Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Wei Peng

Entitled
Seed and Grow: An Attack Against Anonymized Social Networks

For the degree of Master of Science

Is approved by the final examining committee:

Xukai Zou
Chair

Feng Li

Yuni Xia

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Xukai Zou

Feng Li

Approved by: Rajeev Raje 06/23/2011
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Seed and Grow: An Attack Against Anonymized Social Networks

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Wei Peng

Printed Name and Signature of Candidate

06/22/2011

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

SEED AND GROW:

AN ATTACK AGAINST ANONYMIZED SOCIAL NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Wei Peng

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2011

Purdue University

Indianapolis, Indiana

To Mom and Dad: *you* are the why.

ACKNOWLEDGMENTS

First and foremost, to my advisors, or, more truthfully, mentors and friends, Dr. Feng Li and Dr. Xukai Zou. Words alone fall short of my gratitude; I will just be plain. I am grateful for you

- taking me onboard when I was wandering;
- initiating me into the joys and pains of scientific research;
- putting yourselves in my shoes and supporting me;
- making a pitch for me beyond your duty;
- trusting and encouraging me when I was in doubt;
- and showing me life is, after all, larger than work.

I want to thank my professors in the past two and half years for their classes and inspirations: Dr. Arjan Durresi, Dr. Yao Liang, Dr. Yuni Xia, Dr. Mihran Tuceryan, and Dr. James Hill. Special thanks are due to Dr. Xia for serving on my thesis committee.

To the rest of the faculty members, Dr. Shiaofen Fang, Dr. Rajeev Raje, Dr. Jiang Yu Zheng, Dr. Mohammad Al Hasan, Dr. Murat Dundar, Dr. Jake Yue Chen, Dr. Snehasis Mukhopadhyay, Dr. Andrew Olson, Dr. Gavriil Tsechpenakis, and Ms. Lingma Acheson, thank you for the greetings and smiles exchanged in the corridor and after the weekly seminar, which make the department feel like a home.

Things would not work out so smoothly without the cheerful and kind souls that keep the department running. Thank you, Nicole, Josh, DeeDee, Scott, Leah, Debbie, and Nancy.

To my friends (you know who you are): thank you for making the past years so wonderful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

G_T, V_T, E_T	Target graph, its vertices, and its edges; $G_T = \{V_T, E_T\}$.
G_B, V_B, E_B	Background graph, its vertices, and its edges; $G_B = \{V_B, E_B\}$.
G_F, V_F, E_F	Flag graph, its vertices, and its edges; $G_F = \{V_F, E_F\}$.
V_S	Seed; $V_S \subset V_B \cap V_T$; initially connected with V_F.
$V_F(u)$	The vertices in V_F which are connected with $u \in V_S$.
v_h	The head vertex in V_F.
$D_F(u)$	The internal degree for $u \in V_F - \{v_h\}$.
$S_{\mathcal{D}}$	The ordered internal degree sequence of all vertices in $V_F - \{v_h\}$.
$S_{\mathcal{D}}(v)$	The sub-sequence of $S_{\mathcal{D}}$ for $v \in V_S$.
$N_m^T(u), N_m^B(u)$	The mapped neighbors of u in the target/background graph.
$N_u^T(u), N_u^B(u)$	The unmapped neighbors of u in the target/background graph.
$\Delta_T(u,v), \Delta_B(u,v)$	The dissimilarity between u and v in the target/background graph.
$\mathcal{E}_X(x)$	The eccentricity of a number $x \in X$.

ABSTRACT

Peng, Wei M.S., Purdue University, August 2011. Seed and Grow: An Attack Against Anonymized Social Networks. Major Professor: Feng Li and Xukai Zou.

Digital traces left by a user of an on-line social networking service can be abused by a malicious party to compromise the person's privacy. This is exacerbated by the increasing overlap in user-bases among various services.

To demonstrate the feasibility of abuse and raise public awareness of this issue, I propose an algorithm, *Seed and Grow*, to identify users from an anonymized social graph based solely on graph structure. The algorithm first identifies a *seed* sub-graph either planted by an attacker or divulged by collusion of a small group of users, and then *grows* the seed larger based on the attacker's existing knowledge of the users' social relations.

This work identifies and relaxes implicit assumptions taken by previous works, eliminates arbitrary parameters, and improves identification effectiveness and accuracy. Experiment results on real-world collected datasets further corroborate my expectation and claim.

1 INTRODUCTION

A lunch-time walk across a university campus in the United States might lead one to marvel at the prevalence of Internet-based social networking services, among which Facebook and Twitter are two big players in the business. Indeed, as Alexa's "top 500 global sites" statistics retrieved on May 2011 indicates, Facebook and Twitter rank at 2nd and 9th place, respectively.

One characteristics of on-line social networking services is their emphasis on users and their connections, rather than on content as traditional Web services do. These services, while providing conveniences to users, accumulate a treasure of user-produced contents and users' social connection patterns, which were only available to large telecommunication service providers or intelligence agencies a decade ago.

Data from social networks, once published, are of great interest to a large audience. For example, with the massive data sets, sociologists can verify hypotheses on social structures and human behavior patterns. Third-party application developers can produce value-added services like games based on users' contact lists. Advertisers can more accurately infer users' demographic and preference profile and issue targeted advertisement. Indeed, the 22 December 2010 revision of Facebook's Privacy Policy has the following clause, "we allow advertisers to choose the characteristics of users who will see their advertisements and we may use any of the non-personally identifiable attributes we have collected (including information you may have decided not to show to other users, such as your birth year or other sensitive personal information or preferences) to select the appropriate audience for those advertisements".

