3-2014

L-opacity: Linkage-aware graph anonymization

Sadegh NOBARI
Singapore Management University

Panagiotis KARRAS
Rutgers University

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

Stephane BRESSAN
National University of Singapore

$\mathcal{L}$-opacity: Linkage-Aware Graph Anonymization

Sadegh Nobari[¶] Panagiotis Karras[§] HweeHwa Pang[¶] Stéphane Bressan[†]

[¶]SMU School of Information Systems Singapore

[§]Rutgers Business School USA

[†]NUS School of Computing Singapore

ABSTRACT

The wealth of information contained in online social networks has created a demand for the publication of such data as graphs. Yet, publication, even after identities have been removed, poses a privacy threat. Past research has suggested ways to publish graph data in a way that prevents the re-identification of nodes. However, even when identities are effectively hidden, an adversary may still be able to infer linkage between individuals with sufficiently high confidence. In this paper, we focus on the privacy threat arising from such *link disclosure*. We suggest $\mathcal{L}$-opacity, a sufficiently strong privacy model that aims to control an adversary's confidence on *short* multi-edge linkages among nodes. We propose an algorithm with two variant heuristics, featuring a sophisticated look-ahead mechanism, which achieves the desired privacy guarantee after a few graph modifications. We empirically evaluate the performance of our algorithm, measuring the alteration inflicted on graphs and various utility metrics quantifying spectral and structural graph properties, while we also compare them to a recently proposed, albeit limited in generality of scope, alternative. Thereby, we demonstrate that our algorithms are more general, effective, and efficient than the competing technique, while our heuristic that preserves the number of edges in the graph constant fares better overall than one that reduces it.

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: [Graph algorithms, Path problems]; K.4.1 [**Computers and Society**]: [Privacy]

General Terms

Algorithms, Experimentation, Theory

1. INTRODUCTION

Data sets storing information about persons and their relationships are abundant. Online social networks, e-mail exchange records, collaboration networks, are some examples. Such data can be modeled in graph form, which, when published, can provide valuable information in domains such as marketing, sociology, and fraud detection. Still, the publication of such graph data entails privacy threats for the individuals involved. Research on how to mitigate these privacy threats while still enabling the publication of useful information about the network is now taking shape [2, 30, 14, 31, 12, 5, 16, 27, 29, 7, 4, 3, 32, 25, 13, 6, 23, 26, 22].

A privacy threat involves the leakage of *sensitive information*. This information may involve the identity of a node (i.e., person) in the network, in which case we talk of *identity disclosure*. The bulk of previous research has focused on the privacy threat arising from such re-identification of the node representing a certain individual in the network [31, 12, 5, 16, 32, 25, 13]. The common theme in such works is the idea that each node should be rendered, by some notion, indistinguishable from $k-1$ other nodes in the network; this idea is inspired from the precept of k-anonymity, a principle suggested for the anonymization of relational data [21].

Nevertheless, a privacy threat may also involve the information about connections between individuals in a network. In this case, we talk about *linkage disclosure*. Unfortunately, the protection against identity disclosure does not imply protection against linkage disclosure as well.

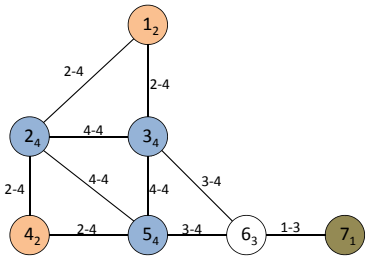


Figure 1: An Example Graph

For illustration, the graph in Figure 1 represents a social network, where each vertex stands for a person and each edge denotes a friendship connection. Vertices are numbered, while subscripts on their number labels indicate their degrees (e.g., vertex 2 has degree 4, therefore it is inscribed with the label 2_4). Edges are labeled by the degrees of the vertices they connect, in ascending order (e.g., the edge between vertex 3_4 and vertex 6_3 is labeled as a $3-4$ edge).

We assume an adversary who attempts to re-identify the vertex corresponding to an individual in the graph via its degree; thus, a node's degree forms the adversary's *background knowledge*. Such an adversary may also try to infer the length of the connection between two particular individuals. Furthermore, assume that an adversary knows that Charles is in the network with four friends, Agatha has four friends, Timothy three, Cynthia two, and Oliver one. Then, looking at Figure 1, the adversary will infer that each of Charles and Agatha are to be found among the vertices 2_4, 3_4, and 5_4, which form a triangle. Thus, regardless of the particular

10.5441/002/edbt.2014.52

valid assignment, it has to be the case that Charles and Agatha are friends. Furthermore, given the graph structure, it must be the case that Timothy and Cynthia are connected by a path of length 2, hence have one friend in common (Timothy is identified as vertex 6_3, and Cynthia as either vertex 1_2 or 4_2). One can also deduce that Oliver is Cynthia's friend, as he is identified with vertex 7_1.

From the preceding discussion it follows that a network preventing identity disclosure (by rendering vertices indistinguishable as in [31, 12, 5, 16, 32, 25, 13]) may still allow the disclosure of a *linkage* between two vertices of interest. In other words, an adversary may infer that a certain linkage between two entities of interest exists in the graph, *regardless* of which among many (e.g., k) indistinguishable nodes represents each of these two entities; in effect, sensitive information is leaked. Zhang and Zhang [29] were the first to make this cardinal observation, and provided a solution for the ensuing *edge anonymity* problem; however, their analysis does not move beyond single-edge connections, while the proposed solutions lack computational efficiency. Cheng et al. [6] reiterated the same observation (namely, protection against identity disclosure does not protect against linkage disclosure), and looked at the general problem of preventing the disclosure of a multi-edge connection; to overcome the problem, they suggest a method that divides the graph into k *disjoint* subgraphs, and renders those *isomorphic* to each other; thus, the full network becomes k*-isomorphic*. While this method achieves the objective of thwarting attempts to infer a multi-edge linkage between two entities of interest, it severely alters the nature of the published network from a connected graph to an assortment of k identical disjoint graphs; thus, by k-isomorphism, we do not publish an anonymized version of the *whole* network, but only $\frac{1}{k}$ thereof. Other approaches have also specifically studied ways to conceal linkages or interactions among entities [30, 27, 4, 6]; however, such approaches are either based on clustering nodes into super-nodes [30], and/or deal with a bipartite interaction graph [4], obliterating the structural information in the network, or follow a randomization approach without clear privacy guarantees [27].

In this paper we revisit the *linkage anonymization* problem. Our approach is positioned between the two extremes found in [29] and [6]. In contrast to [29], we do not consider only *single-edge* links to be important; an adversary who can confidently infer that two individuals in a social network are connected by a multi-edge path still infers valuable sensitive information about them. Still, contrary to [6], we do not attempt to totally extinguish the potential for the inference of an *arbitrarily long* linkage path.

Real-world networks are *connected*; any two individuals in them are bound to be linked by a sufficiently long path. The length of this path is usually rather small, not exceeding *six* steps. Milgram's small world experiment [17] suggested that social networks of people in the United States are characterized by short node-to-node distances, of approximately three links, on average, without considering global linkages; Watts [24] recreated Milgram's experiment on the Internet and found that the average number of intermediaries via which an e-mail message can be delivered to a target was around six; Leskovec and Horvitz [15] found the average path length among users of an instant-messaging system to be 6.6. Goel et al. [11] tested the extent to which pairs of individuals in a large social network can actually *find* the shortest paths connecting them; they introduced a rigorous way of estimating true chain (i.e., search distance) lengths in a messaging network, and found that roughly half of all chains can be completed in 6-7 steps. Most recently, Backstrom et al. [1] reported that the average distance in the entire Facebook network of active users was 4.74.

In view of this connectedness of real world networks, we deduce that no privacy is compromised by revealing the *existence* of a path among two entities in a network; thus, setting a target of thwarting the inference of any linkage *whatsoever*, as in [6], not only irretrievably alters the nature of the network, but also sets an unnecessarily high privacy objective. Instead, we propose that the focus of a privacy concern should be on averting the disclosure of the existence of a *short* path, as opposed to the existence of *any* path. Following this reasoning, we define $\mathcal{L}$-opacity, a privacy principle based on the notion that an adversary possessing certain structural background knowledge should not be able to infer that the distance between two entities in a network is equal to or less than a chosen threshold $\mathcal{L}$ with confidence higher than a threshold θ. Our aim is to prevent such confident inferences by incurring a *minimal* amount of modification on the network.

2. RELATED WORK

The discussion on graph anonymization was initiated by Backstrom et al. [2], who pointed out that an adversary can infer the identity of nodes in an de-annotated graph by solving a restricted graph isomorphism problem. However, [2] proposed no technique for publishing a graph in a privacy-preserving manner.

A particular graph anonymization technique was first proposed by Zheleva and Getoor [30]. This technique assumes that only a subset of the graph's edges are sensitive and attempts to conceal them via clustering nodes, randomly removing non-sensitive edges, and reporting only the number of edges between groups. Korolova et al. [14] consider the problem posed by an adversary breaking into user accounts of an online social network and trying to re-assemble the network graph from a set of local neighborhoods.

Two recent works address problems of preventing structural re-identification of a node by adversaries who know a target's local neighborhood. Zhou and Pei [31] study the problem on *node-labeled* graphs; they propose the notion of k*-neighborhood anonymity* for such graphs, achieved by generalizing node labels and inserting edges so that each node's (one-step) local neighborhood is rendered isomorphic to at least $k-1$ others. Hay et al. [12] address the same problem on unlabeled graphs. They propose the notion of k*-candidate anonymity*, which requires that at least k nodes match a neighborhood-structure query on the graph, aiming to resist attacks from adversaries possessing knowledge of an individual's neighborhood structure. Still, they do not propose algorithms aiming to guarantee the privacy principle they introduce; as an anonymization method, they only propose grouping nodes into partitions and publishing the number of vertices and edge density in each partition, as well as the edge density across partitions. Unfortunately, this method fails to preserve much of the graph's structural information. In similar spirit, Campan and Truta [5] propose a method that divides vertices (labeled by attributes) into clusters of at least k entities, and collapses each cluster to a single vertex.

