{A}$ correspond to variables in the system, i.e., the number of edges in the graph, and rows correspond to the inequalities produced by the model. The fewer the constraints required by the model, the more efficient it is. Note that most social network graphs are sparse, and hence matrix $\mathbb{A}$ is also sparse, and LP solvers optimized for such large systems can be used. We remark that our technique is not dependent on the semantics of edge-weights, and is general enough to encompass any algorithm based on *linear properties* of the graph.

Choice of Objective Function. An added advantage of the LP formulation is that different objective functions F can be used to generate different solution sets, and hence different anonymized graphs. Since any solution to the LP model can act as anonymized weights, the actual objective function used is a free parameter. Stated otherwise, feasibility is sufficient for correctness of the proposed technique. Additionally, the variables can be assigned varying lower and upper bounds to attain different scalings as well as shifts in the values of the solution. Therefore, the publisher of the graph can publish anonymized versions of the same graph where the edge weights in each published version is different.

3. DATA SENSITIVITY AND PRIVACY PRESERVATION

This section formalizes the measures that quantify the extent of anonymity. Our goal is to anonymize the edge weights of the graph, while node identities and the structure of the graph remain unchanged. While anonymizing the edge weights, it is imperative that the anonymized edge weights should have little correlation to the original edge weight. The LP modeling allows a wide selection of objective functions, in addition to shifting the bounds of the variables and scaling the weights, thereby providing a lot of room for manipulating the edge-weights so that the magnitude of the anonymized weights have little correlation to the original weights.

In addition to altering the magnitudes of the edge weights, two additional properties are also important: *first*, how indistinguishable the weight of an edge is compared to the weights of other edges, and *second*, how different the ordering of the edge weights are in the original and anonymized graphs. The reason for indistinguishability is obvious, since a distinguishable edge-weight would aid re-identification of the edge and possibly its weight. Ordering of weights is sensitive for certain semantics of edge-weights. For instance, in the **"Los Angeles"** community example, let edge weights represent "trustworthiness", so for the link $A \rightarrow B$, its weight corresponds to how trustworthy B is according to A. So if A rates B as more trustworthy compared to C, then $w[A,B] > w[A,C]$. Evidently, this ordering is "sensitive" for all the involved users, and the anonymized graph should not reveal this ordering. In other words, anonymization should ensure that given an ordering in the anonymized graph, an adversary cannot determine the original odrder with high confidence.

Note that this ordering or indistinguishability of edges is important only in a *neighborhood*. For instance, ordering of $w[A,B]$ and $w[X,Y]$ is not important if A, B, X, and Y are not related. We therefore define an *edge neighborhood* of a vertex where ordering and indistinguishability is important.

DEFINITION 3.1. ***Edge neighborhood of a vertex.*** *The edge neighborhood of a vertex u, denoted as N_u, is the set of edges emanating from the vertex u, i.e., edges with u as the source.*

To address the privacy concerns, we use two well known metrics used in data privacy and statistics:

k-anonymity. *k-anonymity* is a well known metric used in data privacy for dealing with indistinguishability [26] of data values in an anonymized data sets. We use the following definition of *k-anonymity* in the context of edge weight anonymization:

DEFINITION 3.2. *An edge (u,v) is **k-anonymous** if there exists $k-1$ other edges (u,v') in the neighborhood N_u such that $\|w[u,v]-w[u,v']\| \leq \mu$, where μ is the indistinguishability threshold, i.e., the difference of weights below which two edge weights cannot be distinguished.*

Spearman rank correlation coefficient. The Spearman rank coefficient [25], denoted by ρ, is a statistical measure of the correlation of ranks or orders of two ranked data sets. Given two ranked data sets X and Y, ρ is computed as:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where $d_i = x_i - y_i$ is the difference between the ranks of corresponding values X_i and Y_i, and n is the number of items in each data set. The value of ρ lies between -1 and 1, $\rho = 1$ implies perfect correlation, $\rho = -1$ implies perfect negative correlation, and $\rho = 0$ implies no correlation between the two orders. Therefore, given a list of edges in the *edge neighborhood* of a vertex, for anonymizing the ranks or order of edge weights, values of ρ closer to 0 is desirable.

Algorithm 4.1 Dijkstra'a Algorithm: Shortest paths tree

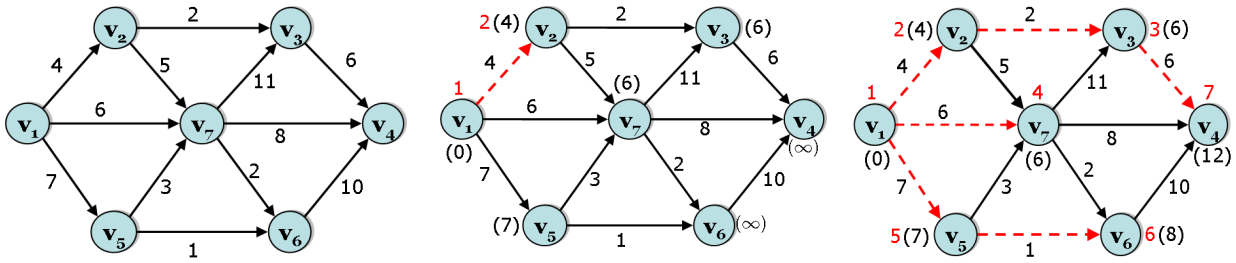
```
1:  D ← (∞) /* Cost of best known path from source. */
2:  Π ← () /* Predecessor in shortest path from source. */
3:  Q ← v0 /* Set of unvisited vertices */
4:  S ← φ /* Vertices to which shortest path is known. */
5:  D[v0, v0] ← 0
6:  while Q ≠ φ do
7:     u ← ExtractMin(Q) /* Unvisited vertex with min cost */
8:     S ← S ∪ {u}
9:     for each vertex v such that (u, v) ∈ E and v ∉ S do
10:       if D[v0, v] > D[v0, u] + w[u, v] then
11:          D[v0, v] ← D[v0, u] + w[u, v]
12:          Π(v) ← u /* Shorter path exists. */
13:       else
14:          /* Do Nothing. */
15:       end if
16:       if v ∉ Q then
17:          Q ← Q ∪ {v}
18:       end if
19:    end for
20: end while
```

4. SINGLE SOURCE SHORTEST PATHS

In this section, we demonstrate how the abstract model described in Section 2 can be used for *single source shortest paths tree*. Given a weighted graph $G = (V, E, W)$, and a source vertex v_0, a *single source shortest paths tree* is a spanning tree of the graph where the path from the source to any other vertex in the tree is the shortest path between the pair in G. This tree is important in a number of applications; for example, if an application assigns weights to the edges based on inverse of "trustworthiness", then this tree will provide the paths with greatest "trustworthiness" for transferring critical information from a specific node.

The *single source shortest paths tree* problem can have various naïve anonymization schemes such as publishing the tree separately along with an unweighted version of the graph. Our motivation for solving this problem separately derives from the following: *First*, the *single source shortest paths tree* algorithm subsumes the *k-nearest neighbors* query, since given the shortest paths tree from node v_0, we can determine the top-k nearest neighbors in increasing order. Our proposed solution preserves this additional property which many naïve solutions cannot preserve. *Second*, this algorithm forms the basis for the *all pairs shortest paths* problem and we use this as a stepping stone towards this goal. *Third*, our composition of the *single source shortest paths trees* to model *all pairs shortest paths* problem demonstrates the composability of the models. For instance, this allows the model for *all pairs shortest paths* to be combined with the model for *minimum cost spanning tree*, and the resulting anonymized graph preserves both these properties without the need to publish two separate graphs each preserving one property.

Dijkstra's algorithm [7] is a well known greedy algorithm for *single source shortest paths tree* and Algorithm 4.1 provides an overview. Given a start vertex v_0, at every step the algorithm selects the vertex u with the smallest known cost from v_0. The algorithm tries to "relax" the neighbors of u by checking to see if the cost



(a) Original weighted graph (b) Dijkstra's algorithm in progress (c) After completion

Figure 2: Illustration of Dijkstra's algorithm. The numbers adjoining the vertices and outside parenthesis correspond to the order in which the vertices were selected by Dijkstra's algorithm, the number in parentheses correspond to the cost of the best known path from the source, and the dashed edges constitute the shortest paths tree.

Algorithm 4.2 Linear Complexity model

1: $D \leftarrow (\infty)$ /* Cost of best known path from source. */
2: $\Pi \leftarrow ()$ /* Predecessor in shortest path from source. */
3: $Q \leftarrow v_0$ /* Set of unvisited vertices */
4: $S \leftarrow \phi$ /* Vertices to which shortest path is known. */
5: $D[v_0, v_0] \leftarrow 0$
6: $u' \leftarrow \phi$ /* Stores the vertex processed in previous iteration */
7: **while** $Q \neq \phi$ **do**
8: $u \leftarrow$ *ExtractMin*(Q)
9: $S \leftarrow S \cup \{u\}$
10: **if** $u' \neq \phi$ **then**
11: *AddConstraint*$(f(v_0, u') \leq f(v_0, u))$
12: **end if**
13: $u' \leftarrow u$
14: **for** each vertex v such that $(u, v) \in E$ and $v \notin S$ **do**
15: **if** $D[v_0, v] > D[v_0, u] + w[u, v]$ **then**
16: $D[v_0, v] \leftarrow D[v_0, u] + w[u, v]$
17: $\Pi(v) \leftarrow u$
18: *AddConstraint*$(f(v_0, v) > f(v_0, u) + x_{(u,v)})$
19: **else**
20: *AddConstraint*$(f(v_0, v) \leq f(v_0, u) + x_{(u,v)})$
21: **end if**
22: **if** $v \notin Q$ **then**
23: $Q \leftarrow Q \cup \{v\}$
24: **end if**
25: **end for**
26: **end while**

from the source has now decreased because of the selection of u. Thus, the algorithm makes a decision when to relax a neighbor, and which vertex to select for the next iteration. Figure 2 shows an illustration of the execution of Dijkstra's algorithm on an example graph, and the resulting tree. We will show how the decisions made by Dijkstra's algorithm can be modeled. For notational conventions refer to Table 1. Recall that $D[u, v]$ is the cost of the path from the vertex u to v, and $f(u, v)$ is $\sum_{(u',v') \in P[u,v]} x_{(u',v')}$. In other words $f(u, v)$ is a shorthand for the expression for the path in terms of the variables representing the edges in the path.