Due to the strong correlation between users' data and the users' social identity, privacy is a major concern in dealing with social network data in contexts such as

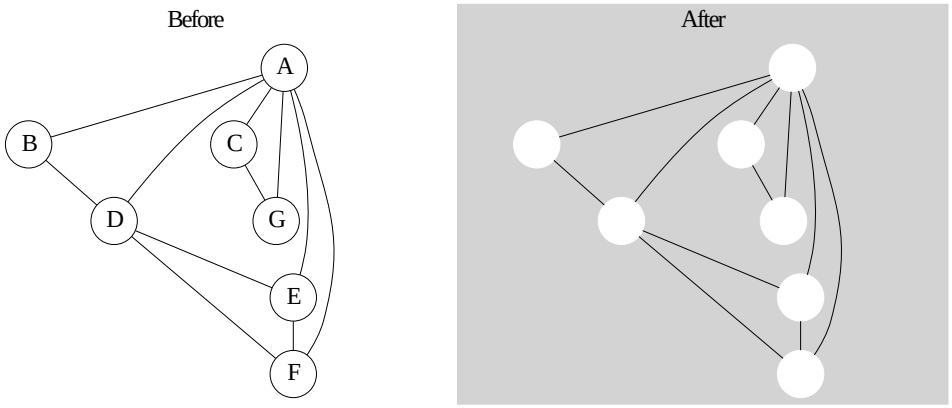


Figure 1.1. An illustration of naive anonymization. Each node represents a user, with the user's ID attached. Naive anonymization simply removes the ID, but retains the network structure.

storage, processing and publishing. Privacy control, through which a user can tune the visibility of her profile, is an essential feature in any major social networking service.

The common practice for privacy-sensitive social network data publishing is through anonymization, i.e., remove plainly identifying labels such as name, social security number, postal or e-mail address, and retain the structure of the network as published data. Figure 1.1 is a simple illustration of this process. The motivation behind such processing prior to data publishing is that, by removing the "who" information, the utility of the social networks is maximally preserved without compromising users' privacy. Narayanan and Shmatikov[1] report several high-profile cases in which "anonymity has been unquestioningly interpreted as equivalent to privacy".

Can the aforementioned "naive" anonymization technique achieve privacy preservation in the context of privacy-sensitive social network data publishing? This interesting and important question was posed only recently by Backstrom et al.[2]. A few privacy attacks have been proposed to circumvent the naive anonymization protection[1, 2]. Meanwhile, more sophisticated anonymization techniques[3, 4, 5, 6, 7] have been proposed to provide better privacy protection. Nevertheless, research in

this area is still in its infancy and a lot of work, both in attacks and defenses, remain to be done.

In this dissertation, I propose a two-stage identification attack, *Seed-and-Grow*, against anonymized social networks. The name suggests a metaphor for visualizing its structure and procedure. The attacker first plants a *seed* into the target social network before its release. After the anonymized data is published, the attacker retrieves the seed and makes it *grow* larger, thereby further breach privacy.

More concretely, my contributions include

- I propose an efficient seed construction and recovery algorithm (Section 3.1). More specifically, I identify and relax the assumption for unambiguous seed identification and drop the assumption that the attacker has complete control over the connection between the seed and the rest of the graph (Section 3.1.1); the seed is constructed in a way which is only visible to the attacker (Section 3.1.1); the seed recovery algorithm examines at most two-hop local neighborhood of each node and thus is efficient (Section 3.1.2).

- I propose an algorithm which grows the seed (i.e., further identifies users and hence violates their privacy) by exploiting the overlapping user bases among social network services. Unlike previous works which rely upon arbitrary parameters on probing aggressiveness, my algorithm automatically finds a good balance between identification effectiveness and accuracy (Section 3.2).

- I demonstrate significant improvements in identification effectiveness and accuracy of the Seed-and-Grow algorithm over previous works with real-world social-network datasets.

In light of the increasing overlapping user bases among social network services, businesses and government agencies should realize that *privacy protection is not only an individual responsibility but also a social one.* This work calls for a re-evaluation of the current privacy-protection practices in publishing social-network data.

2 BACKGROUND AND RELATED WORK

The two most important entities in a social network are *social actors* (i.e., users in a social networking service) and the *relations* between pairs of social actors. Each social actor has a set of associated attributes, such as name, gender, or age. Moreover, each relation between a pair of social actors may also have attributes. An example is a telephone contact history network, in which one possible numerical attribute on links is the total number of calls made between two phones in the past two months.

A natural mathematical model to represent a social network is a graph. A graph G consists of a set V of vertices and a set $E \subseteq V \times V$ of edges. Labels can be attached to both vertices and edges to represent their attributes.

In this context, *privacy* can be modeled in terms of these different components of a graph. Indeed, Zhou et al.[8] categorize privacy as the knowledge of existence or absence of vertices, edges, or labels. One special category is the graph metrics, in which privacy was modeled not in terms of individual component of a graph (e.g., vertices), but in terms of metrics originated from social network analysis studies[9, 10], such as betweenness, closeness, and centrality.

The naive anonymization is to remove those labels which can be uniquely associated with one vertex (or a small group of vertices) from V. This is closely related to traditional anonymization techniques employed on relational datasets[11, 12]. However, the additional information conveyed in edges and its associated labels opens up a new dimension of potential privacy breaches, from which Backstrom et al.[2] propose an identification attack against anonymized graph and coined the term *structural steganography*.

Beside privacy, other dimensions in formulating privacy attack against anonymized social networks, as identified in numerous previous works[4, 5, 7, 8], are the published data's *utility*, and the attacker's *background knowledge*.

Utility of published data measures information loss and distortion in the anonymization process. The more information is lost or distorted, the less useful published data is. Existing anonymization schemes[3, 4, 5, 7, 8] are all based on the trade-off between usefulness of the published data and strength of protection. For example, Hay et al.[7] propose an anonymization algorithm in which the original social graph is partitioned into groups before publication, and "the number of nodes in each partition, along with the density of edges that exist within and across partitions", are published.