Motivated by [2] and [12], Liu and Terzi [16] were the first to suggest an anonymization technique tailored for simple graphs with unlabeled nodes and uniform edges. Under the assumption that an adversary possesses knowledge of a node's degree, their anonymization method first transforms a graph's sorted degree sequence into a k-anonymous one, in which each degree value appears at least k times, using the algorithm in [10]; then, guided by the k-anonymous degree sequence, it inserts edges into the graph to render it k*-degree anonymous*, i.e. ensure that any degree value is shared by at least k nodes.

Ying and Wu [27] show that the topological similarity of nodes can be used to recover original sensitive links from a randomized graph. As discussed, Zhang and Zhang [29] observed that even if a graph preserves vertex anonymity, it may not preserve edge anonymity; they suggest the privacy notion of τ*-confidence*, which

limits an adversary's confidence that a single edge exists between the vertices corresponding to two individuals, and suggest heuristics to achieve this objective by edge swaps and removals.

Cormode et al. [7] propose a family of *safe* (i.e., attack-proof) anonymizations for bipartite graph data that fully preserve the (unlabeled) graph structure, while anonymizing the mapping from entities to nodes of the graph. This approach is taken further by Bhagat et al. [4], who pay attention to the rich interaction information in a social network; they suggest anonymization methods based on carefully grouping entities of the network's bipartite interaction graph into classes, while masking the mapping between entities and the nodes that represent them in the graph, in a way that fulfills a *safety condition*. Recently, Bhagat et al. [3] have also proposed methods to anonymize a dynamic social network while new nodes and edges are inserted, leveraging link prediction algorithms to model the network's evolution.

Both Zou et al. [32] and Wu et al. [25] suggest methods to transform a data graph so that each node in the resulting graph is structurally indistinguishable from $k-1$ other nodes, thus achieving protection against identity disclosure. This property is called k-*automorphism* in [32] and k-*symmetry* in [25]. The anonymization algorithm in [32] uses graph alignment and edge insertion as its main operation, while the one in [25] is based on making duplicate copies of vertices in the network. He et al. [13] suggest an akin anonymization method, which first partitions a graph into local structures of size d, then divides these structures into groups of k structures each, and locally transforms the structures within each group so as to render them isomorphic to each other; the privacy this method achieves is named k^d *graph anonymity*.

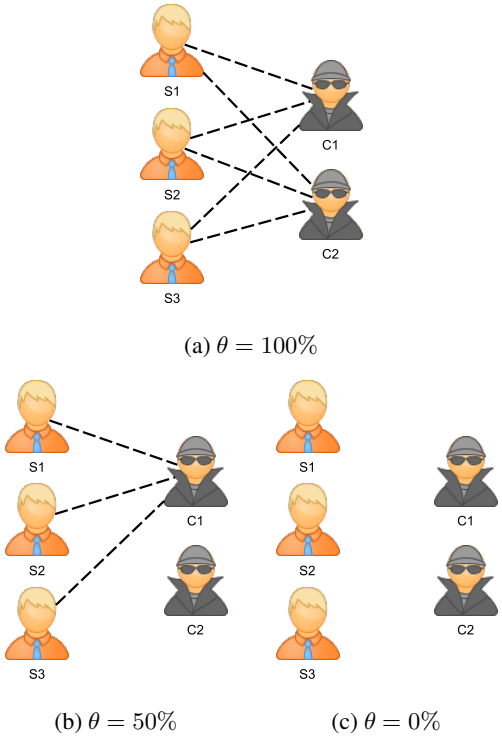
Cheng et al. [6] reiterated the observation that protection against identity disclosure, as provided in [32, 25, 13], does not guarantee protection against the disclosure of *sensitive linkages*. To overcome this problem, they suggest a method that adopts the strategy of rendering sets of nodes structurally indistinguishable from each other, yet also thwarts attempts to infer a linkage between nodes. This aim is achieved by dividing the graph into k *disjoint* subgraphs, rendering it k-*isomorphic*. Unfortunately, even while this method achieves the objective of protection against link disclosure, it severely alters the nature of the published network from a connected graph to an assortment of k identical disjoint graphs.

Recently, Yuan et al. [28] have examined the privacy protection problem with a new twist, in which they consider that most of the nodes in the network face no privacy threats related to structural knowledge at all, while only a few nodes have such needs, arising from an adversary's knowledge of degrees and edge labels. The problem of linkage disclosure is mentioned in [28], albeit the suggested privacy methods do not provide protection against it.

3. MOTIVATING EXAMPLES

In the following we bring some examples and arguments that further justify the privacy model we propose, in particular our use of a path length threshold $\mathcal{L}$ and a confidence threshold θ, as well as our model of adversary knowledge.

First, in the DBLP coauthorship graph, *HweeHwa Pang* is connected to *Elisa Bertino*. A path between them is: *H. Pang* $\rightarrow$ *S. Nobari* $\rightarrow$ *A. Ailamaki* $\rightarrow$ *P. Karras* $\rightarrow$ *S. Bressan* $\rightarrow$ *E. Bertino*. If this path of length 5 were the shortest path from *HweeHwa* to *Elisa*, we propose that it would not be a very serious privacy breach if an adversary inferred its length; two authors in the same community are bound to be connected by such a path. However, there is another path: *H. Pang* $\rightarrow$ *K.-L. Tan* $\rightarrow$ *E. Bertino*. This path shows a more intimate connection between those authors. Thus, pragmatically, starting out from the observation of the small-world phenomenon, we conclude that it is more relevant to prevent the inference of *short-path* connections than of *all* connections.



(a) $\theta = 100\%$

(b) $\theta = 50\%$ (c) $\theta = 0\%$

Figure 2: Example illustrating the θ parameter

Second, assume a social network service intends to publish data about individuals and their networks. The data in question shows a close connection between Albert and an old high school friend, Bruce. Since high school, Bruce's life has been quite different from Albert's; Bruce has recently been convicted for drug trafficking. Under these circumstances, Albert, who has not been in contact with Bruce for years, may reasonably prefer that his connection to Bruce be not revealed to the public, especially not while he is making arrangements for his forthcoming wedding. Thus, Albert has a privacy concern about the revelation of a *short* path connecting him to Bruce. A *longer* path connecting Albert to Bruce, even if confidently inferred, would not cause a major privacy concern. In similar fashion, it is a fundamental assumption of ours that a privacy threat arises out of inferring a short path and does not arise out of inferring a long path. Our work rests on this assumption.

Third, assume a graph in which a node represents a person and a link between two persons shows they are acquainted. We suggest that this graph may be published with names removed, while an adversary may have background knowledge about the number of acquaintances each person has in the network. That adversary may then be able to associate a *criminal* to two nodes (C_1 and C_2), as illustrated in Figure 2. The same adversary may associate a target individual to three other nodes (S_1,S_2 and S_3). If all of S_1, S_2, and S_3 are found to be connected by a path of length $\leq \mathcal{L}$ to both C_1 and C_2, then the probability that the target is connected to the criminal is 100%, i.e. $\theta = 100\%$ (Figure 2a). If all of S_1, S_2, and S_3 are found to be connected to only C_1, then the effective probability that the target is connected to the criminal is 50%, i.e. $\theta = 50\%$ (Figure 2b). Last, if none of S_1, S_2, and S_3 is found to be connected to any of C_1 and C_2, then θ is 0% (Figure 2c). The θ threshold we employ bounds the adversary's confidence in an inferred linkage as in this example.

Last, a few words are due about our adversary model. We assume

an adversary who possesses knowledge of target nodes' degrees. This knowledge exemplifies a kind of structural information the adversary can possess so as to identify nodes; we use this kind of knowledge as a first proposal, noting that research on preventing identity disclosure started out with such background knowledge before expanding into more arcane cases of structural knowledge [16]. We envisage that future work can likewise expand into other types of structural knowledge, while our privacy model definition covers any way of classifying nodes into types.

4. PROBLEM DEFINITION

In this section we formally define the privacy protection *problem* we set out to solve in this paper, provide some results on its *hardness*, and clarify our data *publication model*.

We assume that a social network is modeled by a simple graph (i.e., an undirected, unweighted graph, without self-loops or multiple edges). Let $G(V, E)$ be such a simple graph, where V is the set of nodes and E the set of edges in G. The degree d_v of a vertex $v \in V$ is the number of edges to which v is adjacent. For a pair of vertices, v_i, v_j, the *geodesic distance* (GD) between them is the length ℓ_{ij} of a shortest path connecting them.

We consider that each vertex is characterized by certain properties, which may render it identifiable in the published graph. For the sake of generality, our model is agnostic about what these properties may be. For our purposes, it is sufficient to assert that one can identify *pairs* of distinct vertices belonging to certain *types*. Pair types are meant to be of interest to the data vendor and/or considered vulnerable for identification by an adversary. We outline the properties of a node-pair type $\mathcal{T}$ as follows:

DEFINITION 1. *Given a simple graph G, a collection of vertex-pair* types *$\mathcal{C}$ is defined. For each vertex-pair type $\mathcal{T} \in \mathcal{C}$, a distinct vertex-pair (v_i, v_j), $v_i, v_j \in V$, with distance ℓ_{ij}, belongs to $\mathcal{T}$. Then, we write $(v_i, v_j) \in \mathcal{T}$; for brevity, we also write $\ell_{ij} \in \mathcal{T}$ to denote that there exists a vertex-pair with distance ℓ_{ij} in type $\mathcal{T}$. Each vertex can belong to one or more vertex-pairs, while each vertex-pair belongs to* at most *one type. It is not required that every definable vertex-pair (v, w) belongs to a type; some vertex-pairs may be indifferent to us, belonging to no type at all.*

In the following, we use the notation $\mathcal{T}$ to refer both to a vertex-pair *type* and to the *set* of vertex-pairs of that type. It follows that the cardinality of the *set* $\mathcal{T}$ is equal to the number of distinct *vertex-pairs* (v, w) having *type* $\mathcal{T}$. We define the *$\mathcal{L}$-opacity* of type $\mathcal{T}$ as follows.