4.1 Linear model

Algorithm 4.2 provides the pseudocode for generation of the proposed model. Dijkstra's algorithm [7] makes a number of decisions based on the outcome of comparisons of linear combinations of edge weights. These decisions can be modeled using the following three categories of inequalities:

- **Category I**: When processing edge (u, v), if $D[v_0, v]$ can be improved, then $D[v_0, v] > D[v_0, u]+w[u, v]$, add constraint $f(v_0, v) > f(v_0, u) + x_{(u,v)}$ (line 18 in Algorithm 4.2).
- **Category II**: When processing edge (u, v), if $D[v_0, v]$ can not be improved, then $D[v_0, v] \leq D[v_0, u] + w[u, v]$, add constraint $f(v_0, v) \leq f(v_0, u) + x_{(u,v)}$ (line 20 in Algorithm 4.2).
- **Category III**: When processing $u \leftarrow$ *ExtractMin*(Q), if u' is the previous vertex processed, then $D[v_0, u'] \leq D[v_0, u]$, add constraint $f(v_0, u') \leq f(v_0, u)$. This captures the order in which the vertices are selected (line 11 in Algorithm 4.2).

THEOREM 4.1. *A model built from all the inequalities of Categories I, II, and III combined will correctly model Dijkstra's algorithm, i.e., any solution to the model used to anonymize edge weights in the graph results in the same shortest paths tree in the original as well as the anonymized graph.*

PROOF. **Proof by Contradiction.** Let $G = (V, E, W)$ be the input graph, and $G' = (V, E, W')$ be the anonymized graph. Let T_0 be the shortest paths tree starting at vertex v_0 in G and T_0' be the corresponding tree in G'. By way of contradiction, assume that T_0 and T_0' are different. Let v be a vertex where T_0 and T_0' differ, and let u be its predecessor in T_0, and u' in T_0' such that $u \neq u'$. Since u is the predecessor of v in G and since (u, v) and $(u', v) \in E$, we must have:

$$D[v_0, u] + w[u, v] = D[v_0, v] \quad (2)$$

$$\text{and,} \quad D[v_0, u'] + w[u', v] \geq D[v_0, v] \quad (3)$$

The model will contain constraints corresponding to properties 2 and 3 under Category II. Again, as u' is the predecessor of v in G', and since (u, v) and $(u', v) \in E$, we have:

$$D'[v_0, u'] + w'[u', v] = D'[v_0, v] \quad (4)$$

$$\text{and,} \quad D'[v_0, u] + w'[u, v] \geq D'[v_0, v] \quad (5)$$

Since W' is a solution of the model, properties 4 and 5 will be satisfied only if $u = u'$, which is a contradiction. □

Complexity of the Model. Category I and Category II combined will result in $O(dn)$ inequalities. This is because, when an edge is processed, either the path to its neighbor is improved (Category I), or it remains unchanged (Category II), and hence every edge results in at least one inequality. Since the average degree per node is d,

the resulting number of inequalities is $O(dn)$. The number of inequalities for Category III is $O(n)$ since one inequality of Category III is generated for every vertex processed. Thus, the complexity of the model is $O(dn)$. Since most large real graphs are sparse, i.e., $d \ll n$ (generally d is of the order of tens or hundreds), we refer to this model as the *Linear model* with complexity growing linearly with n.

4.2 Reduced model

In this section, we improve the model explained in the previous section by reducing its complexity. Note that even though Dijkstra's algorithm tries to relax the neighbors when processing a vertex, the ultimate goal is to select an appropriate vertex for the next iteration, i.e., the vertex with the smallest known cost from the source. It does not matter how many times the cost of the path to a particular vertex is improved, the minimum amongst these costs determines its order of selection, and hence the shortest path from the source. Category III inequalities model this information in an efficient way, and hence ideally, only Category III are needed. However Category III inequalities only include the edges that are part of the shortest paths tree. Therefore, if **only** Category III inequalities are considered in the model, then only part of the total number of edges are modeled. The model does not put constraints on non-tree edges, and thus, if no care is taken while reassigning edge weights in the anonymized graph, it can lead to violations of the order in the anonymized graph. For instance, if edge (u, v) is a non-tree edge, then a model using only Category III would not impose any constraint on (u, v). Hence a reassignment of weights in the anonymized graph might assign the edge (u, v) a weight such that Dijkstra's algorithm executing on the anonymized graph selects (u, v) as a tree edge. The following example illustrates this.

EXAMPLE 4.1. *Let us consider the example graph in Figure 2. As shown in Figure 2(c), after the execution of the algorithm, we have the order in which the vertices are selected, and edges that constitute the shortest paths tree. The inequalities for modeling the order would be:*

$$D[v_1, v_1] \leq D[v_1, v_2] \leq D[v_1, v_3] \leq D[v_1, v_7] \leq D[v_1, v_5] \leq D[v_1, v_6] \leq D[v_1, v_4]$$

$$\textit{i.e.,} f(v_1, v_1) \leq f(v_1, v_2) \leq f(v_1, v_3) \leq f(v_1, v_7) \leq f(v_1, v_5) \leq f(v_1, v_6) \leq f(v_1, v_4)$$

*It can be seen that if **only** Category III inequalities are considered in the model, then only part of the total number of edges are modeled. To be specific, Category III inequalities include only the edges which are part of the shortest paths tree (dashed edges in Figure 2(c)). The model does not put constraints on non-tree edges, and thus, if no care is taken while reassigning edge weights in the anonymized graph, it can lead to violation of the order in the anonymized graph. If we consider the original graph, edge (v_2,v_7) is not part of the model, so the solution to the model will not impose any constraint on (v_2,v_7), and as a result if it so happens that in the anonymized graph, $w'[2, 7]$ is set a value such that $w'[1, 2] + w'[2, 7] < w'[1, 7]$ then Dijkstra's algorithm running on the anonymized graph will select v_2 as the predecessor of v_7 instead of v_1 as in the original graph.*

Therefore, to ensure correctness, the model must be augmented to make sure that the non-tree edges are not included in the tree when the algorithm executes on the anonymized graph. The following theorem formalizes this proposition.

Algorithm 4.3 Reduced model

```
1: /* Initialize similar to Dijkstra in Algorithm 4.2. */
2: T ← φ /* Set of edges in the Tree. */
3: while Q ≠ φ do
4:    u ← ExtractMin(Q)
5:    S ← S ∪ {u}
6:    if (Π(u), u) ∉ T then
7:       T ← T ∪ {(Π(u), u)}
8:    end if
9:    if u' ≠ φ then
10:      AddConstraint(f(v0, u') ≤ f(v0, u))
11:   end if
12:   u' ← u
13:   for each vertex v such that (u, v) ∈ E and v ∉ S do
14:      if D[v0, v] > D[v0, u] + w[u, v] then
15:         D[v0, v] ← D[v0, u] + w[u, v]
16:         Π(v) ← u /* Shorter path exists. */
17:      end if
18:      if v ∉ Q then
19:         Q ← Q ∪ {v}
20:      end if
21:   end for
22: end while
```

THEOREM 4.2. *A model which ensures that (i) the order of selection of vertices remains the same even after anonymization, and (ii) non-tree edges in the original graph are not included in the tree constructed on the anonymized graph, will also ensure that the shortest paths tree in the original and anonymized graph are also same, i.e., the model is correct.*

PROOF. **Proof by Contradiction.** Let $G = (V, E, W)$ be the input graph, and $G' = (V, E, W')$ be the anonymized graph. Let T be the shortest paths tree starting at vertex v_0 in G and T' be the corresponding tree in G'. Let us assume that T and T' are different. Let v be first vertex where T and T' differ, and let u be its predecessor in T, and u' in T' such that $u \neq u'$. Then the following two possibilities arise:
Case I: The edge $(u, v) \in T$, and $(u', v) \notin T$. Now if u' is the predecessor of v in T', then $(u', v) \in T'$. But this is a contradiction since (ii) ensures that if $(u', v) \notin T \Rightarrow (u', v) \notin T'$.
Case II: Both edges (u, v) and (u', v) are in T. If (u', v) is a directed edge, then this is not possible since vertex v can have only one predecessor in T which is u, and since (u', v) is a directed edge towards v, it cannot be included in the path to some other vertex processed after v. If (u', v) is undirected, then it is possible only if in T, v is the predecessor of u'. But a vertex v can become a predecessor only for vertices that are processed after v. This implies that in G, v is processed ahead of u'. But if u' is the predecessor of v in T', then in G', u' is processed ahead of v, which again is a contradiction to the condition (i). □

Augmenting the model – Complexity and Correctness. Category III inequalities enforce condition (i) of Theorem 4.2. A simple solution to ensure that condition (ii) is also satisfied is to add all the non-tree edges into the constraints of the model. This can be done as follows: let v_l be the last vertex to be processed by Dijkstra's algorithm, and let T_s represent the shortest paths tree obtained as output from the algorithm, then add all inequalities of the form:

$$\forall (u, v) \in E \wedge (u, v) \notin T_s, \quad AddConstraint(x_{(u,v)} > f(v_s, v_l)) \tag{6}$$