Zhou et al.[8] categorize existing anonymization methods into two general types, namely, clustering-based approaches and graph modification approaches. Clustering-based approach[7] clusters vertices and edges into groups and anonymizes a subgraph into a super vertex. In contrast, the anonymization techniques adopted in the graph modification approach[4] is more local, by modifying graph elements like vertices and edges in a way that make a node hard to be identified from a group, while still keep some important graph metrics.

Although trade-off between utility and privacy is necessary[13], it is hard, if not impossible, to find a proper balance in general. Besides, it is hard to prevent attackers from proactively collecting intelligence on the social network. It is especially relevant today as major online social networking services provide APIs to facilitate third-party application development. These programming interfaces can be abused by a malicious party to gather information about the network.

Background knowledge characterizes the information in the attacker's possession which can be used to compromise privacy protection. It is closely related to what is perceived as privacy in a particular context. For example, Zhou et al.[8] categorize (user's) privacy types and defines a (attacker's) background knowledge model for each type.

In the battle between protecting and compromising privacy, the attacker always has an upper hand because he may obtain some information unknown to the defender. Many existing privacy protection mechanisms assume an adversary with limited background knowledge. For example, in one particular model[3], the attacker is assumed to only know the degree sequence around the target. Though it is necessary for understanding privacy threat to make assumptions on the attacker's capability, the assumptions should nevertheless be realistic for the protection mechanism to be effective. The strength of protection depends on the effort required for the attacker to gather enough information to breach privacy, not on arbitrary assumption on the attacker's capability.

The attacker's background knowledge is not restricted to the target's neighborhood in a single network, but may span multiple networks and include the target's alter egos in all these networks[1]. This is a realistic assumption. Consider the status quo in the social networking service business, in which service providers, like Facebook and Flickr, offer complementary services. It is very likely a user of one service would simultaneously use another service[14]. As a person registers to different social networking services, her social connections in these services, which somehow relates to her social relationships in the real world, might reveal valuable information which the attacker can make use of to threaten her privacy.

The above observation inspires Seed-and-Grow, which exploits the increasingly overlapping user-bases among social networking services. A concrete example is helpful in understanding this idea.

> [Motivating Scenario] Bob, as an employee of a social networking service provider F-net, acquires from his employer a graph, in which vertices represent users and edges represent private chat logs. The edges are labeled with attributes such as timestamps. In accordance with its privacy policy, F-net has removed users' ID from the graph before giving it to Bob.

> Bob, being an inquisitive person, wants to know who these users are. Suppose, somehow, Bob identifies 4 of these users from the graph (this will become clear in the "Seed Construction" and "Seed Recovery" interludes in Section 3.1). By using a graph (with user ID tagged) he crawled a month ago from the website of another service provider T-net (the 4 identified persons are also users of T-net) and carefully measuring structural similarity of these graphs, he manages to identify 10 more persons from the anonymized graph from F-net (the "Dissimilarity" interlude in Section 3.2 will illustrate how to do this).
>
> By doing so, Bob defeats his employer's attempt to protect the customers' privacy.

I conclude this chapter with a brief comment on the choice of model. *Undirected* graph is used to represent social networks, which arises naturally in scenarios where the relation under investigation is *mutual*, e.g., friend requests must be confirmed in Facebook. In contrast, *directed* graph is a natural model in other cases, e.g., a fan follows a movie star in Twitter. A directed graph reveals more information about the social relationships than its undirected counterpart. Thus, results on de-anonymizing undirected graphs can be extended without essential difficulty to directed graphs.

3 SEED-AND-GROW: THE ATTACK

This chapter studies an attack, Seed-and-Grow, that identifies users from an anonymized social graph. Let an undirected graph $G_T = \{V_T, E_T\}$ represent the public *target* social network after anonymization. The attacker is assumed to have another undirected graph $G_B = \{V_B, E_B\}$, which models his *background knowledge* about the social relationships among a group of people (i.e., V_B are labeled with the identities of these people). The motivating scenario demonstrates one way to obtain G_B. The attack concerned here is to infer the identities of the vertices V_T by considering structural similarity between G_T and G_B.

I assume that, *before* the release of G_T, the attacker obtain (either by creating or stealing) a few accounts and connect them with a few other users in G_T (e.g., chatting in the motivating scenario). The attacker does not need much effort to do this because these are only basic operations in a social networking service. Besides user ID, the attacker knows nothing about the relationship between other users in G_T. Furthermore, unlike previous works, we *do not assume the attacker has complete control over the connections*; he just *knows* them before G_T's release. This is more realistic. An example is a confirmation-based social network, in which a connection is established only if the two parties confirm it: the attacker *can decline but not impose* a connection.

In contrast to a pure structure-based vertex matching algorithm[15], Seed-and-Grow is a *two-stage* algorithm.

The *seed* stage plants (by obtaining accounts and establishing relationships) a small specially designed subgraph $G_F = \{V_F, E_F\} \subseteq G_T$ (G_F is referred to as the "flag" graph hereafter) into G_T before its release. After the anonymized graph is

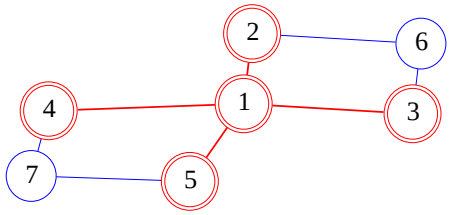


Figure 3.1. A randomly generated graph G_F may be symmetric. Vertices in $G_F = \{v_1, \ldots, v_5\}$ are double-circled.

released, the attacker locates G_F in G_T. The neighboring vertices V_S of G_F in G_T are readily identified and serve as an *initial seed* to be grown.