DEFINITION 2. *Given a simple graph G and a vertex-pair type $\mathcal{T} \in \mathcal{C}$, the* $\mathcal{L}$-opacity *of $\mathcal{T}$, $\mathcal{LO}_G(\mathcal{T})$, is the* ratio *of the number of vertex-pairs in $\mathcal{T}$ with distance at most $\mathcal{L}$, $|\{\ell_{ij} \in \mathcal{T} | \ell_{ij} \leq \mathcal{L}\}|$, to the number of* all *vertex-pairs in $\mathcal{T}$, including pairs of mutually unreachable vertices:*

$$\mathcal{LO}_G(\mathcal{T}) = \frac{|\{\ell_{ij} \in \mathcal{T} | \ell_{ij} \leq \mathcal{L}\}|}{|\mathcal{T}|}$$

We wish to render inferences involving linkage disclosure harder and less confident. That is, we would like a graph to obey the following property.

DEFINITION 3. *Given a graph $G(V, E)$ and a collection $\mathcal{C}$ of types of interest defined on G, G satisfies* **$\mathcal{L}$-opacity** *(is said to be* $\mathcal{L}$-opaque*) with respect to a threshold θ, if and only if, for every vertex-pair type $\mathcal{T} \in \mathcal{C}$, $\mathcal{LO}_G(\mathcal{T})$, does not exceed a threshold θ, $0 \leq \theta \leq 1$, that is:*

$$\mathcal{LO}(G) = \max_{\mathcal{T} \in \mathcal{C}} \{\mathcal{LO}_G(\mathcal{T})\} < \theta$$

Again, for the sake of simplicity, when the value of $\mathcal{L}$ is clear from the context, we refer to the $\mathcal{LO}(G)$ value as the *opacity* of G. Given an $\mathcal{L}$-opaque form $\hat{G}$ of a graph G, an adversary *cannot* infer that a vertex-pair of a predefined type $\mathcal{T} \in \mathcal{C}$ of interest have distance at most $\mathcal{L}$ with certainty more than θ. Our aim is to bring the published graph to such a form by inducing a minimum amount of distortion to it. The basic distortion operations we employ are *edge removal* and *edge insertion*, transforming the edge set E of the original graph G to the set $\hat{E}$ in the anonymized graph $\hat{G}$. We measure the amount of distortion $\mathcal{D}$ as the graph edit distance between G and $\hat{G}$, i.e. the symmetric difference between the edge sets $|E \Delta \hat{E}|$, normalized over the number of edges of the original graph. In other words the total proportion of the missing and inserted edges over $|E|$:

$$\mathcal{D}(E, \hat{E}) = \frac{|E \cup \hat{E} - E \cap \hat{E}|}{|E|} \quad (1)$$

In effect, we define our *$\mathcal{L}$-opacification* problem as follows:

PROBLEM 1. *Given a graph $G(V, E)$, a collection $\mathcal{C}$ of types of interest defined on G, an integer $\mathcal{L}$ and a threshold θ, transform G to an $\mathcal{L}$-opaque form $\hat{G}(V, \hat{E})$ with respect to θ, so that $\mathcal{D}(E, \hat{E})$ is minimized.*

Eventually, our goal is to select a set of edges $\hat{E}$ that renders $\hat{G}(V, \hat{E})$ $\mathcal{L}$-opaque (i.e., the proportion of vertex-pairs with distance $\mathcal{L}$ or less within every vertex-pair type $\mathcal{T}$ defined therein is at most θ) and minimizes $\mathcal{D}(E, \hat{E})$. This is a combinatorial optimization problem. An exhaustive-search solution would be to try out all possible sets $\hat{E}$, check which ones yield an $\mathcal{L}$-opaque graph $\hat{G}(V, \hat{E})$, and opt for the one that minimizes $\mathcal{D}(E, \hat{E})$; this approach would result to an optimal solution. However, there are $O\left(2^{|V|^2}\right)$ possible sets of edges $\hat{E}$ to try, while each check would require at least an $O\left(|V|^3\right)$ all-pairs-shortest-path computation. Indeed, Theorem 1 shows that this problem is NP-hard.

THEOREM 1. *$\mathcal{L}$-opacification is NP-hard.*

PROOF. We show that we can reduce the NP-hard 3-SAT problem [9] to the $\mathcal{L}$-opacification problem in polynomial time. The 3-SAT problem is a version of the satisfiability problem in which every clause has 3 variables, as follows:

Input: $\{C, B\}$, where $C = \{C_1, C_2, \ldots, C_S\}$ is a collection of clauses, each clause being the disjunction of 3 literals over the finite set of N Boolean variables $B = \{v_1, v_2, \ldots, v_N\}$.

Output: Decides whether there is an assignment of truth values to B that makes every clause of C true.

Given any instance of the 3-SAT problem, we construct an instance of the $\mathcal{L}$-opacification problem as follows. First we construct a graph $G(V, E)$ based on the given 3-SAT problem. For each boolean variable $v \in B$ we insert two edges (v_i, v_j), (v'_i, v'_j) in E. We classify these two edges as belonging to the same type, namely type (A_v, B_v). Then, for each clause C_k in which v participates without negation we create a pair of vertices (A_k, B_k), such that A_k is an one-hop neighbor of v_i and B_k an one-hop neighbor of v_j. Thus, we say that clause C_k is *appended* to edge (v_i, v_j). Besides, we classify each such vertex-pair as belonging to type (A_k, B_k). In effect, we create a vertex-pair of type (A_k, B_k), connected via a path of length 3 passing through edge (v_i, v_j). The same appending occurs for all other variables in clause C_k and any other clause. Likewise, for each clause C_k in which a variable v participates with negation, as $\neg v$, a pair of vertices of type (A_k, B_k) is created and appended to edge (v'_i, v'_j) as above. Having defined vertex-pair types of interest as above, we define the $\mathcal{L}$-opacification problem

for the ensuing graph G with $\mathcal{L} = 3$ and $\theta = 1$. We turn this optimization problem to a decision problem by asking whether it can be solved via the *removal* of no more than N edges.

Notably, for each variable $v \in B$, the opacification of its associated vertex-pair type, (A_v, B_v), requires the removal of at least one of the two edges of that type, hence we need to perform at least N removals. Thus, if the problem is solvable at all, it will be solved by *exactly* N edge removals, i.e. by the removal of *one and only one* edge associated with each variable v, i.e. either edge (v_i, v_j) or the edge (v'_i, v'_j). Besides, given that $\mathcal{L} = 3$, for each clause C_k, the opacification of its associated vertex-pair type, (A_k, B_k), necessitates the removal of at least one of the edges in the paths from a vertex denoted as A_k to one denoted as B_k, namely at least one of the N removed edges should be in such a path.

We consider the action of *edge removal* in $\mathcal{L}$-opacification to represent the action of *truth assignment* in 3-SAT. Then the above requirements translate to the following:

1. Each of the N variables, v, is set to be either true or false, namely *true* if edge (v_i, v_j) is removed, and *false* if edge (v'_i, v'_j) is removed.

2. Each clause C_k must have at least one of its literals set as *true*, namely a literal corresponding to a removed edge in a path from an A_k vertex to a B_k vertex.

In effect, if we can decide whether the $\mathcal{L}$-opacification problem we have devised can be solved with exactly N edge removals, then we can answer the original 3-SAT problem as well. □

For instance, consider the following 3-SAT clauses:

$$(a \vee \neg b \vee c)_1 \wedge (\neg a \vee \neg c \vee d)_2 \wedge (a \vee b \vee \neg d)_3 \wedge$$
$$(a \vee \neg b \vee \neg c)_4 \wedge (\neg b \vee c \vee d)_5 \wedge (\neg a \vee b \vee \neg d)_6$$

The subscript of every clause in this statement indicates the clause number. Figure 3 shows the graph constructed for the corresponding $\mathcal{L}$-opacification problem. In this figure, vertex labels indicate the vertex-pairs to which these vertices belong. For example, the negated variable $\neg a$ appears in clauses C_2 and C_6, hence vertex pairs of type (A_2, B_2) and (A_6, B_6) are appended to edge (a'_i, a'_j).

As we have discussed, our analysis, and hence our hardness result, applies with *any* choice of properties that may be used to define vertex-pair types of interest. However, it has been noted that the *degree* of a vertex *in the original graph* is the most elementary structural information about a vertex in a de-annotated graph that an adversary can use to re-identify that vertex [16]. Thus, in the rest of this paper we choose to focus on the repercussions of using the ***original degree*** as the vertex property we work with. Based on this strategic choice, a pair type $\mathcal{T}$ is associated with a certain pair of degrees, not necessarily distinct, (d_1, d_2) with distinct vertex-pairs, (v, w) belonging to $\mathcal{T}$, where v has degree d_1 and w has degree d_2 in the original graph G. We emphasize that our solution is concerned with degrees of vertices *in the original graph* only, even while such degree may be altered in the published form of the graph. In our *publication model*, the graph is simply published along with the *original degree* information. This publication model preserves the utility emanating from such degree information, even though vertices may appear with different degrees in the anonymized form; at the same time, it does *not* raise any privacy concerns, as our privacy model is already tailored for adversaries having such knowledge. Besides, this publication model eschews the redundant complication of having to consider artificially changing node degrees throughout the operation of our algorithms.

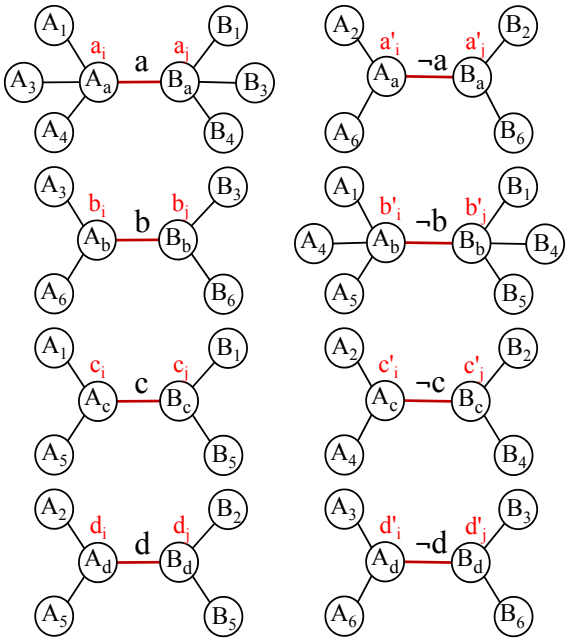


Figure 3: Graph for the given 3-SAT problem in Theorem 1

As the problem is intractable, we now direct our efforts towards devising an efficient solution assisted by heuristics.