Algorithm 4.4 Reassignment of weights in Reduced model

Require: v_l is the last vertex processed by Algorithm 4.3
1: **for** each edge $(u, v) \in E$ **do**
2: **if** $(u, v) \in T$ **then**
3: $w'[u, v] \leftarrow$ Value obtained from solution of model.
4: **else**
5: /* v_s is the source vertex. */
6: $w'[u, v] \leftarrow D'[v_s, v_l] + rand()$
7: **end if**
8: **end for**

This ensures that any path which includes these non-tree edges will have a cost greater than the corresponding path involving only the edges in T_s, and hence all such paths with non-tree edges will not be selected by Dijkstra's algorithm running on G'. The $O(n)$ edges in T_s are modeled by Category III inequalities, and the remaining $O(dn)$ edges are modeled by the inequalities in (6). Thus, the complexity of the model still remains $O(dn)$, even after eliminating inequalities of Categories I and II. Note that the inequalities in (6) add very little to the model except for ensuring that any non-tree edge should be assigned a weight that is greater than $D'[v_s, v_l]$, and it does not really matter what weight is assigned to these edges as long as the above condition is satisfied. Therefore, the edges not in T_s need not be part of the model, as long the edges are in T_s are tracked, and when assigning weights to the anonymized graph, non-tree edges are assigned weights greater than $D'[v_s, v_l]$. This captures the information as modeled by the constraints in (6), while not adding to the complexity of the model to be solved by the LP solver. Thus, Category III inequalities along with some additional information can model Dijkstra's algorithm, and the complexity of the modified model becomes $O(n)$ ($n - 1$ to be exact). The pseudocode for the modified algorithm is shown in Algorithm 4.3, while Algorithm 4.4 is one possible scheme for weight reassignment. The asymptotic complexity of the models in this section and in Section 4.1 are the same: both grow linearly with n (assuming that d is a constant compared to n). But considering the fact that d is generally of the order of 10 or 100 (as shown in our experiments using social network graphs), the model suggested in this section provides 1 to 2 orders of magnitude reduction in the number of inequalities.

5. ALL PAIRS SHORTEST PATHS

In the previous section, we presented models for the *single source shortest paths tree*. In this section, we build on the proposed models to solve the *all pairs shortest paths* problem [7]. Shortest paths trees with multiple source vertices can be modeled by repeated application of Dijkstra's algorithm for single source shortest paths. All pairs shortest paths is then the case where every vertex in the graph is considered as a source. This problem is important for various applications involving content distribution. Orkut and LinkedIn, for example, display the degree of separation (the number of edges on a shortest path) amongst any pair of users.

In this section, we discuss ways of applying the techniques we have described for the single-source shortest paths tree to the all-pairs shortest paths problem. Refer to Table 1 for notational conventions. The proposed abstract modeling technique can be used for the Floyd-Warshall [8] algorithm for all-pairs shortest paths, since the Floyd-Warshall is also based on linear properties. We use Dijkstra's algorithm in this paper for two reasons. *First*, rather than starting from scratch, we can build on the models developed in the previous section. *Second*, Dijkstra's algorithm has additional properties (described in Section 5.4) which makes it better suited for certain applications.

5.1 Naïve composition of single source model

The simplest approach is to apply the **Reduced** algorithm (Section 4.2) for all the vertices of the graph and combine the resulting models of the shortest paths trees. Since the constraints from different trees cannot contradict each other (we will prove this in the next section in Theorem 5.1), they can be combined. The edges that are not part of any tree can be assigned a sufficiently large value, just as the non-tree edges in Section 4.2. This approach is a straightforward extension of the principles developed in the previous section, but is not correct as described by the following counterexample.

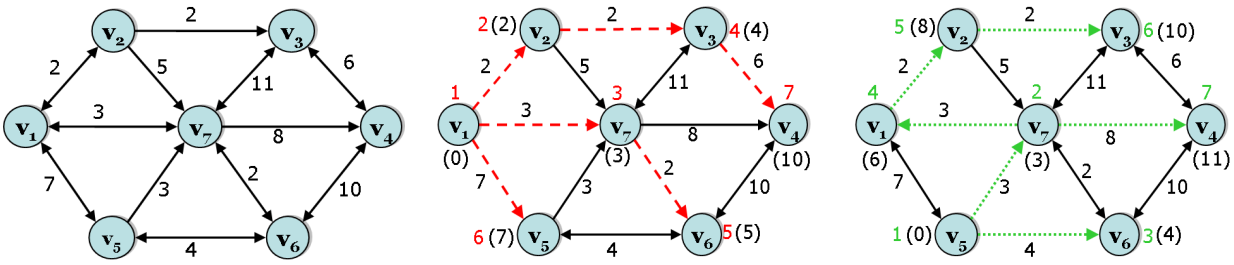
EXAMPLE 5.1. ***Counterexample.*** *Let us consider the weighted graph G as shown in Figure 3(a). Figure 3(b) shows the shortest paths tree for vertex v_1 (T_1) with the edges in the shortest paths tree represented in dashed lines, while Figure 3(c) shows the shortest paths tree for vertex v_5 (T_5) with edges in the shortest paths tree represented in dotted lines. When computing all pairs shortest paths trees for all the vertices, the constraints of all the trees are merged. Now, as is evident from the figure, there will be three types of edges.* ***First****, some edges like (v_1, v_2), which will be part of all trees. As noted earlier, and to be proved in the next section, since the constraints generated by the algorithm for different source vertices are not contradictory, the combination of the constraints do not pose any problem. A solution will satisfy each of the constraints individually, therefore it does not affect the outcome in the anonymized graph.* ***Second****, some edges like (v_7, v_3) which are not part of any of the trees. These can be handled similar to the scheme described in Section 4.2, and hence are not of concern.* ***Third****, some edges like (v_7, v_6) which are part in some of the trees and not in others. These seemingly innocuous edges render the suggested model incorrect. This is how it works: since these edges are part of at least one of the trees, they will be assigned values as obtained from the solution of the model. Again, since they are not part of all the trees, these solutions only satisfy the trees that contain these edges. Therefore, in this example, (v_7, v_6) belonging to T_1 has no constraints due to T_5, so there can be a solution such that $D'[5, 7] + w'[7, 6] \leq w'[5, 6]$. Therefore, when Dijkstra's algorithm is executed for vertex v_5 in the anonymized graph, it will set v_7 as the predecessor of v_6 instead of v_5 as in the original graph. This shows that the model is not correct.*

5.2 Quadratic solution

As demonstrated in Section 5.1, the **Reduced** model for single source shortest paths tree does not translate to a correct model for the all-pairs problem. In this section, we take the **Linear** solution explained in Section 4.1 and show how that model can be extended to the all-pairs problem. But before we proceed, we need to first prove the following:

THEOREM 5.1. ***Non-Contradictory Composition.*** *Composition of shortest paths trees and the constraints in the corresponding models do not result in contradictory constraints.*

PROOF. **Proof by Contradiction.** Let $G = (V, E, W)$ be the original weighted graph. Let T_1 be the shortest paths tree obtained after executing Dijkstra's algorithm for source vertex as v_1, and let S_1 be the set of constraints or inequalities. Similarly, let T_2 be the tree for vertex v_2 and S_2 be the set of inequalities. Let us assume that there exist a contradictory pair of constraints in $S_1 \cup S_2$, i.e., there does not exist a single solution for the set of constraints $S_1 \cup S_2$. Since the set S_1 is built based on the original set of weights W,



(a) Original weighted graph (b) Dijkstra's algorithm on v_1 (c) Dijkstra's algorithm on v_5

Figure 3: Illustration of Dijkstra's algorithm for *all pairs shortest paths*. The dashed edges corresponds to the shortest paths tree for vertex v_1, while the dotted edges correspond to the tree for v_5. Other conventions are similar to Figure 2.

hence W is a valid solution satisfying S_1 (and there are possibly many more solutions satisfying S_1). Using similar arguments, W also satisfies S_2. Since W satisfies S_1 and S_2 individually, hence W must also satisfy $S_1 \cup S_2$. This leads to a contradiction that $S_1 \cup S_2$ had a contradicting pair of inequalities. □

Theorem 5.1 is an extremely important result that shows composability and extendibility of the models developed in this paper to other models, and hence preserving *multiple* linear properties in the same anonymized graph. The following corollary provides a more general result, the proof of which follows the same logic as that of Theorem 5.1:

COROLLARY 5.2. ***Composability of Models.*** *The composition of linear programming models developed as extensions of the* **Abstract** *model (Section 2) and modeling different* linear properties *do not lead to contradictory constraints.*

Therefore, a simple solution for the all-pairs problem would be to generate the **Linear** model (as in Section 4.1) for all the vertices $v_1, v_2, \ldots, v_n$, obtain the set of constraints $S_1, S_2, \ldots, S_n$, and then obtain the model for all-pairs as $S_1 \cup S_2 \cup \cdots \cup S_n$. Since each of the S_i's provide constraints on *all* edges, hence the constraints from one *single source shortest paths tree* cannot set values to edges which result in another tree being inadvertently modified. This overcomes the problem that arises in the naïve merging of the **Reduced** model discussed earlier in this section.

THEOREM 5.3. *A model comprised of all the constraints generated by the* ***Linear*** *solution for single source shortest paths tree, repeated for all the vertices of the tree, is a correct model for the all-pairs problem.*

PROOF. **Proof by Contradiction.** Let $G = (V, E, W)$ be the input graph, and $G' = (V, E, W')$ be the anonymized graph. Let us assume that there exists at least one pair of vertices v_i, v_j whose shortest paths in G' differs from its shortest path in G. The shortest path from v_i to v_j in the all-pairs problem is the path from v_i to v_j in the single source shortest paths tree with v_i as the source, i.e., T_i. This implies that T_i in G does not match T_i' in G', which is a contradiction of Theorem 4.1. □

Complexity of the Model. The complexity of the model can be derived trivially from the complexity of the constituting model. Each of the shortest paths trees have a complexity of $O(dn)$, and this repeated for n vertices gives us a total complexity of $O(dn^2)$.