The *grow* stage is essentially a structure-based vertex matching, which further identifies vertices adjacent to the initial seed V_S. This is a self-reinforcing process, in which the seed grows larger as more vertices are identified.

3.1 Seed

Successful retrieval of G_F in G_T is guaranteed if G_F exhibits the following structural properties.

- G_F is *uniquely* identifiable, i.e., no subgraph $H \subseteq G_T$ except G_F is isomorphic to G_F. For example, in Figure 3.1, subgraph $\{v_1, v_2, v_3\}$ is isomorphic to subgraph $\{v_1, v_4, v_5\}$ because there is a structure-preserving mapping $v_1 \mapsto v_1, v_2 \mapsto v_4, v_3 \mapsto v_5$ between them. Therefore, they are structurally indistinguishable.

- G_F is *asymmetric*, i.e., G_F does not have any non-trivial automorphism. For example, in Figure 3.1, subgraph $\{v_1, v_2, \ldots, v_5\}$ has an automorphism $v_1 \mapsto v_1, v_2 \mapsto v_3, v_3 \mapsto v_4, v_4 \mapsto v_5, v_5 \mapsto v_2$.

In practice, since the structure is unknown to the attacker before its release, the uniquely identifiable property is not realizable. However, as was previously proved[2], with a large enough size and randomly generated edges under the Erdös-Rényi model[16], G_F will be uniquely identifiable with high probability.

Although a randomly generated graph G_F is very likely to be uniquely identifiable in G_T, it may violate the asymmetric structural property. An example is shown in Figure 3.1. If this graph is used as G_F, even if the attacker can uniquely recover it from G_T, he will have a hard time identifying vertices v_2, v_3, v_4, and v_5. Although it was shown that almost all "large enough" randomly generated graphs are asymmetric[17], in practice, the attacker is more likely to generate a relatively small G_F, which demands less effort on his part.

However, because the goal of seed is to identify the initial seed V_S rather than the flag G_F, the asymmetric requirement for G_F can be relaxed. For $u \in V_S$, let $V_F(u)$ be the vertices in V_F which connects with u ($|V_F(u)| \geq 1$ by the definition of V_S). For each pair of vertices, say u and v, in V_S, as long as $V_F(u)$ and $V_F(v)$ are distinguishable in G_F (e.g., $|V_F(u)| \neq |V_F(v)|$ or the degree sequences are different; more precisely, no automorphism of G_F exists which maps $V_F(u)$ to $V_F(v)$), once G_F is recovered from G_T, V_S can be identified uniquely. In Figure 3.1, since $V_F(6)$ and $V_F(7)$ are not distinguishable, vertices v_6 and v_7 can not be identified through G_F.

Based on these observations, I propose the following method for constructing and recovering G_F.

3.1.1 Construction

The construction of G_F starts with a *star* structure (like in Figure 3.1). The motivation for adopting such a structure will be clear in Section 3.1.2. We call the vertex at the center of the star the *head* of G_F and denote it by v_h. In other words, v_h connects to every other vertices in G_F and no others.

The vertices in $V_F - \{v_h\}$ are connected with some other vertices V_S (the initial seed) in G_T, which the attacker has no complete control over (he can only ensure that $V_F(u) \neq V_F(v)$ for any pair of vertices u and v from V_S by declining connections which render indistinguishable vertices in V_S).

As discussed before, the attacker has to ensure that no automorphism of G_F will map $V_F(u)$ to $V_F(v)$. Therefore, he first connects pairs of vertices in $V_F - \{v_h\}$ with a probability of p (in the fashion of Erdös-Rényi model). Then, he collects the *internal degree* $D_F(v)$ for every $v \in V_F - \{v_h\}$ (i.e., v's degree in G_F rather than in G_T; hence *internal* degree) into an *ordered* sequence $\mathcal{S}_D$.

Now, for every $v \in V_S$, v has a corresponding subsequence $\mathcal{S}_D(v)$ of $\mathcal{S}_D$ according to its connectivity with V_F. For example, in Figure 3.1, v_6 connects to v_2 and v_3 from G_F; since $D_F(v_2) = D_F(v_3) = 1$, $\mathcal{S}_D(v_6) = \langle 1, 1 \rangle$. As long as $\mathcal{S}_D(u) \neq \mathcal{S}_D(v)$ for u and v from V_S, no automorphism of G_F will map $V_F(u)$ to $V_F(v)$. Therefore, the attacker guarantees unambiguous recovery of V_S by ensuring that the randomly connected G_F satisfies this condition. If not, the attacker will simply redo the random connection among $V_F - \{v_h\}$ until it does (which eventually will since the attacker can ensure $V_F(u) \neq V_F(v)$ for any pair u and v from V_S by declining connections that will violate this condition). Algorithm 1 summarizes this procedure.

> [Seed Construction] Bob had created 7 accounts v_h and $v_1, \ldots, v_6$, i.e., V_F. He first connected v_h with $v_1, \ldots, v_6$. After awhile, he noticed that users v_7 to v_{10} are connected with $v_1, \ldots, v_6$, i.e., $V_S = \{v_7, \ldots, v_{10}\}$.
>
> Then, he randomly connected $v_1, \ldots, v_6$ and got the resulting graph G_F as shown in Figure 3.2. The ordered internal degree sequence $\mathcal{S}_D = \langle 2, 2, 2, 3, 3, 4 \rangle$.
>
> Bob found $\mathcal{S}_D(v_7) = \langle 2 \rangle$, $\mathcal{S}_D(v_8) = \langle 2, 2 \rangle$, $\mathcal{S}_D(v_9) = \langle 3, 3, 4 \rangle$, and $\mathcal{S}_D(v_{10}) = \langle 2, 3 \rangle$. Since they are mutually distinct, Bob was sure that he could identify v_7 to v_{10} once V_F were found in the published anonymized graph.