5. $\mathcal{L}$-OPACIFICATION ALGORITHM

In a nutshell, our $\mathcal{L}$-opacification algorithm follows a greedy rationale, trying to make a good choice of an edge to remove or insert. In the default mode of operation, it works by making moves involving one edge at a time. Still, its greedy logic is not irretrievable. If there is no beneficial move involving one edge to be made, then it considers a pair of two edges for its next step, and so on up to a threshold. We call this threshold *look-ahead* parameter, *la*. In contrast to [29], with $\mathcal{L} = 1$ and for $la > 1$, we can find a solution for a graph where [29] cannot or find an $\mathcal{L}$-opaque graph with much less amount of distortion, as the *look-ahead* parameter lets our algorithms expand their search space.

We discuss two variants of this algorithm. The former tries to achieve $\mathcal{L}$-opacity by removing edges. The latter attempts to counterbalance every edge removal by a corresponding insertion. Before we enter into details, we describe some fundamental operations involving the computation of probability values.

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	1	1	2	2	2	3
2		0	1	1	1	2	3
3			0	2	1	1	2
4				0	1	2	3
5					0	1	2
6						0	1
7							0

(a) All-pairs shortest paths

degree, i		4 **2**	4 **3**	2 **4**	4 **5**	3 **6**	1 **7**
2	**1**	1	1	0	0	0	0
4	**2**		1	1	1	0	0
4	**3**			0	1	1	0
2	**4**				1	0	0
4	**5**					1	0
3	**6**						1

(b) Boolean values of $\ell_{ij} \leq \mathcal{L}$

Figure 4: Path length matrices

5.1 Basic Operations

In order to decide whether a graph G satisfies $\mathcal{L}$-opacity, we need to compute the number and lengths of geodesic distances of each type $\mathcal{T} \in \mathcal{C}$. To perform this computation, we start out by running Floyd-Warshall's $O(|V|^3)$ all-pairs-shortest-paths algorithm [8] on G, assuming each edge has weight 1. The output of this algorithm on the graph of Figure 1 is the triangular matrix $\mathbf{A}$ of Figure 4a.

Cell $\mathbf{A}_{ij}$, $i \leq j$, contains the geodesic distance (GD) ℓ_{ij} between vertices v_i and v_j. We call this matrix the *distance matrix* of G.

5.1.1 Opacity Value Computation

The information that is interesting for us is whether a GD value ℓ_{ij} in the matrix of Figure 4a satisfies the $\ell_{ij} \leq \mathcal{L}$ predicate for a given $\mathcal{L}$. For the sake of illustration, we present, in Figure 4b, a boolean triangular matrix that shows whether ℓ_{ij} satisfies this predicate for our running example and $\mathcal{L} = 1$. This matrix does not feature elements for $i = j$, since we do not consider paths from a vertex to itself. We represent the matrix concisely, omitting the row for $i = 7$ and the column for $j = 1$. We also annotate to this matrix information about the degree of each vertex v_i in the original graph. Having this degree information and matrix $\mathbf{A}$ of Figure 4a, we can straightforwardly derive, for each pair type $\mathcal{T}$, the number of GDs in $\mathcal{T}$ of length $\ell_{ij} \leq \mathcal{L}$, as well as the number of those whose length is $\ell_{ij} > \mathcal{L}$. The matrices in Figure 5a,b show the results for the running example. In particular, the matrix in Figure 5a, which shows, for each pair type $\mathcal{T}$, the number of GDs in $\mathcal{T}$ of length $\ell_{ij} \leq \mathcal{L}$, is denoted as $\mathbf{L}$. This is the main matrix we need to compute in order to derive a graph's overall opacity value.

$\mathcal{T}$	**1**	**2**	**3**	**4**
1	0	0	1	0
2		0	0	4
3			0	2
4				3

(a) $\ell_{ij} \leq \mathcal{L}$

$\mathcal{T}$	**1**	**2**	**3**	**4**
1	0	2	0	3
2		1	2	2
3			0	1
4				0

(b) $\ell_{ij} > \mathcal{L}$

$\mathcal{T}$	**1**	**2**	**3**	**4**
1	0	0	1	0
2		0	0	$\frac{2}{3}$
3			0	$\frac{2}{3}$
4				1

(c) Opacity Matrix

Figure 5: GD numbers and Opacity Matrix

Having the GD numbers with respect to the length threshold $\mathcal{L}$ calculated above, we can easily derive an *opacity matrix*, i.e., the matrix of $\mathcal{LO}_G(\mathcal{T})$ values for each $\mathcal{T} \in \mathcal{C}$. Figure 5c shows the result. For example, there are three GDs of type $\mathcal{P}_{\{3,4\}}$ (i.e., paths between a vertex of degree 3 and one of degree 4), namely the geodesic distances between vertex v_6, on the one hand, and vertices v_2, v_3, and v_5, on the other hand. Out of these three GDs, two (namely $\ell_{3,6}$ and $\ell_{5,6}$) satisfy the $\ell_{ij} \leq \mathcal{L}$ predicate (see the matrix of Figure 5a), while one (namely $\ell_{2,6}$) does not (see the matrix Figure 5b). Thus, the $\mathcal{L}$-opacity of $\mathcal{P}_{\{3,4\}}$ in G is $\mathcal{LO}_G = \frac{2}{3}$, as the opacity matrix of Figure 5c shows.

Eventually, we can calculate the maximum $\mathcal{L}$-opacity among all $\mathcal{T} \in \mathcal{C}$, namely $\max_{\mathcal{T} \in \mathcal{C}} \{\mathcal{LO}_G(\mathcal{T})\}$. In this case, the value is 1, i.e., G satisfies $\mathcal{L}$-opacity only with respect to $\theta = 1$. Algorithm 1 shows a pseudo-code for this computation. In this pseudo-code, d_k denotes the degree of vertex k in the original graph and $NV(d)$ denotes the number of vertices with degree d.

Algorithm 1: $\max \mathcal{LO}$ Algorithm

Input: $G(V, E)$; $D = [d_0, \ldots, d_{|V|-1}]$; $\mathcal{L}$ parameter
Output: $\max_{\mathcal{T} \in \mathcal{C}} \{\mathcal{LO}_G(\mathcal{T})\}$

```
max LO = 0; L = 0;
Calculate distance matrix of G, A;
foreach ℓ_ij ∈ A do
    g = min{d_i, d_j}; h = max{d_i, d_j};
    if ℓ_ij ≤ L then
        L_gh = L_gh + 1;
foreach P_{g,h} ∈ L do
    if g = h then
        LO_G(P_{g,h}) = (2 × L_gh) / (NV(g) × (NV(g) − 1));
    else
        LO_G(P_{g,h}) = L_gh / (NV(g) × NV(h));
    max LO = max{max LO, LO_G(P_{g,h})};
Return max LO;
```

5.1.2 Distance Matrix Computation

As we have seen, a basic operation, which we will have to perform repeatedly in our heuristics, is the calculation of the distance matrix of a graph G, for which we can employ Floyd-Warshall's all-pairs-shortest-paths algorithm for an undirected graph (i.e., a triangular adjacency matrix). As our problem entails the distance calculation between all pairs of vertices [18], techniques developed for the point-to-point shortest path problem and its approximate variant [19] are not applicable to it.

The Floyd-Warshall algorithm starts out with the adjacency matrix of G, and leads to the respective distance matrix by allowing the paths under consideration (between i and j, handled by the two inner loops) to use one more intermediate vertex (k, handled by the outer loop) at each iteration. Since the distance matrix is triangular, $\mathbf{A}^k_{ij}$ is interpreted as $\mathbf{A}^k_{ji}$ when $j < i$. We do not indicate this distinction in the pseudo-code for the sake of simplicity. Eventually, the algorithm accurately calculates all geodesic distances in G in an elaborate computation. However, we do not need all these distances to be explicitly calculated. For our purposes, it suffices to calculate those distances that have value less than or equal to $\mathcal{L}$. Thus, we can render the computation more efficient by pruning those parts that involve distances already longer than or equal to $\mathcal{L}$. We also prune redundant cases when the two inner loops of the algorithm would check for a path that involves the considered intermediate vertex (k) as either origin or destination (i.e., i or j).

Algorithm 2: $\mathcal{L}$-pruned Floyd-Warshall Algorithm

Input: $G(V, E)$: An undirected graph; $\mathcal{L}$ threshold;
Output: distance matrix of $G(V, E)$ for path lengths $\leq \mathcal{L}$

```
A^0 = adjacency matrix of G(V, E);
for k = 0; k < |V|; k = k + 1 do
    for i = 0; i < |V| − 1; i = i + 1 do
        if (i ≠ k) ∧ (A^k_ik < L) then
            for j = i + 1; j < |V|; j = j + 1 do
                if (j ≠ k) ∧ (A^k_kj < L) then
                    if A^k_ik + A^k_kj ≤ L then
                        A^{k+1}_ij = min(A^k_ij, A^k_ik + A^k_kj);
Return A^{|V|};
```

Algorithm 2 forms an improvement over the naive straightforward application of Floyd-Warshall's algorithm. However, it still performs many redundant operations, as it sequentially scans each row and column of the triangular adjacency matrix, and repeatedly checks whether the scanned distance values are less than $\mathcal{L}$.