5.3 Optimized solution

In the previous section, we presented a solution with complexity $O(dn^2)$. It is correct, but there are many redundant inequalities due to the composition. For example, edges that are not part of any of the trees need not be part of the model, and can be treated as the non-tree edges in Section 4.2. However in the described model, there are no means for filtering out these inequalities. In this section, we delve deeper into the problem of composition of the solution of Section 4.2 for the all pairs problem.

When merging the constraints of multiple trees developed using the **Reduced** model, some edges that are part of some but not all of the trees result in problems. We formalize this as follows:

DEFINITION 5.1. ***Problematic edges:*** *An edge (u, v) is said to be* ***problematic*** *for composition if there exists a shortest paths tree T_i such that $(u, v) \in T_i$, and there exists a tree T_j ($T_i \neq T_j$) such that $(u, v) \notin T_j$.*

A problematic edge $(u, v) \notin T_j$ will not have any constraint involving $x_{(u,v)}$ in the model developed for T_j (refer to Section 4.2), and hence the constraints of T_i (or any other tree T_k which contains (u, v)) can set a value $w'[u, v]$ in the anonymized graph such that when T_j' is reconstructed in the anonymized graph, (u, v) is selected as an edge in T_j'. There was a decision which the algorithm took when (u, v) was not included in T_j, but since (u, v) was not selected in T_j, this decision was not part of the model. But if we devise a mechanism to model this decision in T_j, then the edge will no longer be problematic for T_j.

PROPOSITION 5.4. ***Eliminating Problematic Edges:*** *The problematic edge (u, v) was not selected in T_j, since there exists another path from the source vertex v_j to v which is cheaper than the path from v_j to v through the vertex u, i.e., $D[v_j, v] < D[v_j, u] + w[u, v]$. If the corresponding constraint $f(v_j, v) < f(v_j, u) + x_{([u,v])}$ is added to the model of T_j, then (u, v) is no longer a problematic edge for T_j. Similarly, if the process is repeated for all trees T_k such that $(u, v) \notin T_k$, then (u, v) is no longer a problematic edge for any of the trees.*

If we consider the graph in Example 5.1, for edge (v_7, v_6) this will amount to adding the constraint $f(5, 6) < f(5, 7) + x_{(7,6)}$, which will address the problem illustrated in Example 5.1. Therefore, once we have made sure that the problematic edges are eliminated during the combination of the constraints of the individual trees, we combine the individual constraints to form the model for all-pairs shortest paths. Therefore, if $T_1, \ldots, T_n$ are the trees and $S_1, \ldots, S_n$ are the corresponding set of constraints, then we want to form $S = S_1 \oplus S_2 \oplus \cdots \oplus S_n$ which would model the all-pairs shortest paths problem. Algorithm 5.1 provides an overview of how the shortest paths trees and the constraints can be combined. Again, since at the end of the algorithm, T contains all the edges

Algorithm 5.1 Optimized model for *all pairs shortest paths*

```
1: Run Algorithm 4.3 for all vertices v_1, ..., v_n
2: T ← φ
3: S ← φ
4: for each T_i in {T_1, ..., T_n} do
5:    S ← S ∪ S_i
6:    for each edge (u, v) ∈ T_i do
7:       for each T_k in {T_1, ..., T_n} such that (u, v) ∉ T_k do
8:          S ← S∪ {f(v_k, v) < f(v_k, u) + x_(u,v)}
9:       end for
10:   end for
11:   T ← T ∪ T_i
12: end for
```

that are part of at least one of the trees, $E - T$ yields the non-tree edges, and then a technique similar to Algorithm 4.4 can be used to reassign weights to the anonymized graph.

The algorithm combines the trees one at a time, while making sure that the set of inequalities cannot produce problematic edges. In Algorithm 5.1, line 8 adds a constraint that ensures that the problematic edge (u, v) is eliminated, and after iteration i, all the constraints in the set S preserve the trees $T_1, \ldots, T_i$. Theorem 5.5 formally proves the correctness of the model.

THEOREM 5.5. *The model created by Algorithm 5.1 preserves all the trees $T_1, \ldots, T_n$.*

PROOF. **Proof by Mathematical Induction.**

Base Case. At the beginning of the algorithm, $T = \phi$ and $S = \phi$. Hence it is true trivially.

Inductive Case. Let us assume that after iteration i, we have T and set of constraints S that preserves trees $T_1, \ldots, T_i$, and at iteration $i + 1$, we are adding the tree T_{i+1}. Let us assume that (u, v) is a problematic edge. For every T_k such that $(u, v) \notin T_k$ ($T_k \in \{T_1, \ldots, T_n\}$), means that Dijkstra's algorithm did not pick (u, v) in T_k, and addition of the constraint in line 8 makes sure that Dijkstra's algorithm executing on the anonymized graph will not pick (u, v) as an edge in T'_k. This property exists in the original graph that made sure that (u, v) was not picked in any of T_k. Therefore, it is evident that when the edge (u, v) is added, the algorithm makes sure that it is not problematic, and hence at the end of the iteration, the set of constraints S preserves trees $T_1, \ldots, T_i, T_{i+1}$.

Therefore, by the principle of mathematical induction, the set of constraints at the end of the algorithm preserved the trees $T_1, \ldots, T_n$, and hence in the anonymized graph, all the trees can be reconstructed which are identical to the trees in the original graph. □

An interesting property to note here is that this combination of trees is incremental, and if the algorithm stops after iteration i, we can still reconstruct the trees $T_1, \ldots, T_i$ from the anonymized graph.

THEOREM 5.6. *A model that preserves the trees $T_1, \ldots, T_n$ correctly models the shortest path between all pairs of vertices.*

PROOF. **Proof by Contradiction.** Let $G = (V, E, W)$ be the input graph, and let $G' = (V, E, W')$ be the anonymized graph. Let us assume that there exists at least one pair of vertices v_i, v_j whose shortest paths in G' differs from its shortest path in G. The shortest path from v_i to v_j in the all-pairs problem is the path from v_i to v_j in the single source shortest paths tree with v_i as the source, i.e., T_i. This implies that T_i in G does not match T'_i in G', which is a contradiction, since the T_i is preserved by Theorem 5.5. □

COROLLARY 5.7. *The model generated by Algorithm 5.1 correctly models the shortest paths between all pairs of vertices.*

Complexity of the Model. The analysis of the complexity of the algorithm is a bit more involved. In the *best case*, all the trees have the same edges. Since there are no problematic edges, no new constraints were added, and hence the complexity is $O(n^2)$. In the *worst case*, every problematic edge will add $O(n)$ inequalities, and again, there can be at most $O(dn)$ problematic edges. Therefore, the number of added constraints are:

$$\underbrace{(n-1)+(n-1)+\cdots+(n-1)}_{dn \text{ terms}} + \underbrace{(n-1)+(n-1)+\cdots+(n-1)}_{n \text{ terms}}$$

Therefore, the total number of inequalities is $O(dn^2)$. Thus the complexity is no worse than the model described in Section 5.2. Our experimental evaluation on real datasets in Section 6 shows that this model performs significantly better on the average than $O(dn^2)$.

5.4 Incremental Algorithm

The incremental nature of the algorithm makes it suitable for a specific class of applications in which the client mining the graph may not be interested in the shortest paths for all pairs of vertices in the graph (or subgraph). As is evident from Theorem 5.5, the model is consistent at every step of the algorithm. Therefore, the algorithm can be terminated after processing trees $T_1, \ldots, T_k$ (where $k < n$) and the model is still consistent for these trees. In addition, the trees $T_1, \ldots, T_k$ can be arbitrarily chosen as well as composed in any arbitrary order. Note that each shortest paths tree can also be truncated after processing k of the n vertices, resulting in the k-nearest-neighbors of each node.

Consider the **"Los Angeles"** community example in Section 1. A client requesting the anonymized data corresponding to all the members in the **"Los Angeles"** community might only be interested in shortest paths between all pairs of "computer scientists". In such a scenario, only the shortest paths trees with "computer scientists" as roots need to be combined. If the number of trees $k \ll n$, then this technique will have a complexity of $O(kn)$, i.e., linear in the number of vertices in the graph. Similarly, if edge weights indicate "trustworthiness", then the method allows the client to request the top k "trustworthy" computer scientists according to a member in the community.

Parallelization for Multicores Since the *single source shortest paths tree* computation for one vertex is independent of the others, the first phase of the algorithm in which the individual shortest paths trees are computed, is *embarrassingly parallel*. Therefore, this phase can be easily parallelized. Considering the ubiquitous presence of multicores and the compute intensive nature of the algorithm, this should lead to significant improvements in the processing time for large graphs. We leave parallelization as a future extension to this work.

5.5 Implementation Issues

It is appropriate to address some of the subtleties of implementation for completeness. Every decision modeled results in an inequality. For example, in Dijkstra's algorithm, if vertex u' is processed before vertex u, then we output $f(v_0, u') \leq f(v_0, u)$ (Category III, Section 4.1). In order to deal with ties and different im-

Single source	All pairs
Linear: $O(dn)$	**Quadratic:** $O(dn^2)$
Reduced: $O(n)$	**Optimized:** $O(n^2)$ (best), $O(dn^2)$ (average)

Table 2: Summary of Complexity of the models.

plementations of queues, the ties in the original graphs should be modeled exactly in the same way in which it was resolved while generating the model in the original graph. Consequently, in this scenario, to make sure that in the anonymized graph u' is chosen ahead of u, we model the decision as $f(v_0, u') \leq f(v_0, u) - \epsilon$, where $\epsilon > 0$ is a small real number. Additionally, LP solvers do not accept strict inequalities of the type $f(x, y) < b$. Therefore, such inequalities are converted to non-strict inequalities as $f(x, y) \leq b - \epsilon$, where again $\epsilon > 0$ is a small real number.

6. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the different models presented in this paper, compare their performance, and validate our analysis. All the algorithms were implemented in Java, and the experiments were run on an Intel Core 2 Quad Q6600 processor operating at a clock speed of 2.4GHz. The machine has 4GB main memory (3.2GB available) and runs Fedora Core Linux with kernel 2.6.26.6-49.fc8. We used four real social network data sets obtained from the authors of [22]. In our experiments, we used a free open-source LP Solver (*lp_solve 5.5*) [21]. We report the time taken to generate the model, complexity of the model, and the time taken to solve the models. Since we use the LP Solver as a library, we assume that it is de-coupled from the system generating the model. Therefore, the model is written to disk, and the system solving the model reads the model from disk, and generates the solution, which is then used to anonymize the model. The reported times therefore include the disk access latencies. Most open source implementations of LP solvers are not heavily optimized, and are stable for smaller systems. There are commercial systems which are much faster than these open source implementations, and can also handle larger models. Correctness of the models is also experimentally validated by checking the equivalence of the shortest paths trees and all-pairs shortest paths in the original and the anonymized graphs.

We first provide details of the data sets used for the experiments, then experimentally evaluate the model for *single source shortest paths tree*, followed by that of *all pairs shortest paths problem*, and finally evaluate the privacy guarantees of the models. In our experiments, we focus more on the complexity of the models as it affects both the time to generate and to solve the models. Table 2 summarizes the complexity of the models developed in the previous two sections. As noted in Section 2, the objective function is a choice of either the publisher of the data set, or the client who will mine the data set. We experimented with a limited number of object functions coefficients such as setting all of then to unity (unity object function), or setting them to random values picked from uniform as well as Gaussian distributions, but no significant difference was observed. In the reported experiments, we use a unity object function.

6.1 Datasets

Mislove et al. [22] crawled a number of social network sites for analyzing the properties of these large social graphs, and have made their data sets publicly available. Their data sets include the graphs for a number of popular social networking sites: **Flickr** (`www.flickr.com`), **LiveJournal** (`www.livejournal.com`), **Orkut** (`www.orkut.com`), and **Youtube** (`www.youtube.com`). While Orkut is a pure social networking site, LiveJournal (referred to as **LJ** in the data sets) is a blogging site whose users form a social network, while Flickr and Youtube are photo sharing and video sharing sites respectively, with an overlayed social network structure amongst its users. We model the graphs of these networks as directed graphs where edges have positive weights, but the models can be extended for undirected graphs. Even though some social network graphs (such as Orkut) might have symmetric edges (i.e., an edge (u, v) means both u and v are connected to each other), the weights on either direction might not be the same (for example if edge weights are based on "trustworthiness", for the (u, v) edge, u's rating of v, or $w[u, v]$, might be different from $w[v, u]$). Even though we assume directed graphs in all our discussion, and use directed graphs for our experiments, the model can accommodate undirected graphs as well, and the algorithms in Sections 4 and 5 can be extended for undirected graphs as well. The published graph data sets are unweighted, but since our model is not dependent on the semantics of the weights or their magnitude, we assign randomly generated weights (real numbers in the range 1 to 100) to the edges of the graph. We used different distributions for assigning edge weights, but no considerable change in complexity was observed.

In our experiments, we consider different sub-graphs from the original graph data sets. There are multiple reasons for considering sub-graphs: *First*, both the applications involving shortest paths are more meaningful in a portion of a graph rather than the entire graph. For example, a user will hardly be interested in knowing the shortest paths to all other reachable vertices in the graph. Rather, shortest path to all members of a community to which the user belong to, or to all the vertices which are within k *degrees of separation* will be of more interest to a particular user. *Second*, the shortest paths algorithms considered in this paper are exact algorithms with very high time complexity, and hence do not scale to large graphs. Therefore, even if the entire anonymized graph is available, executing, for instance, Dijkstra's algorithm for all the vertices of the graph is extremely costly. *Third*, from the perspective of the companies such as Facebook hosting the social networks, the information content of the graphs drives their businesses, and releasing the entire graph structure can be detrimental to their business in addition to introducing a very high overhead on their engineering infrastructure. Therefore, they are more likely to release portions of their social graphs rather than the entire social graph.

We consider two specific forms of sub-graphs for our experiments:

User Driven Structures: These are sub-graphs where a specific user is of interest, and is useful for applications focussed on a user. For example, for marketing purposes, a company might select some *influential* users for free trials of their products so that they can influence other users to use or buy the product [14]. Similarly, applications such as *shortest paths trees* and *nearest neighbors* will also be interested in similar structures. To simulate these structures, we select a vertex in the graph as the root, and extract the graph induced by the vertices which are within k *degrees of separation* from the root (a vertex v is the first degree connection of the root v_0 if there exists an edge (v_0, v)). We use the *user* suffix for referring to the user data sets, and for our experiments, we consider 3^{rd} degree of separation (e.g., *Orkut-user-3*).

Community Driven Structures: These graphs correspond to communities (or groups) within the social networks. For example, in our examples in Section 1, we refer to the **"Los Angeles"** community in Facebook. Community structures are very important for applications such as *shortest paths, nearest neighbors, targeted ad-*

Data Set	No. of Vertices	No. of Edges	Avg. Degree
Flickr-user-3	55,803	6,662,377	119.39
LJ-user-3	15,508	384,947	24.82
Orkut-user-3	26,110	899,638	34.46
Youtube-user-3	237,469	2,457,206	10.35
Flickr-comm	1,382	69,321	50.16
LJ-comm	1,497	21,481	14.35
Orkut-comm	1,047	28,240	26.97
Youtube-comm	1,823	29,342	16.1

Table 3: Summary of the Social Graphs.

vertising etc. This is primarily since users in the same community share some common interests, and hence many applications can be driven by the community structure. For the experiments, we select communities inside the social networks, and extract the graph induced by the members of the community. We use the *comm* suffix for referring to the community data sets (e.g., *Orkut-comm*).

Table 3 summarizes the different data sets in terms of the number of vertices, number of edges, and average out-degrees[1]. To provide a better insight into the distribution of the out-degrees of the vertices, in Figure 4, we plot the *cumulative distribution function (CDF)* of the out-degrees of the graphs in the data set. Along the x-axis is the out-degree, and along the y-axis is the fraction of the total number of vertices whose out-degree is less than the corresponding value of the x-axis. Figure 4(a) plots the CDF for the *user driven graphs*, while Figure 4(b) plots the CDF for the *community driven graphs*. Each line in the figure corresponds to a graph in the data set, and represents the fraction of vertices that have out degree less than or equal to the corresponding point on the x-axis. As can be noted from Figures 4(a) and 4(b), Flickr data set has a considerably higher out degree compared to the other three data sets, and for the *user driven graphs*, about 12% of the vertices have an out degree higher than 250.

6.2 Single source shortest paths

In this section, we experimentally evaluate the models for *single source shortest paths tree* described in Section 4. The *shortest paths tree* is interesting for both types of data sets under consideration. For the *user driven* data sets, the shortest paths tree with the root of the graph as the source is of interest for both the shortest paths as well as the *k-nearest neighbors*. Similarly, for the *community driven* data sets, selecting an "influential" user and using it as the source of the shortest paths tree is again of interest for many applications. We therefore run our experiments on both types of data sets. We compare the **Linear** model developed in Section 4.1 to the **Reduced** model developed in Section 4.2 in terms of the complexity of the model, and the time taken to build the model and write it to the disk. Recall that the complexity of the model corresponds to the number of inequalities generated during modeling, and the time taken includes the time for execution of Dijkstra's algorithm, generation of the inequalities, and writing the generated inequalities to disk. Figures 5 and 6 provide a comparison of the two modeling techniques for both types of data sets for all the social graphs. Figures 5(a) and 5(b) compare the complexity of the models, while Figures 6(a) and 6(b) compare the time taken to build the model. In all the figures, the x-axis represents the social graphs, and the y-axis for Figures 5(a) and 5(b) plots the number of inequalities constituting the model, while the y-axis for Figures 6(a) and 6(b) plot the time in seconds. Note that the y-axis of all the plots have been plotted in logarithmic scale. It is evident from the figures that the **Reduced** model is extremely efficient compared to the **Linear** model both in terms of complexity and time. The complexity of the **Reduced** model is about 1 to 2 orders of magnitude lesser when compared to the **Linear** model and so is the time taken in computing the model.

Table 4 provides the results from these experiments along with a detailed breakup of the number of inequalities, as well as the reduction in complexity and time of the **Reduced** model compared to the **Linear** model. For the **Linear** model, the categories of inequalities in Table 4 correspond to the categories defined in Section 4.1. As is evident from Table 4, the **Reduced** model provides about $O(d)$ times improvement in complexity of the models for all the graphs, as observed in Section 5.3. Depending on the graph, the value of d varies, and so does the factor of improvement. For example, for the *Flickr-user-3* data set, d is 119.39, and the complexity of the **Reduced** model is about 120 times less than that of the **Linear** model. The large reduction in the number of inequalities also affects the time for generating the model, since in the **Linear** model, fewer number of inequalities need to be *generated*, and more importantly, fewer number of inequalities need to be *written to the disk*. This is illustrated by the almost 90% improvement in time to generate the **Reduced** model.