The degree of head vertex v_h, the ordered internal degree sequence $\mathcal{S}_D$ and the subsequences chosen for V_S are the *secrets* held by the attacker. As shown in Section 3.1.2, these secrets are used to recover G_F from G_T and thereafter to identify V_S. From the defender's point of view, without knowing the secrets, there is no structure

Algorithm 1 Seed construction.

Create $V_F = \{v_h, v_1, v_2, \ldots\}$.
Given connectivity between V_F and V_S.
Connect v_h with v for all $v \in V_F - \{v_h\}$.
loop
  for all pairs $v_a \neq v_b$ in $V_F - \{v_h\}$ **do**
    Connect v_a and v_b with a probability of p.
  end for
  for all $u \in V_S$ **do**
    Find $\mathcal{S}_D(u)$.
  end for
  if $\mathcal{S}_D(u)$ are mutually distinct for all $u \in V_S$ **then**
    return
  end if
end loop

which characterizes G_F due to the random nature in seed construction. Therefore, G_F is *visible only to the attacker*.

3.1.2 Recovery

Once G_F has been successfully planted and G_T is released, the recovery of G_F from G_T consists of a systematic check of attacker's secrets. The first step is to find a candidate u for the head vertex v_h in G_T by degree comparison. Then, the ordered internal degree sequence of the candidate flag graph (i.e., 1-hop neighborhood of u) and the subsequence secret of candidate initial seed (i.e., exact 2-hop neighborhood of u) are checked. If the candidate flag graph passes these secret checks, it is identified with G_F and its neighbor are identified with V_S by subsequence secret comparison. Algorithm 2 has the detail.

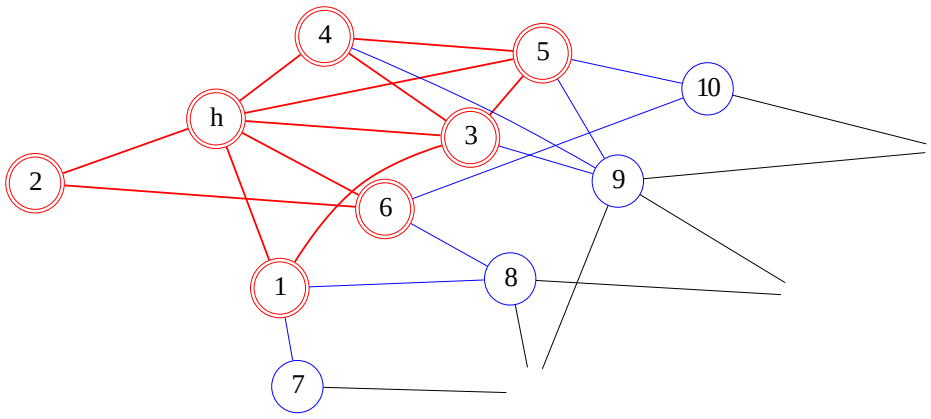


Figure 3.2. An illustration of the seed stage. Vertices in the flag $G_F = \{v_h, v_1, \ldots, v_6\}$ are double-circled. The ordered internal degree sequence $\mathcal{S}_D = \langle 2, 2, 2, 3, 3, 4\rangle$. The internal degree subsequence for the neighboring vertices $V_S = \{v_7, \ldots, v_{10}\}$ of G_F are $\mathcal{S}_D(v_7) = \langle 2\rangle$, $\mathcal{S}_D(v_8) = \langle 2, 2\rangle$, $\mathcal{S}_D(v_9) = \langle 3, 3, 4\rangle$, and $\mathcal{S}_D(v_{10}) = \langle 2, 3\rangle$. Since they are mutually distinct, V_S can be uniquely identified once G_F is recovered.

[Seed Recovery] After the anonymized graph G_T was released, Bob started to check the graph to find the flag. He did this by examining all the vertices in G_T for one with degree 6 (because he knew v_h had degree of 6).

Suppose now, he reached v_h (but he did not know at that moment). He found the vertex had degree of 6. So he isolated it (which he called *candidate head* v_c) along with its 1-hop neighbors (which he called *candidate flag* G_c), and recorded for each of the neighbors the number of connections in G_c (internal degrees). He found that the 1-hop neighbors of v_c had an ordered internal degree sequence $\langle 2, 2, 2, 3, 3, 4\rangle$, which matched with that of V_F. He then proceeded to isolate v_c's exact 2-hop neighbors (which he called *candidate initial seed* V_c) and checked their ordered internal degree subsequences with the candidate flag G_c. He found they again matched with those of V_S.

> Bob was convinced that he had found G_F. By matching the ordered internal degree subsequences of V_c, he identified v_7, v_8, v_9 and v_{10}. For example, for a 2-hop neighbor $u \in V_c$ which connected with three 1-hop neighbors with internal degrees 3, 3 and 4, he identified u with v_9.

The motivation for incorporating the head vertex technique in the seed construction stage is clear now. The only connections v_h has are *internal* ones. Therefore, once a candidate head vertex u is found, the candidate flag can be readily determined by reading off the 1-hop neighborhood of u. Thereafter, no probing or backtracking is needed for finding G_F like in previous works[1, 2].

The efficiency of the algorithm is evident by observing that, in Algorithm 2, the maximal level of nested loops is 3 (2 of them are on a vertex's neighborhood) and no recursion is involved. Because the 2-hop neighborhood of u_v (e.g., $V_F \cup V_S$) are controlled by the attacker (as secrets), if the size (i.e., the number of vertices) of the 2-hop neighborhood is N, the complexity of the recovery algorithm is $O(N|V_T|)$.