We can further improve on Algorithm 2 by avoiding these repeated scans. In a pre-processing step, we scan the adjacency matrix $\mathbf{A}$ and build linked lists that connect those cells of the matrix that contain distance values $\ell_{ij} < \mathcal{L}$ (i.e., in the original state of the matrix, value 1) along each row and column of $\mathbf{A}$. In effect, each cell $\mathbf{A}_{ij}$ such that $\ell_{ij} < \mathcal{L}$ gets two pointers, to its successor cells along the same row and column, with distance values less than $\mathcal{L}$. Our *pointer-based* version of the Floyd-Warshall algorithm rides these linked lists, and appropriately amends them whenever it creates a new cell of distance value less than $\mathcal{L}$. Thus, repeated sequential checks are avoided; sequential scan operations are performed only in the pre-processing step, and during linked-list amendments, whenever a new distance value less than $\mathcal{L}$ is created. This *pointer-based $\mathcal{L}$-pruned* Floyd-Warshall algorithm is shown in Algorithm 3; in the pseudo-code, $nexti$ ($nextj$) is the next cell along the same column (row). We distinguish between the *outer* and the *inner* loop of the conventional Floyd-Warshall algorithm using the notations out and in for the cells they handle, respectively, in order to avoid any confusion caused by the notations i and j (where i denotes a row and j a column of the matrix). As $\mathbf{A}$ is triangular, the only distinguishing mark of these two loops is the fact that one functions as the outer

and the other as the inner one; otherwise, both loops traverse both columns and rows of the matrix; at the k^{th} iteration of the k-loop, the inner loops jointly traverse the k^{th} column and k^{th} row of $\mathbf{A}$, turning from the former to the latter when they reach the diagonal of the matrix.

Algorithm 3: Pointer-based $\mathcal{L}$-pruned F-W Algorithm

```
Input: G(V, E): An undirected graph; L threshold;
Output: distance matrix of G(V, E) for path lengths ≤ L
1  A = adjacency matrix of G(V, E);
2  for k = 0; k < |V|; k++ do
3      out = first cell of column/row k of A with value < L;
4      while out ≠ NULL do
5          if out.i ≠ k then
6              in = out → nexti // next cell along column
7          else
8              in = out → nextj // next cell along row
9          while in ≠ NULL do
10             if in.value + out.value ≤ L then
11                 new = A_coordinates of in, out that are ≠k;
12                 sum = in.value + out.value;
13                 if sum < new then
14                     if (sum < L) ∧ (new ≥ L) then
15                         update connections of cell new;
16                     new = sum;
17             if in.i ≠ k then
18                 in = in → nexti;
19             else
20                 in = in → nextj;
21         if out.i ≠ k then
22             out = out → nexti;
23         else
24             out = out → nextj;
25 Return A;
```

Algorithm 4: Edge Removal Algorithm

```
Input: G(V, E) : An undirected graph; L threshold; confidence threshold θ;
Output: L-opaque graph of G'(V, E') wrt θ
1  G'(V, E') = G(V, E);
2  Calculate degrees of vertices in G, D;
3  while (LO(G') > θ) ∧ (E' ≠ ∅) do
4      best_lo = ∞ // lowest opacity value
5      foreach edge e_ij ∈ E' do
6          E' = E' − e_ij // try removing e_ij
7          lo = LO(G') // achieved L-opacity
8          if lo < best_lo then
9              best_lo = lo; best_pop = N(lo);
10             chosen_edge = e_ij; t = 1;
11         if (lo = best_lo) ∧ (N(lo) < best_pop) then
12             best_pop = N(lo);
13             chosen_edge = e_ij; t = 1;
14         if (lo = best_lo) ∧ (N(lo) = best_pop) then
15             Generate uniform random ρ ∈ [0, 1);
16             t = t + 1;
17             if ρ < 1/t then
18                 chosen_edge = e_ij;
19         E' = E' + e_ij // recover checked edge
20     E' = E' − chosen_edge // remove chosen edge
21 Return G'(V, E');
```

5.2 Edge Removal

Our first approach aims to render an input graph G $\mathcal{L}$-opaque via *edge removal* operations. Given a graph $G(V, E)$, our edge removal algorithm tries to arrive at an $\mathcal{L}$-opaque graph $G'(V, E')$ by greedily removing some of the edges from E. At each step, the algorithm chooses to remove the edge that achieves the lowest opacity value, $\mathcal{LO}(G') = \max_{\mathcal{T} \in \mathcal{C}} \{\mathcal{LO}_{G'}(\mathcal{T})\}$, in the ensuing graph. Should more than one edge achieve the same opacity value, we opt for the edge that minimizes the number of pair types $\mathcal{T}$ that obtain the maximum opacity. We define a function $\mathcal{N}$ as:

$$\mathcal{N}(p) = |\{\mathcal{T} \in \mathcal{C} | \mathcal{LO}_G(\mathcal{T}) = p\}|$$

We opt for the edge that minimizes $\mathcal{N}\big(\mathcal{LO}(G')\big)$ after its removal. The rationale for this choice is that it is preferable to have less than more degree-pairs that reach the highest opacity value. Should more than one edge achieve the same value in that function as well, we pick one of them uniformly at random, while maintaining a counter of such instances. As we witnessed in our experimental study, this state of affairs arises quite often. The pseudocode is shown in Algorithm 4. In the edge removal algorithm with look-ahead (not depicted) we delay this random decision until after checking all the possible combinations of size up to the given *la* threshold. In order to check the combinations of edges, the algorithm starts by combinations of size 1 and after each step increments the size. To be concrete, the algorithm uses a recursive function to generate all the possible combinations of a specific size; in order to save space for every generated combination, it checks the result of removing a combination on the fly.

5.3 Edge Removal and Insertion

Our heuristic based on edge removal alone may successfully achieve the desired $\mathcal{L}$-opacity constraint, yet it does so solely by truncating edges of the input graph. Therefore, the more operations it performs, the more it is bound to diverge from the statistical properties of the original graph. We devise an alternative heuristic that, in addition to, and as a counterweight to, edge removal operations, also performs edge *insertion*. Following a greedy logic similar to the one applied on edge removal, at each step the algorithm chooses to insert the edge that results to a graph of lowest opacity. The heuristic proceeds by performing removals and insertions alternately, thus maintaining the number of edges of the original graph; in order to avoid loops, we never allow the insertion of an edge that has been previously removed, and vice versa. Algorithm 5 shows the pseudocode of this heuristic.

The edge insertion process is symmetric to the edge removal process, which mirrors Algorithm 4, but considers the effects of inserting instead of removing. Edge Removal/Insertion with look-ahead is analogous to Edge Removal with look-ahead.

Algorithm 5: Edge Removal/Insertion Algorithm

```
Input: G(V, E) : An undirected graph; L threshold; confidence threshold θ;
Output: L-opaque graph G(V, E) wrt θ
1  G'(V, E') = G(V, E); E_D = ∅; E_A = ∅;
2  while (LO(G') > θ) ∧ (E' ≠ ∅) do
3      best_lo = ∞ // lowest opacity value (removal)
4      foreach edge e_ij ∈ E' ∩ E_A^c (not inserted before) do
5          E' = E' − e_ij // try removing e_ij
6          lo = LO(G') // achieved L-opacity
7          update best_lo, best_pop, chosen_edge as necessary
8          E' = E' + e_ij // recover checked edge
9      E' = E' − chosen_edge // remove chosen edge
       E_D = E_D ∪ {chose_edge} // set of removed edges
10     best_lo = ∞ // lowest opacity value (insertion)
11     best_pop = ∞ // smallest number of pairs
12     foreach edge e_ij ∉ E' ∪ E_D (not removed before) do
13         E' = E' + e_ij // try inserting e_ij
14         lo = LO(G') // achieved L-opacity
15         update best_lo, best_pop, chosen_edge as necessary
16         E' = E' − e_ij // recover checked edge
17     E' = E' + chosen_edge // insert chosen edge
18     E_A = E_A ∪ {chosen_edge} // set of inserted edges
19 Return G'(V, E');
```

5.4 Complexity Analysis

We now analyze the complexity of our Edge Removal and Edge Removal/Insertion algorithms (Algorithms 4 and 5, respectively).

Algorithm 4 first calculates degrees of vertices by one iteration over the edges in $O(|E|)$. Then, the algorithm enters two nested loops (Lines 3-20); the inner loop tries each candidate for removal,

Data Set	Nodes	Links	Description	
			Nodes	Links
Google	875713	5105039	Web pages	Hyperlinks
Berkeley-Stanford	685230	7600595	Web pages	Hyperlinks
Epinions	132000	841372	Users	717667 Trusts and 123705 Distrusts statements
Enron	36692	367662	Email addresses	Transferred emails
Gnutella	10876	39994	Hosts in network topology	Connections
ACM Digital Library	10000	19894	Authors	Co-Authors
Wikipedia	7115	103689	Users and candidates	Votes

Table 1: Description of the original datasets

while the outer loop iterates until, in the worst case, no edge is left; thus these nested loops require $O(|E|^2)$ iterations. The most computationally intensive part of the inner loop (Lines 5-19) is the computation of the $\mathcal{L}$-opacity achieved after every removal at Line 7, which invokes Algorithm 1. In its turn, Algorithm 1 elicits a $O(|V|^3)$ calculation of the distance matrix by the Pointer-based $\mathcal{L}$-pruned F-W algorithm, Algorithm 3. Then, Algorithm 1 goes through two loops; the former (Lines 3-6) iterates over every pair of vertices in the distance matrix in $O(|V|^2)$; the latter (Lines 7-12) iterates over each degree pair in $O((|d_{max}| - |d_{min}|)^2) = O(|V|^2)$. In effect, the complexity of Algorithm 1 is dominated by the $O(|V|^3)$ term. Eventually, the worst-case complexity of Algorithm 4 is $O(|E| + |E|^2 \times |V|^3) = O(|V|^7)$.

Similarly, the Edge Removal/Insertion algorithm, Algorithm 5, attempts to either remove or insert each possible edge in the graph at most *once*, until no further candidates for either removal or insertion exist (Lines 2-18). Hence, these loops require $O(|V|^4)$ iterations. At each iteration, Algorithm 5 also invokes the $O(|V|^3)$ Algorithm 1; hence, in total, Algorithm 5 also raises a $O(|V|^7)$ worst-case time complexity.

As our experimental study will demonstrate, these worst-case complexity requirements are ameliorated in practice, as the algorithms satisfy their termination conditions without having to exhaustively examine all possible candidate edges. In effect, we achieve a runtime growth *linear* in number of nodes, instead of the septic polynomial complexity predicted in the worst-case scenario.

In both Edge Removal algorithm (Alg. 4) and Edge Removal and Insertion algorithm (Alg. 5) the input graph in form of adjacency list is stored in the main memory. In addition to the input graph, we need to maintain the distance matrix and the Opacity matrix as explained in the subsection 5.1. Therefore, the total space complexity of both algorithms is $O(|V|^2)$.