6.3 All pairs shortest paths

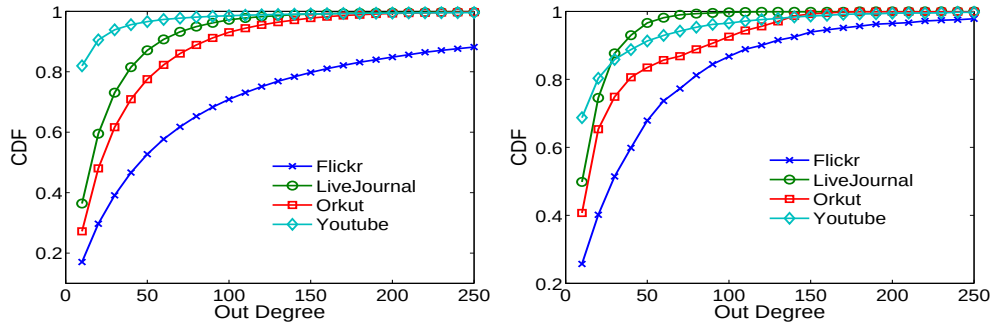
In this section, we experimentally evaluate the models for the *all pairs shortest paths problem* described in Section 5. In a community of a social network, users share common interests, and an application that uses minimum cost paths between any two members of the community would require the all-pairs shortest paths. On the other hand, for a *user driven* social graph, two users in the graph might be completely unrelated, and from an application's perspective, shortest paths between them is not interesting. Thus, we evaluate the models for *all-pairs* only for the *community driven graphs*. We first evaluate the model for shortest paths between all pairs of vertices in the graph, and then evaluate the *Incremental* algorithm discussed in Section 5.4 for shortest paths between a subset of vertices.

6.3.1 Evaluating shortest paths between all pairs

In this section, we experimentally evaluate the *all pairs* models (described in Sections 5.2 and 5.3). We refer to the model of Section 5.2 as the **Quadratic** model, and that of Section 5.3 as the **Optimized** model. Figure 7 compares the two models in terms of complexity and the time taken to build the model, and both of these terms have the same meaning as described in the previous section. In Figure 7(a), the number of inequalities in the models is plotted along the y-axis, while in Figure 7(b), the time taken (in seconds) to generate the model is plotted along the y-axis. In both the figures, the x-axis represents the different graph data sets, and again note that the y-axis is plotted in logarithmic scale. As noted in Section 5.3, Figure 7 illustrates the benefits of the **Optimized** model compared to the **Quadratic** model both in terms of complexity and time.

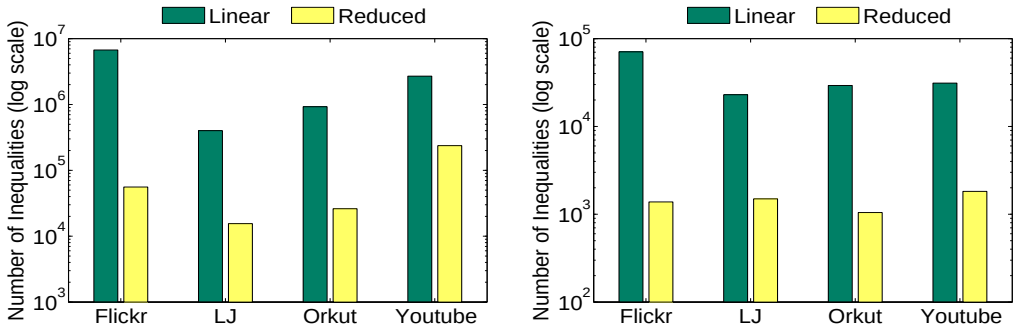
Table 5 provides the experimental results, tabulating the breakup of the categories of the constituent inequalities that make up the model. For the **Quadratic** model, the categories of the inequalities correspond to the ones defined in Section 4.1. For the **Optimized** model, the *Merge* inequalities are the ones generated when the individual shortest paths trees are merged into one consistent model compensating for the problematic edges, while the *Trees* inequalities are the total number of inequalities generated for the trees. Since this corresponds to *Category III* inequalities (as the **Reduced** model for single source only uses *Category III* inequali-

[1]Since we extract sub-graphs from the social network graphs, the average degree is not representative of the original graphs.



(a) User driven graph (b) Community driven graph

Figure 4: Cumulative Distribution Function for the out-degrees of the different graph data sets used for the experiments.



(a) *User* data sets. (b) *Community* data sets.

Figure 5: Complexity of the models for *single source shortest paths* tree.

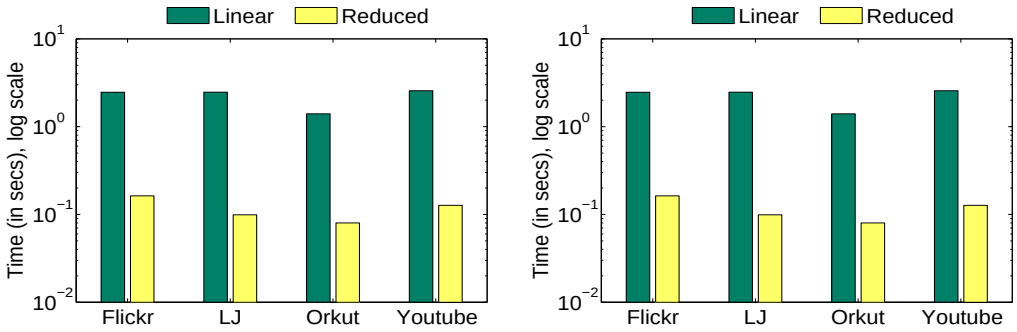
ties), Columns 4 and 8 of Table 5 are exactly identical. Table 5 also provides data that allows a deeper analysis of the reasons for the improved performance of the **Optimized** model, even though both models have the same complexity bound $O(dn^2)$. As was noted in Section 5.3, the inefficiency of the **Quadratic** model stems from the fact that it cannot leverage the absence of some edges from all the trees, which allows these edges to be excluded from the model. These edges are represented by the column titled *Unconstrained Edges* in Table 5. It can be seen that in all the social graphs, a high percentage of edges are not part of any tree, and eliminating these edges from the model considerably simplifies the model. This is evident from the 70–80% reduction in complexity of the **Optimized** model compared to the **Quadratic** model. As seen in the case of *single source shortest paths tree*, reduction in complexity of the model also considerably reduces the time, primarily because fewer inequalities are written to disk.

6.3.2 *Shortest paths for a subset of vertices*

In this section, we experimentally evaluate the **Incremental** all-pairs model as explained in Section 5.4. This kind of model finds application in *community driven* graphs where the application is interested in a specific subset of vertices. Our goal of this evaluation is to validate our analysis that if *all-pairs shortest paths* are not required by the application, we can reduce the complexity of the model considerably. For our experiments, we consider two cases, one where the pairwise shortest paths between a subset of 100 vertices is to be determined and the shortest paths tree for the rest suffices, and in another, we are interested in a subset of 200 vertices. These subsets are randomly selected, and simulate our example where the application is interested in "computer scientists" or "guitarists" in the **"Los Angeles"** community of Facebook. Table 6 summarizes the results from these experiments, and compares it with the numbers obtained for the **Optimized** model for all-pairs shortest paths from Table 5. The significant reductions obtained (reflected by the rightmost columns in Table 6) corroborates our claim that when shortest paths between all pairs of vertices is not required, the complexity of the model can be reduced significantly. The great reduction is primarily due to the fact that a huge portion of the inequalities for the merge phase is not required for the trees that are not of interest and therefore do not need to be merged.

6.4 Overall time overhead

In all the above experiments, we considered only the complexity of the model, and the time taken to generate the model. Once the model has been generated, it has to be solved to anonymize the graph. The time required for this step depends on the efficiency of the LP solver. We use an open source LP Solver [21] in our experiments, and it is widely acknowledged that commercial LP solvers are far more efficient compared to open source implementations. As example timings, for the **Reduced** model of *single source shortest paths tree* problem, the LP solver we used took 0.394 seconds to solve the model for *Orkut-comm* graph, 0.541 seconds for the *Youtube-comm* graph, 150.638 seconds for the *LJ-user-3* graph, and 629.869 seconds for the *Flickr-user-3* graph. For the all-pairs problem, where the complexity of the model rises to about $100K$ inequalities, the solvers took about an hour to find a solution. We remark that our open source LP solver is not optimized for solving large, sparse models, and these timings are not the best possible. Furthermore, solving the model constitutes an offline cost and hence the exact times are not significant for our evaluation.



(a) *User* data sets. (b) *Community* data sets.

Figure 6: Time to build the model for the *single source shortest paths* tree.

	Linear Model					Reduced Model		Summary	
	Number Inequalities				Time	Number of	Time	Times Reduction	% Reduction
Data Sets	Cat I	Cat II	Cat III	Total	Taken (s)	Inequalities	Taken (s)	in Complexity	in Time
Flickr-user-3	204,626	6,457,751	55,802	6,718,179	98.81	55,802	2.835	120.39	97.13
LJ-user-3	39,030	345,917	15,507	400,454	4.783	15,507	0.938	25.83	80.39
Orkut-user-3	72,130	827,508	26,109	925,747	15.735	26,109	1.752	35.47	88.87
Youtube-user-3	417,526	2,039,680	237,468	2,694,674	44.943	237,468	8.226	11.35	81.7
Flickr-comm	4,112	65,209	1,381	70,702	2.464	1,381	0.163	51.2	93.39
LJ-comm	3,148	18,333	1,496	22,977	2.471	1,496	0.099	15.36	95.99
Orkut-comm	2,409	25,831	1,046	29,286	1.401	1,046	0.08	27.99	94.29
Youtube-comm	3,605	25,737	1,822	31,164	2.564	1,822	0.127	17.11	95.05

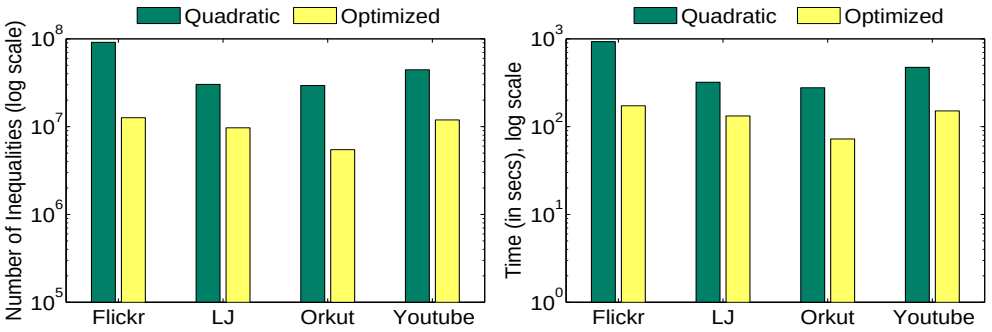
Table 4: Experimental evaluation of single source shortest paths tree.