3.2 Grow

The initial seed provides a firm ground for further identification in the anonymized graph G_T. Background knowledge G_B comes into play at this stage.

At this stage, there is a partial mapping between G_T and G_B, i.e., the initial seed V_S in G_T maps to its corresponding identities in G_B. Two examples of partial graph mappings are the Twitter and Flickr datasets[1] and the Netflix and IMDB datasets[18]. The straightforward idea of testing all possible mappings for the rest of the vertices has an exponential complexity, which is unacceptable even for a medium-sized network. Beside, the overlapping between G_T and G_B may well be *partial* (e.g., $|V_T| \neq |V_B|$), so a *full* mapping is either impossible or undesirable. Therefore, the grow algorithm adopts a progressive and self-reinforcing strategy, mapping multiple vertices at a time.

Algorithm 2 Seed recovery.

1: **for all** $u \in G_T$ **do**
2: **if** $\deg(u) = |V_F| - 1$ **then**
3: $U \leftarrow$ exact 1-hop neighborhood of u
4: **for all** $v \in U$ **do**
5: $d(v) \leftarrow$ number of v's neighbors in $U \cup \{u\}$
6: **end for**
7: $s(u) \leftarrow \text{sort}(d(v)|v \in U)$
8: **if** $s(u) = \mathcal{S}_D$ **then**
9: $V \leftarrow$ exact 2-hop neighborhood of u
10: **for all** $w \in V$ **do**
11: $U(w) \leftarrow w$'s neighbors in U
12: $s(w) \leftarrow \text{sort}(d(v)|v \in U(w))$
13: **end for**
14: **if** $\langle s(w)|w \in V\rangle = \langle \mathcal{S}_D(v)|v \in V_S\rangle$ **then**
15: $\{w \in V$ is identified with $v \in V_S$ if $s(w) = \mathcal{S}_D(v)\}$
16: **end if**
17: **end if**
18: **end if**
19: **end for**

Figure 3.3 shows a small example. v_7 to v_{10} have already been identified in the seed stage (recall Figure 3.2). The task is to identify other vertices in the target graph G_T.

The grow algorithm centers around a pair of *dissimilarity* metrics between a pair of vertices from the target and the background graph respectively. In order to enhance the identification accuracy and to reduce the computation complexity and the false-positive rate, I introduce a *greedy heuristic* with *revisiting* into the algorithm. These details are examined below.

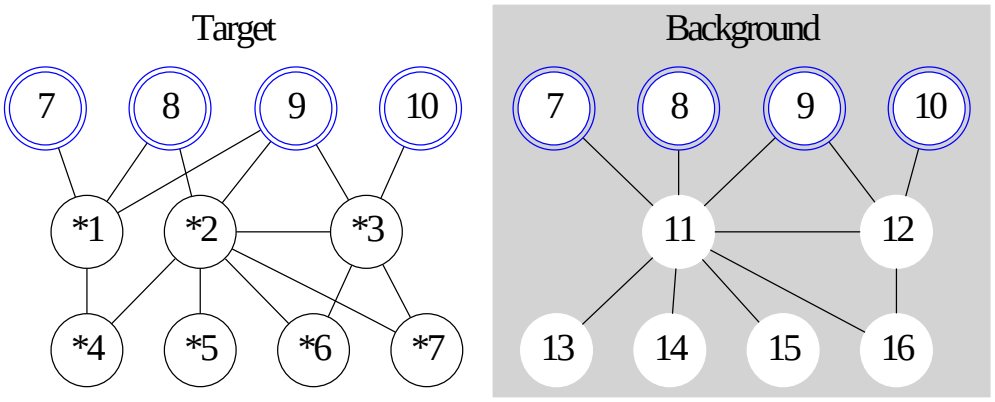


Figure 3.3. An illustration of the grow stage. Vertices in the initial seed $V_S = \{v_7, \ldots, v_{10}\}$ are double-circled. Those vertices in the target graph G_T with labels starting with an asterisk are yet to be identified. The task of the grow stage is to identify these vertices.

3.2.1 Dissimilarity

It is natural to start with those vertices in G_T which connect to the initial seed V_S because they are more close to the *certain* information, i.e., the already identified vertices V_S. For these vertices, their neighboring vertices can be divided into two groups. Namely, for such a vertex u, its neighborhood in G_T is composed of $\mathcal{N}_m^T(u)$ (*mapped* neighbors) and $\mathcal{N}_u^T(u)$ (*unmapped* neighbors). For instance, in Figure 3.3, $\mathcal{N}_m^T(u_{*1}) = \{u_7, u_8, u_9\}$ and $\mathcal{N}_u^T(u_{*1}) = \{u_{*4}\}$.

Similar definitions can be made for the background graph G_B. Suppose the seed $V_S \subseteq V_T$ maps to $V_S^* \subseteq V_B$. For a V_S^*'s neighboring vertex v, let $\mathcal{N}_m^B(v)$ be v's neighbors in V_S^* and $\mathcal{N}_u^B(v)$ be the other (i.e., unmapped) neighbors. Hence, in Figure 3.3, $\mathcal{N}_m^B(v_{12}) = \{v_9, v_{10}\}$ and $\mathcal{N}_u^B(v_{12}) = \{v_{11}, v_{16}\}$.

The mapped vertices in V_S and V_S^* are identified so that $\mathcal{N}_m^T(u_{*1}) - \mathcal{N}_m^B(v_{12}) = \{u_7, u_8\} = \{v_7, v_8\}$ in Figure 3.3.