6. EXPERIMENTAL EVALUATION

We now experimentally evaluate our heuristics for attaining $\mathcal{L}$-opacity in terms of the alterations they inflict on the graph they operate on. Each heuristic, Edge Removal (Rem) and Edge Removal and Insertion (Rem-Ins), can expand its search space by varying the look-ahead (la) parameter. Furthermore, we implement the three heuristics proposed by Zhang and Zhang [29] in order to compare them against our heuristics. This comparison is only appropriate when $\mathcal{L} = 1$, as [29] considers only single-edge linkages, while our model and methods consider connections of length up to $\mathcal{L}$. Therefore, we cannot conduct comparisons to [29] when $\mathcal{L} \geq 2$.

Zhang and Zhang in [29] propose *GADED-Rand*, *GADED-Max* and *GADES* heuristics. *GADED-Rand* removes a random edge among the edges participating in disclosure in each step; *GADED-Max* removes an edge with maximum reduction of the maximum link disclosure and minimum increase of the total link disclosures. The last heuristic, *GADES*, finds a pair of edges for swapping that can reduce the maximum link disclosure in each iteration.

We implemented and evaluated all algorithms on an IBM X3550

Data Set	Diameter	Av. Deg.	STDD	ACC
Google	22	11.6	16.4	0.6047
Berkeley-Stanford	669	22.1	10.99	0.6149
Epinions	9	12.7	32.68	0.1062
Enron	12	20	18.58	0.4970
Gnutella	9	7.4	3.01	0.0080
ACM Digital Library	400	3.97	6.23	0.5279
Wikipedia	7	29.1	60.39	0.2089

Table 2: Dataset properties

Data Set	Nodes	Links	Diameter	Av. Deg.	STDD	ACC
Google	100	746	7	14.92	11.13	0.76
Google	500	3104	15	12.42	10.54	0.70
Google	1000	6445	25	12.89	12.62	0.70
BS	500	4454	6	17.82	21.50	0.62
Epinions	100	65	4	1.3	0.72	0.04
Enron	100	346	4	6.92	9.28	0.31
Enron	500	5686	4	22.74	25.81	0.37
Gnutella	100	116	6	2.32	3.00	0.05
Gnutella	500	721	8	2.88	3.19	0.09
Gnutella	1000	1852	8	3.71	3.51	0.02
Wikipedia	100	919	3	18.38	15.19	0.54
Wikipedia	500	7244	4	28.98	33.02	0.39

Table 3: Sampled graph properties

Intel Xeon 3.16 GHz 64-bit processor cluster of 64 CPUs / 256 GB of main memory uniformly distributed among 8 nodes. The nodes operate CentOS 6.2 with gcc 4.4.6. We repeat each experiment 10 times for each θ value, and select the graph of minimum distortion.

6.1 Description of Data

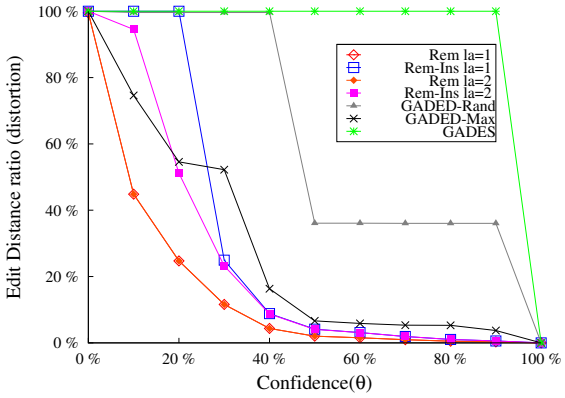
We use seven real-world data sets. Table 1 shows the size of the original datasets in terms of vertices and edges, and the domains these vertices and edges describe. We have randomly sampled the vertices of six of these seven data sets to derive smaller graphs of $100 - 1000$ nodes. The edges in the sampled graph are the adjacent edges of the sampled nodes. These six data sets are obtained from the Stanford Large Network Dataset[1] collection. Our seventh data set, used in our last experiments, is a extracted by crawling $10,000$ nodes from the ACM Digital Library.

Table 2 presents some of the properties of the original data sets that we have sampled for use in our experiments. Diameter is the longest shortest path in a graph; Av. Deg. and STDD stand for the average and standard deviation of the degrees, respectively, and ACC stands for a graph's average clustering coefficient. For each dataset we take three samples with 100, 500 and 1000 vertices. Table 3 presents properties of these *sampled* graphs for several sizes.

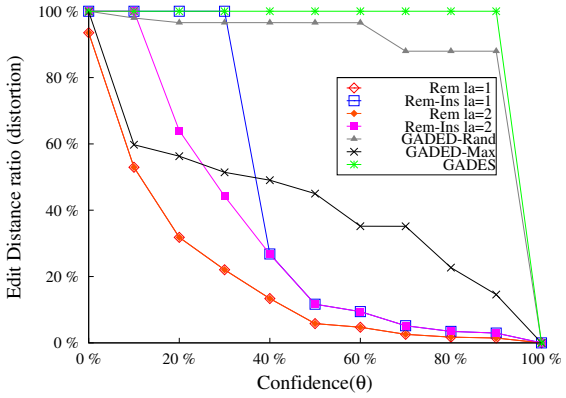
6.2 Utility metrics

Apart from the distortion measure (Equation 1), we employ two other measures of alteration and utility, the Earth-Mover's Distance (EMD) among distributions [20] and the Clustering Coefficient. We compute EMD between the degree and geodesic distance distribu-

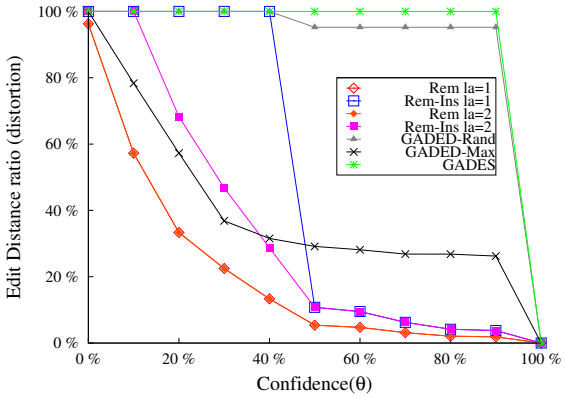
[1] Available online at http://snap.stanford.edu/data/