6.5 Evaluating Data privacy

In this section, we evaluate the privacy preserving properties of the proposed models. In our evaluation, we use the two measures presented in Section 3, i.e., *k-anonymity* [26] and *Spearman rank correlation coefficient* [25]. As explained in Section 3, in the context of sensitivity of edge weights, both the measures are defined in a neighborhood. Computation of *k-anonymity* of edges follow directly from its definition. The *Spearman rank correlation coefficient* is computed for every vertex in the graph. For each vertex in the original graph and the corresponding vertex in the anonymized graph, the list of edges emanating from the vertex comprises the ranked lists used for computing the coefficient. The lists are sorted by the edge weights, and the coefficient measures correlation between the ranks of the edges in the two lists. Figures 8, 9, and 10 provide the experimental results for the two measures on the real data sets. In these experiments, we use the **Reduced** model for *single source shortest paths* tree and the **Optimized** model for *all pairs shortest paths* problem.

Figures 8 and 9 plot the percentage of edges in the graph that are *k-anonymous* in their neighborhood for a given value of k and indistinguishability threshold μ. Figure 8 plots the graphs of the model for *single source shortest paths* tree and Figure 9 plots the graphs of the model for *all pairs shortest paths* problem. Along the x-axis we plot the different values of k, and along the y-axis, we plot the percentage of edges that are *k-anonymous* for the corresponding value of k on the x-axis. Each graph plots two selected data sets and compares the *k-anonymity* of the original and anonymized versions of the same graphs. Different graphs correspond to different data sets, different values of μ, and different algorithms. In these experiments, we select the *Flickr* and *Orkut* graphs as representatives. Similarly, μ values of 1 and 3 are representatives chosen to show the variance of the anonymity levels as the indistinguishability threshold increases. In our experiments, the edge weights were in the range of 1 to 100, so $\mu = 1$ corresponds to 1% of the total range of edge weights. As is evident from the Figures 8 and 9, our anonymization models considerably improve *k-anonymity* of the anonymized graphs when compared to the original graphs. The improvement is even more significant for larger values of k and smaller values of μ, which demonstrates the improved anonymity of edges in the anonymized graph. Therefore, in the anonymized graphs, individual edge-weights are even less distinguishable. Note that this level of *k-anonymity* is provided by the model at no additional cost. We remark that the *k-anonymity* can be further improved by adding constraints and setting bounds on the variables that ensure that the anonymized weights are even closer to each other. Additionally, note that the *k-anonymity* of the edges is better for the *Flickr* data set due to the higher average out-degree of the vertices which allows for more room for hiding in the *edge neighborhood*.

Figure 10 plots the *Spearman rank correlation coefficient* of the models for *single source shortest paths tree* and *all pairs shortest paths* problem. Since the value of the coefficient ρ forms a continuum in the range $-1.0 \leq \rho \leq 1.0$, for ease of presentation, we maintain a equi-width histogram of the coefficient values. Along the x-axis, we plot the bucket boundaries of the histogram, and along the y-axis we plot the percentage of vertices that have the value of ρ in the range corresponding to the bucket. The two graphs plot four data sets and Figure 10(a) plots the results for the *single source shortest paths* tree while Figure 10(b) plots the results for *all pairs shortest paths* problem. Figures 10(a) and 10(b) demonstrate the excellent scrambling of the order of the edge weights. Note that $\rho = 0$ corresponds to no correlation of ordering, and the closer it is to 0, the harder it is for an adversary to determine the original order with high confidence. Our experiments show that for all data sets, more than 75% of vertices have $-0.3 \leq \rho \leq 0.3$, and about 90% of the vertices have $-0.5 \leq \rho \leq 0.5$. Additionally, note that higher the average out degree (refer to Table 3 for the average



(a) Complexity of the model. (b) Time to build the model.

Figure 7: Performance of the models for the *all pairs problem*.

Data Sets	Quadratic Model					Optimized Model				
	Number Inequalities				Time	Number of Inequalities			Time	Unconstrained Edges
	Cat I	Cat II	Cat III	Total	Taken (s)	Merge	Trees	Total	Taken (s)	
Flickr	3,645,749	85,824,651	1,813,512	91,283,912	926.71	10,837,381	1,813,512	12,650,893	172.66	60,166
LJ	2,330,938	25,847,924	2,107,957	30,286,819	320.42	7,588,195	2,107,957	9,696,152	132.31	15,003
Orkut	1,428,809	26,907,339	1,088,890	29,425,038	277.33	4,377,502	1,088,890	5,466,392	72.32	23,018
Youtube	2,762,305	38,902,975	2,756,994	44,422,274	473.945	9,163,912	2,756,994	11,920,906	151.04	22,802

Table 5: Experimental evaluation of all pairs shortest paths problem for the community driven data sets.

degrees of the graphs in the data sets), the lesser is the correlation between the original and the anonymized orders.

Therefore, these experiments demonstrate the robustness of the privacy models, and show how hard it is for an adversary to determine the original edge weight, to uniquely identify edge weights, or to determine the original ordering of the weights, thereby effectively preserving the sensitivity of the weights.

7. RELATED WORK

The need to protect the privacy of social entities involved in social networks has given rise to active research in anonymization techniques for social network graphs. This interest has been primarily driven by the findings of Backstrom et al. [3] and Korolova et al. [15]. Backstrom et al. [3] described a technique based on the structural properties of graphs such as isomorphism and automorphism to re-identify vertices in the anonymized graph. Their technique was based on implanting unique structures in the graph which can be re-identified in the anonymized graph with very high probability. On the other hand, Korolova et al. [15] devised an attack where a node can be re-identified based in part on background information regarding the neighborhood. As a result, a lot of research has focused on *node identity anonymization* and *structural anonymization*. A comprehensive survey is provided in [18].

A class of proposals, by Hay et al. [12], Zhou et al. [31], and Liu et al. [19], suggest different methods for anonymization that are based on the addition and/or deletion of edges in the graph for altering the structure of the graph and the prevention of re-identification in the anonymized graph. On the other hand, Cormode et al. [6] suggest a technique for the anonymization of bipartite graphs based on safe groupings, Ying et al. [29] propose a randomization based spectrum preserving approach which effectively preserves the properties of the eigenvalues of the network, while anonymizing the edges, and Campan et al. [5] suggest a clustering based approach for node anonymization. Along different lines, Zheleva et al. [30] formulate the problem of edge re-identification in an unweighted graph, where the edge labels are sensitive information and need to be anonymized.

A large portion of existing work considers unweighted graphs for node identity and structural anonymization. But as reflected by recent work [28, 17], the weighted social network model is gradually gaining importance, and edge weight anonymization is gaining significance. Liu et al. [20] suggest a probabilistic technique for anonymizing edge weights by perturbing the actual edge weights by a small σ obtained from a probability distribution. The goal here is to keep the total cost of the shortest path close to the cost of the path in the original graph. However with this approach, the anonymized weights are close to the original edge weights, and hence may reveal sensitive information about the original values. Our proposed technique aims at preserving general linear properties of the graph. For the shortest paths, our goal is to preserve the paths rather than the values and for most applications, the ability to reconstruct the actual path is more important than maintaining approximate values. In addition, if necessary, our model can approximately preserve the cost of the shortest paths as well by adding constraints of the form $f(u, \ldots, v) = D[u, v] \pm \epsilon$. Note that since the edge weights are only perturbed by a small value, the technique of [20] can neither significantly improve *k-anonymity*, nor can it scramble the ordering of edge weights.

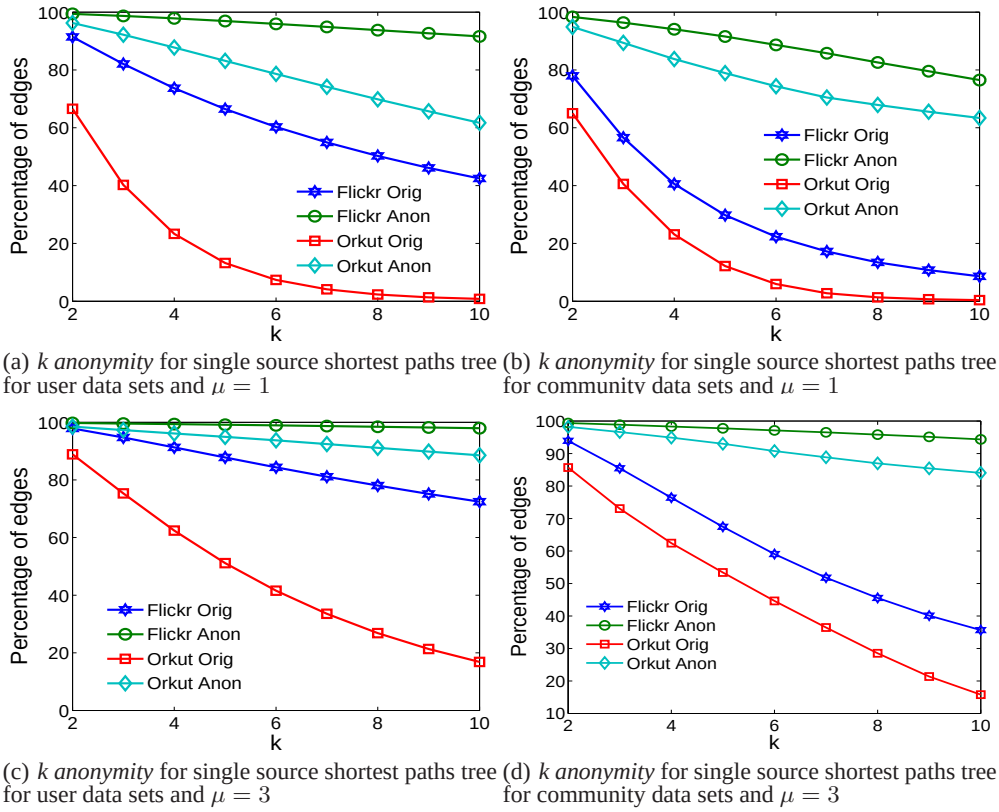
8. CONCLUSION

The huge amount of information contained in weighted social networks spawned a renewed interest in information extraction and data mining. Due to privacy issues, this has resulted in the need for effective techniques for the anonymization of social network graphs. In this paper, we propose a technique for edge anonymization of a weighted social network graph such that linear properties of the edge-weights are preserved across anonymization, while revealing little information about the actual magnitude of the edge weights. Since many important algorithms of interest are functions that depend upon linear properties of edge-weights, our approach provides an anonymization methodology that preserves substructures of interest in the given social graph.