For a pair of nodes, $u \in V_T$ and $v \in V_B$, the following definitions are made.

$$\Delta_T(u, v) = \frac{|\mathcal{N}_m^T(u) - \mathcal{N}_m^B(v)|}{|\mathcal{N}_m^T(u)|}, \tag{3.1}$$

Table 3.1
Dissimilarity metrics for pairs of unmapped vertices in Figure 3.3. Each tuple consists of a (Δ_T, Δ_B) pair.

Δ	u_{*1}	u_{*2}	u_{*3}
v_{11}	(0.00, 0.00)	(0.00, 0.33)	(0.50, 0.67)
v_{12}	(0.67, 0.50)	(0.50, 0.50)	(0.00, 0.00)

and

$$\Delta_B(u, v) = \frac{|\mathcal{N}_m^B(v) - \mathcal{N}_m^T(u)|}{|\mathcal{N}_m^B(v)|}, \tag{3.2}$$

in which $|\cdot|$ is the number of set elements, i.e., set cardinality. I call these metrics *dissimilarity*. In Figure 3.3, $\Delta_T(u_{*1}, v_{12}) = |\{u_7, u_8\}|/|\{u_7, u_8, u_9\}| = 2/3 \approx 0.667$ and $\Delta_B(u_{*1}, v_{12}) = |\{v_{10}\}|/|\{v_9, v_{10}\}| = 1/2 = 0.5$.

$\Delta_T(u, v)$ and $\Delta_B(u, v)$ together measure how different u and v's mapped neighborhoods are. By its definition in Equations 3.1 and 3.2, both $\Delta_T(u, v)$ and $\Delta_B(u, v)$ are in the range of $[0, 1]$. More precisely, when their mapped neighborhoods are the same ($\mathcal{N}_m^T(u) = \mathcal{N}_m^B(v)$), we have $\Delta_T(u, v) = \Delta_B(u, v) = 0$, which means u and v match perfectly in regard to their mapped neighborhoods. Otherwise, when $\mathcal{N}_m^T(u) \cap \mathcal{N}_m^B(v) = \emptyset$, $\Delta_T(u, v) = \Delta_B(u, v) = 1$. The reason to have two asymmetric metrics (in regard to the target and background graphs) instead of a symmetric one is that I want to choose those mappings which are the mutually best choices for the graphs.

Again, a concrete example helps.

> [Dissimilarity] Bob applied the dissimilarity metrics defined in Equations 3.1 and 3.2 to Figure 3.3 and got the results shown in Table 3.1.
>
> Bob first identified the tuples in Table 3.1 which has the smallest Δ_T and Δ_B in both its row and column. In this case, these tuples are (u_{*1}, v_{11}) and (u_{*3}, v_{12}). Since they are *from different rows and columns*, they do

not conflict with each other. So Bob decided to map u_{*1} to v_{11} and u_{*3} to v_{12}.

He then added $v_{*1} \leftrightarrow v_{11}$ and $v_{*3} \leftrightarrow v_{12}$ to the seed and moved on to the next iteration of identification.

3.2.2 Greedy Heuristic

Bob's story suggests a way of using the dissimilarity metrics defined in Equations 3.1 and 3.2 to iteratively grow the seed. In each iteration, the neighboring vertices of the seed in V_T and V_B are mixed and matched and for each pair, say $u \in V_T$ and $v \in V_B$, $\Delta_T(u, v)$ and $\Delta_B(u, v)$ are computed; the results are collected into a table like Table 3.1.

Since smaller dissimilarity implies better match, I identify those tuples in the table which has *smallest* Δ_T and Δ_B in both its row and column; these tuples are the mutually best matches from/to the target graph to/from the background graph. The mappings corresponding to these tuples are added to the seed and move on to the next iteration.

I gloss over a subtlety in the above description: if there are *conflicts* in choice, i.e., there are more than one tuples satisfying the above criterion in a row or a column, which one shall I choose? Rather than randomly selecting a tuple, I select the tuple that *stands out* and add the corresponding match to the seed. If there is still a tie, these tuples are reckoned as indistinguishable under the dissimilarity metrics. To reduce incorrect identifications, I refrain from randomly picking one and adding the mapping to the seed in these scenarios.

This boils down to the question of how to quantify the concept of "a tuple standing out among its peers". Since each tuple consists of two numbers in the range of $[0, 1]$, the question translates to that of quantifying the concept of "a number standing out among a group of numbers". I define an *eccentricity* metric for this purpose. Let X be

a multi-set of numbers (the same number can occur multiple times). The *eccentricity* of a number $x \in X$ is defined as

$$\mathcal{E}_X(x) = \begin{cases} \frac{\Delta_X(x)}{\sigma(X)\#_X(x)} & \text{if} \quad \sigma(X) \neq 0 \\ 0 & \text{if} \quad \sigma(X) = 0 \end{cases}. \tag{3.3}$$

in which $\Delta_X(x)$ is the absolute difference between x and its closest *different* value in X; $\#_X(x)$ is the *multitude* of x in X, i.e., the number of elements equal to x in X; $\sigma(X)$ is the standard deviation of X. The larger $\mathcal{E}_X(x)$ is, the more x stands out among X. The role of $\#_X(x)$ becomes evident by considering the eccentricity of 1 in $(0, 0, 1)$ and $(0, 1, 1)$, in which $\Delta_X(x)$ and $\sigma(X)$ are the same but 1 arguably stands out more prominently in $(0, 0, 1)$ than in $(0, 1, 1)$.

Therefore, if there are conflicts *in a row*, these tuples have the same Δ_T and Δ_B. For each such tuple, Δ_T and Δ_B *in the same column* are collected into X_T and X_B respectively and compute $\mathcal{E}_{X_T}(\Delta_T)$ and $\mathcal{E}_{X_B}(\Delta_B)$. If there is a *unique* tuple with *largest* $\mathcal{E}_{X_T}(\Delta_T)$ and $\mathcal{E}_{X_B}(\Delta_B)$, the corresponding mapping is added to the seed; otherwise, no mapping is added to the seed.