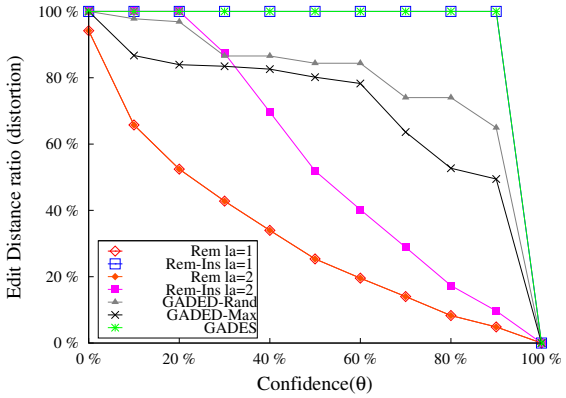
(a) Google, $\mathcal{L} = 1$



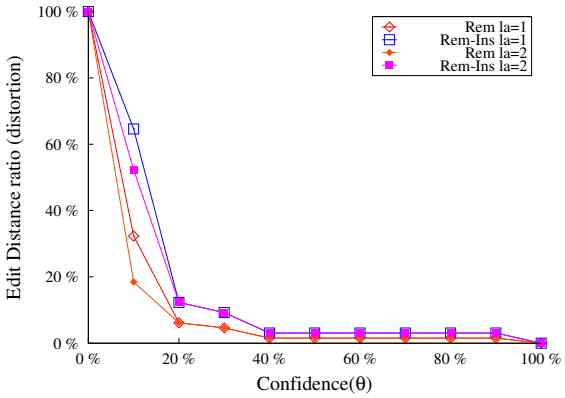
(b) Wikipedia, $\mathcal{L} = 1$



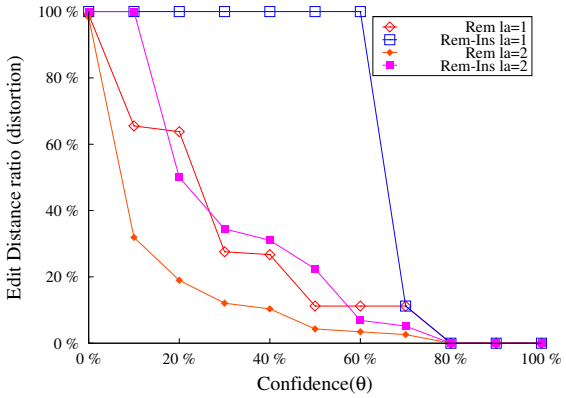
(c) Enron, $\mathcal{L} = 1$



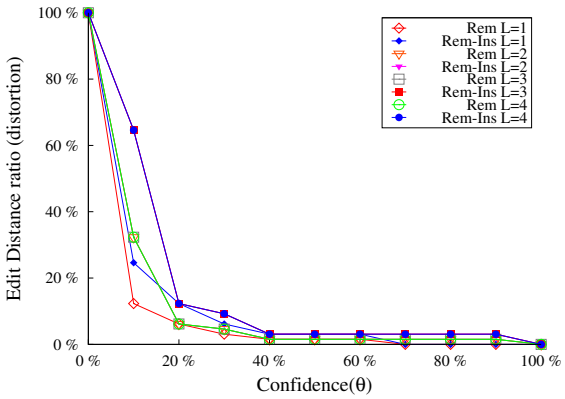
(d) B-S, $\mathcal{L} = 1$



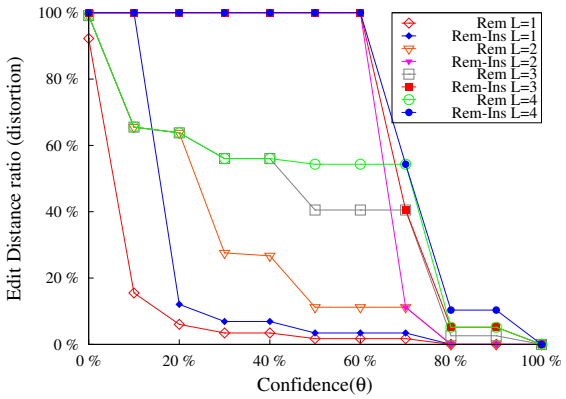
(e) Epinions(Trust), $\mathcal{L} = 2$



(f) Gnutella, $\mathcal{L} = 2$

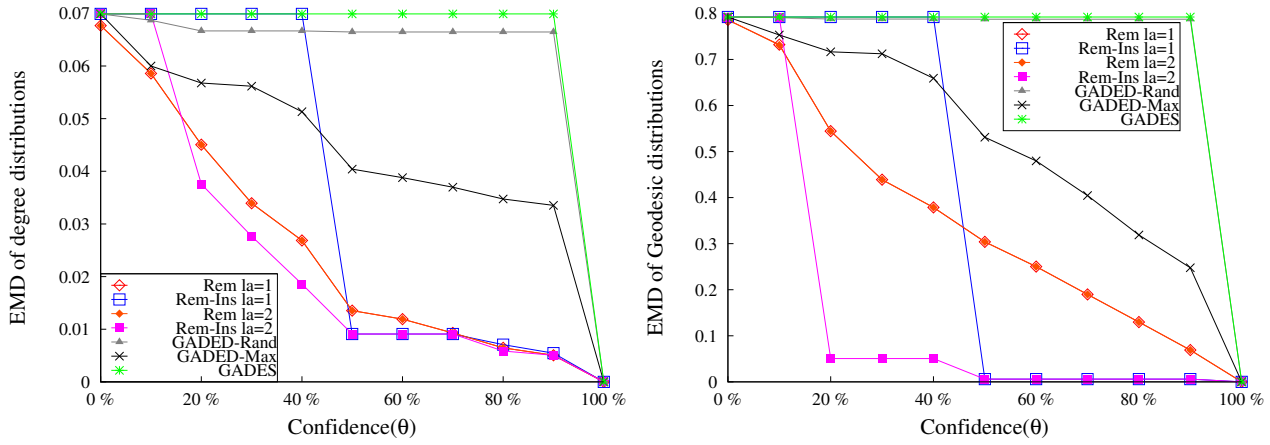


(g) Epinions(Trust), $la = 1$



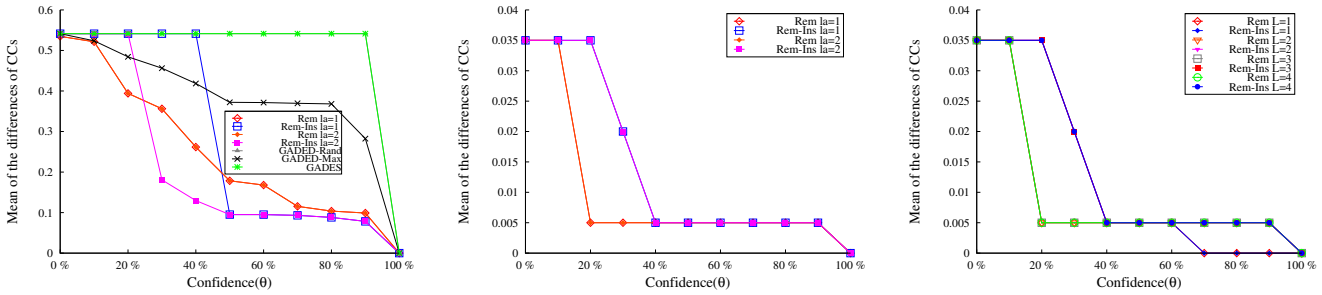
(h) Gnutella, $la = 1$

Figure 6: Graph edit distance ratio (Distortion) vs. θ



(a) EMD of degree distribution (b) EMD of geodesic distribution

Figure 7: EMD of distributions vs. θ



(a) Wikipedia, $\mathcal{L} = 1$ (b) Epinions(Trust), $\mathcal{L} = 2$ (c) Epinions(Distrust), $la = 1$

Figure 8: Mean of the differences of Clustering Coefficients vs. θ

tions in the original graph and the altered graph. We emphasize that we use the EMD measure among distributions only as a way of assessing the amount of alteration inflicted on a graph. As we have discussed, our publication model does publish the original degree of each node, hence that information itself is always preserved.

A clustering coefficient indicates the extent to which nodes tend to cluster together. It can be measured either as a global metric for the whole graph, or as a local metric for every vertex. We measure the *local clustering coefficient* for each vertex v_i, $\mathcal{C}_i = \frac{|\{e_{jk} \in E | e_{ij}, e_{ik} \in E\}|}{|N_i| * (|N_i|-1)}$ where N_i is the number of neighbors of v_i and $|e_{jk}|$ is the number of edges among those neighbors. In order to measure the difference of clustering coefficient between an original and an anonymized graph, we calculate $\Delta\mathcal{C}_i = |\mathcal{C}_i - \mathcal{C}_i^{'}|$ for every vertex and report the mean of $\Delta\mathcal{C}_i$.

6.3 Comparison on Distortion

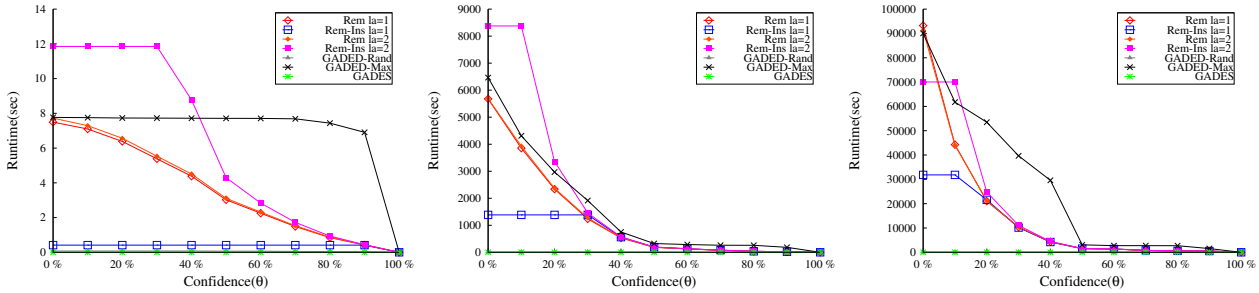
We first compare the performance of our two heuristics for different look-ahead (la) on the Distortion measure, as a function of the θ parameter, with the six sampled data sets; the θ and $\mathcal{L}$ parameters together define the privacy condition we wish to achieve. The less the θ the less the adversary's confidence of the linkage disclosure, and hence the more secure the relations are against threats.

Figures 6(a,b,c,d) shows our results for $\mathcal{L}=1$, and Figures 6(e,f) for $\mathcal{L}=2$. Notably, the look-ahead assists the Removal/Insertion heuristic for every $\mathcal{L}$ and the Removal heuristic for $\mathcal{L} \geq 2$. This advantage appears clearly with the Berkeley-Stanford data, with which the Removal/Insertion heuristic with one look-ahead (Rem-Ins $la = 1$) cannot find a solution, while increasing the look-ahead to two (Rem-Ins $la = 2$) allows the heuristic to find solutions even for $\theta = 30\%$. The advantage of look-ahead with edge Removal is seen for the Gnutella network when $\mathcal{L}=2$, while Removal achieves lower distortion than Removal/Insertion. This experiment also shows that we can find an $\mathcal{L}$-opaque graph with $\theta = 50\%$ for all the datasets with a distortion of less than 20%. Our heuristics, which opt for an edge that minimizes $\mathcal{N}\big(\mathcal{LO}(G')\big)$ (number of degree-pairs $\mathcal{T}$ that obtain the maximum opacity), obtain a clear advantage in comparison to the heuristics of [29], which aim to minimize the total increase of the linking probabilities. Besides, with all datasets, the edge swapping technique (GADES) cannot find any $\mathcal{L}$-opaque graph unless returning an empty graph.

Charts (g) and (h) in Figure 6 show the amount of distortion for fixed look-ahead of *one* when varying the $\mathcal{L}$ from *one* to *four* for the two of the sampled datasets. For every $\mathcal{L}$, the Removal heuristic always finds an opaque graph with lower distortion, which means less modifications for the same confidence threshold in comparison to the Removal/Insertion heuristic. These results also suggest that the impact of $\mathcal{L}$ on the amount of distortion is lower for the sparser graphs (the sample of Epinions(Trust) network has 130 edges while Gnutella network has 232 edges).

6.4 Comparison on EMD

Next, we compare the performance of the heuristics on the EMD measure of degree distributions and geodesic distributions, with regards to the θ, $\mathcal{L}$ and la parameters. Figure 7a shows the results of the EMD between the degree distributions for the Enron network when $\mathcal{L}=1$. For θ greater than 20%, the Removal/Insertion heuristic results in an $\mathcal{L}$-opaque graph with less EMD in comparison to the Removal heuristic. This is due to the fact that in the Removal heuristic, edges are only removed and never inserted, hence the frequency counts of the high degrees are only decreased. On the other hand, the Removal/Insertion heuristic allows the reduction



(a) Google ($|V|$=100) (b) Google ($|V|$=500) (c) Google ($|V|$=1000)

Figure 9: Runtime vs. θ

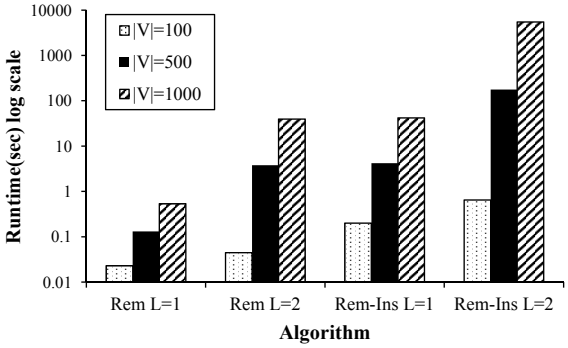


Figure 10: Runtime comparison

of frequency counts, caused by removal, to be compensated by insertion. However, after some point (here for θ less than 30%), removing/inserting pairs of edges eventually increases the EMD. The Removal/Insertion heuristic preserves the total number of edges, hence is more likely to preserve the degree distribution of the original graph more accurately. By contrast, the heuristics of [29] always performs poorly. Among them, GADED-Max achieves the best performance, but is still outperformed by at least one or the other of our look-ahead-based methods. This advantage can be seen in both of our look-ahead heuristics for all the datasets.