In this paper, we provide a solution for effective anonymization of weighted social network graphs. We first present an abstract

Data Sets	100 vertices		200 vertices		Optimized All Pairs		Percent Reduction	
	Number of Inequalities	Unconstrained Edges	Number of Inequalities	Unconstrained Edges	No. of inequalities	Unconstrained edges	100 vertices	200 vertices
Flickr-comm	513,414	64,186	1,177,428	63,433	12,650,893	60,166	95.94	90.69
LJ-comm	314,107	18,339	732,212	17,819	9,696,152	15,003	96.76	92.45
Orkut-comm	253,002	25,709	562,005	25,429	5,466,392	23,018	95.37	89.72
Youtube-comm	374,516	25,596	835,831	25,162	11,920,906	22,802	96.86	92.99

Table 6: Experimental evaluation of all pairs shortest paths between a subset of vertices for the community driven data sets.



(a) *k anonymity* for single source shortest paths tree for user data sets and $\mu = 1$

(b) *k anonymity* for single source shortest paths tree for community data sets and $\mu = 1$

(c) *k anonymity* for single source shortest paths tree for user data sets and $\mu = 3$

(d) *k anonymity* for single source shortest paths tree for community data sets and $\mu = 3$

Figure 8: Evaluating *k-anonymity* for *single source shortest paths* tree model.

model that can effectively preserve any linear property of edge weights. As a proof of concept, we consider the *shortest paths problem* and show how off-the-shelf linear programming libraries can be used to effectively anonymize the graphs. We also prove the correctness of the proposed models, analyze their complexity, and experimentally validate our claims using real social network data. Our experiments demonstrates the effectiveness of the anonymization achieved by the proposed techniques by using two well known measures: *k-anonymity* and *Spearman rank correlation coefficient*.

The proposed abstract technique can model any linear property of the edge weights. As examples, we considered the two most common applications, the *single source shortest paths tree* (which subsumes the *k nearest neighbors*), and *all pairs shortest paths problem*. In future work, we would like to explore the extensions of the abstract models for other applications such as graph clustering, graph summarization, etc., which also rely on linear combinations of edge weights.

9. ACKNOWLEDGEMENTS

The authors would like to thank Divyakant Agrawal, Pamela Bhattacharya, and Sayan Ranu for their insightful comments on the earlier versions of the paper which has helped in improving this paper. The authors would also like to thanks Alan Mislove for providing the data sets used for the experiments. This work is partially supported by NSF Grant IIS-0744539.

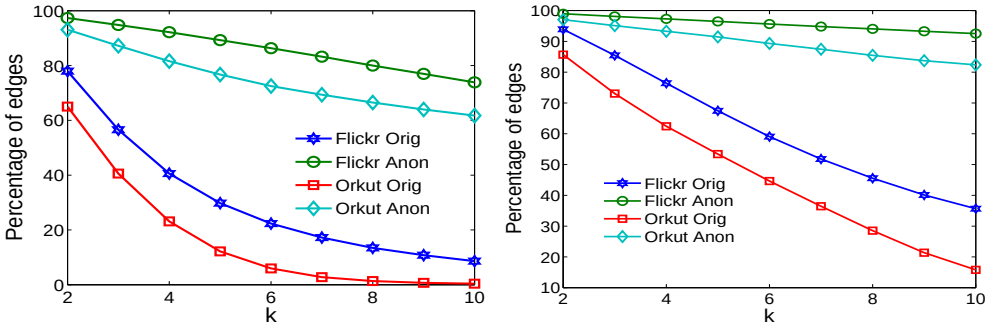
10. REFERENCES

[1] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *WWW*, pages 835–844, New York, NY, USA, 2007. ACM.

[2] S. Amer-Yahia, L. V. S. Lakshmanan, and C. Yu. Socialscope: Enabling information discovery on social content sites. In *CIDR*, 2009.

[3] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore Art Thou R3579X?: Anonymized Social Networks, Hidden Patterns, and Structural Steganography. In *WWW*, pages 181–190, 2007.

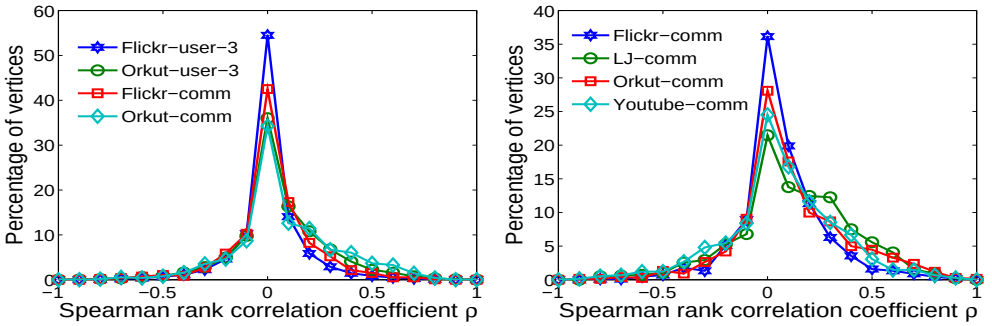
[4] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, New York, NY, USA, 2006. ACM.



(a) *k anonymity* for incremental all pairs shortest paths for community data sets and $\mu = 1$

(b) *k anonymity* for incremental all pairs shortest paths for community data sets and $\mu = 3$

Figure 9: Evaluating *k-anonymity* for the *all-pairs shortest paths* tree models. For incremental all pairs, algorithm is terminated after 50 vertices.



(a) Spearman rank correlation coefficient for single source shortest paths tree.

(b) Spearman rank correlation coefficient for incremental all pairs shortest paths.

Figure 10: Evaluating *Spearman rank correlation coefficient* for the models. For incremental all pairs, algorithm is terminated after 50 vertices.

[5] A. Campan and T. M. Truta. A Clustering Approach for Data and Structural Anonymity in Social Networks. In *PinKDD*, pages 1–10, 2008.

[6] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *Proc. VLDB Endow.*, 1(1):833–844, 2008.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

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

[10] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proc Natl Acad Sci U S A*, 99(12):7821–7826, June 2002.

[11] M. Granovetter. Threshold models of collective behavior. *The American Journal of Sociology*, 83(6):1420–1443, 1978.

[12] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. *Proc. VLDB Endow.*, 1(1):102–114, 2008.

[13] S. Hill, F. Provost, and C. Volinsky. Network-based marketing: Identifying likely adopters via consumer networks. *Statistical Science*, 22(2):256–275, 2006.

[14] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[15] A. Korolova, R. Motwani, S. Nabar, and Y. Xu. Link Privacy in Social Networks. In *ICDE*, pages 1355–1357, 2008.

[16] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, February 1956.

[17] J. M. Kumpula, J. P. Onnela, J. Saramaki, K. Kaski, and J. Kertesz. Emergence of communities in weighted networks. *Physical Review Letters*, 99:228701–1–228701–4, 2007.

[18] K. Liu, K. Das, T. Grandison, and H. Kargupta. *Privacy-Preserving Data Analysis on Graphs and Social Networks*, chapter 21, pages 419–437. CRC Press, December 2008.

[19] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, pages 93–106, 2008.

[20] L. Liu, J. Wang, J. Liu, and J. Zhang. Privacy preservation in social networks with sensitive edge weights. In *SDM*, pages 954–965, 2009.

[21] LPSolve 5.5. `http://lpsolve.sourceforge.net/5.5/`.

[22] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, pages 29–42, 2007.

[23] V. Rastogi, D. Suciu, and S. Hong. The boundary between privacy and utility in data publishing. In *VLDB*, pages 531–542, 2007.

[24] M. K. Sparrow. The application of network analysis to

criminal intelligence: An assessment of the prospects. *Social Networks*, 13:251–274, 1991.

[25] C. Spearman. The proof and measurement of association between two things. *American J. of Psychology*, 15:72–101, February 1904.

[26] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. J. Uncert. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[27] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *KDD*, pages 990–998, New York, NY, USA, 2008. ACM.

[28] R. Toivonen, J. M. Kumpula, J. Saramäki, J.-P. Onnela, J. Kertész, and K. Kaski. The role of edge weights in social networks: modelling structure and dynamics. *Noise and Stochastics in Complex Systems and Finance*, 6601(1):B1–B8, 2007.

[29] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, pages 739–750, 2008.

[30] E. Zheleva and L. Getoor. Preserving the Privacy of Sensitive Relationships in Graph Data. In *PinKDD*, pages 153–171, 2007.

[31] B. Zhou and J. Pei. Preserving Privacy in Social Networks Against Neighborhood Attacks. In *ICDE*, pages 506–515, 2008.