Figure 7b shows the results for the EMD between the geodesic distributions for the Enron network when $\mathcal{L}=1$. As θ decreases, the Removal/Insertion heuristic incurs less modification, as measured by EMD, to geodesic distances in comparison to the Removal heuristic. This is due to the fact that the Removal/Insertion heuristic can compensate some of the geodesics destroyed by edge removal via edge insertion. However, this compensation does not always carry on when we further decrease θ, and the EMD of Removal eventually becomes smaller than that of Removal/Insertion. Overall, our look-ahead heuristics always outperform those of [29].

Altogether, the EMD of the two distributions follow the same pattern as the distortion results. Furthermore, the results show that keeping the number of edges of the graph constant is hard to attain. This result indicates that Removal/Insertion can be the heuristic of choice for attaining $\mathcal{L}$-opacity except in settings where the desired value of θ is not easily attainable for the given value of $\mathcal{L}$; in such cases, we face the trade-off of either increasing the look-ahead at the expense of runtime or opting for the Removal heuristic at the expense of utility. Besides, the charts (e) and (f) in Figure 7b both indicate that larger $\mathcal{L}$ requires more modification to the graph.

6.5 Comparison on Clustering Coefficients

Figure 8 shows the results of the mean of the clustering coefficient differences between the original graph and the anonymized graph (explained in the subsection 6.2). This figure shows that for large value of θ the Removal/Insertion heuristic changes the CC less than the Removal. Just removing edges, as in the Removal heuristic, breaks the edges among the neighbors of the vertices and hence reduces the clustering coefficient. However removing more edges, will reduce the number of neighbors of the vertices and this is the reason for the better performance of Removal heuristic for small θ in comparison to the Removal/Insertion.

Figure 8 also shows that the Removal heuristic finds anonymized graphs with smaller change to the CC in comparison to the best competing heuristic of [29], namely GADED-Max.

6.6 Runtime comparison

Figure 9 shows the runtime of sampled graphs of the Google network with 100, 500 and 1000 nodes. We record the time for varying θ from 100% to 0% with steps of 10%. As soon as an algorithm finds a solution with less θ than the previous achieved θ, we record the time for all the θ values in between as the same time. Therefore some heuristics present the same time for different θ values. For the GADES algorithm, a constant time appears simply due to its inability to find a solution better than an empty graph.

Remarkably, the best-performing heuristic of [29], GADED-Max, not only results in graphs of less utility, but is also always slower than our Removal heuristic. Figure 9 also shows that, as we increase the la parameter, the runtime of the Removal/Insertion heuristic is affected significantly, while that of the Removal heuristic is affected minimally. This large increase in runtime is due to its significantly expanded search space, which helps the heuristic find a solution of higher utility at the cost of extra runtime.

We also measure the runtime of the two proposed $\mathcal{L}$-opacification heuristics for graphs of different sizes. Figure 10 shows the results. The graphs are sampled from the Gnutella data, tuning the size to 100, 500 and 1000 nodes. We record the runtime of each algorithm for $\mathcal{L}$ thresholds, 1 and 2. As expected, runtime grows with graph size and $\mathcal{L}$. Moreover, on the same data, the Removal algorithm is faster than Removal/Insertion; this is due to the fact that, at each iteration, the Removal/Insertion algorithm needs to try all possible edges, whose number is larger than that of existing ones.

6.7 Testing Larger Graphs

Our experimental study was hitherto limited to sample data sets of up to 1000 nodes, on which our algorithms elicit reasonable runtime. Nevertheless, we argue that our algorithms can be used for larger data as well, when the need arises, provided sufficient computational resources. To illustrate this point, we end our experimental study with an experiment on graph sizes ranging from 1000 to 10000 nodes and 3874 to 39788 edges, sampled from the ACM Digital Library data set.

We ran our Edge Removal algorithm on these data for $\mathcal{L}=1$ and θ ranging from 50% to 90% . Figures 11 and 12 show our runtime and distortion results, respectively. As expected, the runtime

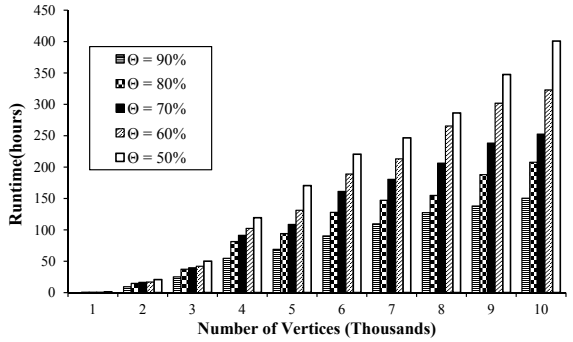


Figure 11: Runtime vs. size with variant θ

grows both with data size and with decreasing confidence threshold θ (i.e., increasing adversary's uncertainty). Our longest-running experiment, on the ACM dataset with 10,000 nodes for confidence threshold $\theta = 50\%$, took approximately 16 days. At the same time, we observe that runtime grows linearly in both size and θ.

This long-running experiment on graphs of increasing size reveals that, as data size grows, a solution at the same privacy level can be obtained for less distortion, as shown in Figure 12. Thus, according to this result, it becomes increasingly attractive for a data vendor to publish large graphs offering the $\mathcal{L}$-opacity privacy guarantee, since the same guarantee can be delivered at lower information loss as the size of the published graph grows. Besides, in order to achieve this advantage, a data vendor may reasonably be willing to invest the linearly increasing runtime observed in Figure 11.

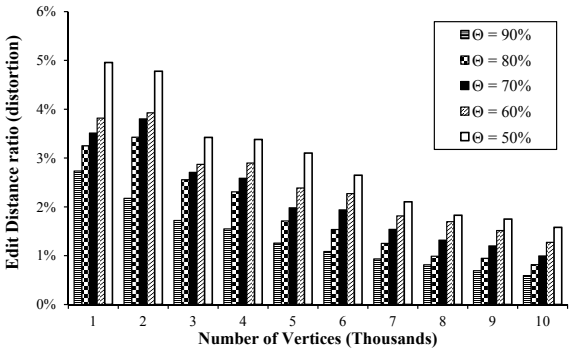


Figure 12: Distortion vs. size with variant θ

7. CONCLUSIONS

In this paper we examined the problem of anonymizing graph data, with a focus on preventing the disclosure of sensitive information that pertains to linkages. We formulated a specific yet practically relevant instance of this problem, in which the aim is to prevent an adversary who possesses background information about node degrees in the original network from inferring the existence of any short-path connection among nodes with high confidence. We defined $\mathcal{L}$-opacity, a precise and sufficiently strong privacy condition that encapsulates this requirement, and formulated two effective and comparatively efficient greedy heuristics that attempt to inflict minimal alterations on the graph so as to abide by $\mathcal{L}$-opacity.

Our experimental study demonstrates that our heuristic performing edge Removal/Insertion is better disposed to preserve key properties of the graph than one that only removes edges, while the latter is more capable of always arriving at an alteration of the graph that satisfies the problem constraints. Moreover, we demonstrate that our two heuristics outperform a recently proposed method which applies only to a limited version of our problem, in terms of both the alteration they incur to the original graph and runtime.

8. REFERENCES

[1] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four degrees of separation. In *WebSci*, 2012.

[2] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou R3579X?: Anonymized social networks, hidden patterns, and structural steganography. In *WWW*, 2007.

[3] S. Bhagat, G. Cormode, B. Krishnam, and D. Srivastava. Privacy in dynamic social networks. In *WWW*, 2010.

[4] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. *PVLDB*, 2(1):766–777, 2009.

[5] A. Campan and T. M. Truta. Data and structural k-anonymity in social networks. In *PinKDD*, 2008.

[6] J. Cheng, A. W.-C. Fu, and J. Liu. k-isomorphism: Privacy-preserving network publication against structural attacks. In *SIGMOD*, 2010.

[7] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *The VLDB Journal*, 19(1):115–139, 2010.

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

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[10] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast data anonymization with low information loss. In *VLDB*, 2007.

[11] S. Goel, R. Muhamad, and D. Watts. Social search in "small-world" experiments. In *WWW*, 2009.

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

[13] X. He, J. Vaidya, B. Shafiq, N. Adam, and V. Atluri. Preserving privacy in social networks. In *WI-IAT*, 2009.

[14] A. Korolova, R. Motwani, S. U. Nabar, and Y. Xu. Link privacy in social networks. In *CIKM*, 2008.

[15] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *WWW*, 2008.

[16] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.

[17] S. Milgram. The small world problem. *Psych. Today*, 2:60–67, 1967.

[18] S. Nobari. *Scalable Data-Parallel graph algorithms from generation to management*. PhD thesis, National University of Singapore, 2012.

[19] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, 2009.

[20] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. 40(2):99–121, 2000.

[21] P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, 2001.

[22] Y. Song, P. Karras, S. Nobari, G. Cheliotis, M. Xue, and S. Bressan. Discretionary social network data revelation with a user-centric utility guarantee. In *CIKM*, 2012.

[23] Y. Song, P. Karras, Q. Xiao, and S. Bressan. Sensitive label privacy protection on social network data. In *SSDBM*, 2012.

[24] D. J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton, 2003.

[25] W. Wu, Y. Xiao, W. Wang, Z. He, and Z. Wang. k-symmetry model for identity anonymization in social networks. In *EDBT*, 2010.

[26] M. Xue, P. Karras, R. Chedy, P. Kalnis, and H. K. Pung. Delineating social network data anonymization via random edge perturbation. In *CIKM*, 2012.

[27] X. Ying and X. Wu. On link privacy in randomizing social networks. In *PAKDD*, 2009.

[28] M. Yuan, L. Chen, and P. S. Yu. Personalized privacy protection in social networks. *PVLDB*, 4(2):141–150, 2010.

[29] L. Zhang and W. Zhang. Edge anonymity in social network graphs. In *CSE*, 2009.

[30] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *PinKDD*, 2007.

[31] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, 2008.

[32] L. Zou, L. Chen, and M. T. Özsu. k-automorphism: A general framework for privacy-preserving network publication. *PVLDB*, 2(1):946–957, 2009.