3.2.3 Revisiting

The dissimilarity metric and the greedy search algorithm for optimal combination are heuristic in nature. At an early stage with only a few seeds, there might be quite a few mapping candidates for a particular vertex in the background graph; it is likely to pick a wrong mapping no matter which strategy is used in resolving the ambiguity. If left uncorrected, the incorrect mappings will propagate through the grow process and lead to large-scale mismatch.

I address this problem by reexamining previous mapping decisions given new evidences in the grow algorithm; I call this *revisiting*. More concretely, for each iteration, all vertices which have at least one seed neighbor are considered; these are the vertices on which the dissimilarity metrics in Equations 3.1 and 3.2 are well-defined.

I observed in experiments that, for a few initial seeds, the revisiting technique as presented above led to the following situation: on iteration i, the algorithm chose the vertices $T_i \subset V_T$ and $B_i \subset B_T$, from target and background graphs respectively, as mapping candidates; on iteration $i+1$, the algorithm chose $T_{i+1} \subset V_T$ and $B_{i+1} \subset V_B$; on iteration $i+2$, the algorithm chose $V_{i+2} = V_i$ and $B_{i+2} = B_i$ again; the algorithm stuck in these two cases and never finished. I address this problem by recording all mapping candidate pairs and stop as soon as a mapping candidate pair occurs twice. In the scenarios mentioned earlier, the algorithm will stop at iteration $i+2$ and output as result the seed produced by iteration $i+1$.

As shown in Section 4.3.3, the revisiting technique increases the accuracy of the algorithm. The greedy heuristic with revisiting is summarized in Algorithm 3.

Algorithm 3 Grow.

Given the initial seed V_S.
$C = \emptyset$
loop
  $C_T \leftarrow \{u \in V_T | u \text{ connects to } V_S\}$
  $C_B \leftarrow \{v \in V_B | v \text{ connects to } V_S\}$
  if $(C_T, C_B) \in C$ **then**
    return V_S
  end if
  $C \leftarrow C \cup \{(C_T, C_B)\}$
  for all $(u, v) \in (C_T, C_B)$ **do**
    Compute $\Delta_T(u, v)$ and $\Delta_B(u, v)$.
  end for
  $S \leftarrow \{(u, v) | \Delta_T(u, v) \text{ and } \Delta_B(u, v) \text{ are smallest among conflicts}\}$
  for all $(u, v) \in S$ **do**
    if (u, v) has no conflict in S *or* (u, v) has the uniquely largest eccentricity among conflicts in S **then**
      $V_S \leftarrow V_S \cup \{(u, v)\}$
    end if
  end for
end loop

4 EXPERIMENTS

I conducted a comparative study on the performance of the Seed-and-Grow algorithm by simulation on real-world social network datasets.

4.1 Setup

I used two datasets collected from different real-world social networks in this study.

The `Livejournal` dataset, which was collected from the friend relationship of the on-line journal service LiveJournal on 9–11 December 2006 and kindly provided to the research community by Mislove et al.[19], consists of 5.2 million vertices and 72 million links. The links are directed. I conducted the experiments on the more difficult setting of undirected graph: an undirected link between two vertices was retained if and only if there was a directed link in *either* direction.

The other dataset, `emailWeek`[1], consists of 200 vertices and $1,676$ links. This dataset, by its nature, is undirected.

Using datasets collected from different underlying social networks helped to reduce bias induced by the idiosyncrasy of a particular network in performance measurements.

The performance of the grow algorithm was measured by its ability to identify the anonymous vertices in the target graph. I derived the target and background graphs from each dataset and used their shared vertices as the *ground truth* to measure against.

[1]The dataset and its visualization are publicly available at `http://www.infovis-wiki.net/index.php/Social_Network_Generation`.

More precisely, I derived the graphs with the following procedure. First, I chose a connected subgraph with $N_\cap$ vertices from the dataset, which served as *shared portion* of the background and target graphs. I then picked other two sets of vertices (totally different from the previous $N_\cap$ vertices) with $N_B - N_\cap$ and $N_T - N_\cap$ vertices, respectively, and combined with shared portion graph to obtain the background graph (with N_B vertices) and the target graph (with N_T vertices). After this, N_S ($N_S < N_\cap$ and not necessarily connected) vertices were chosen from the shared portion to serve as the initial seed. Finally, random edges were added to the target graph to simulate the difference between the target and background graphs.

The motivation for adopting such a procedure was to simulate a more realistic scenario. The attacker had a (connected) background graph with N_B vertices and an anonymous target graph with N_T vertices. Apart from the N_S initial seed, the overlap of these two graphs (with $N_\cap$ vertices) might well be partial. The desirable behavior of an identification algorithm was to stop as soon as the vertices in the shared portion had been identified. Since the background graph was an unperturbed graph the attacker obtained from elsewhere, I opted to perturb the target graph to simulate the difference between these two graphs. I perturbed by addition rather than deletion of edges to avoid fragmenting the target graph into disconnected pieces, which would create a false impression of early stopping in simulation.

4.2 Seed

The Seed construction (Algorithm 1) and recovery (Algorithm 2) algorithms ensure that, once the flag graph G_F is successfully recovered, the initial seed V_S can be unambiguously identified. Therefore, the seed construction depends on G_F being uniquely recovered from the released target graph.

In the experiment, a number of modest-sized flag graphs with 10 to 20 vertices are generated and planted into the `Livejournal` dataset with Algorithm 1. I was

able to uniquely recover them from the resulted graph with Algorithm 2 without an exception.

To explain this result, I made the following estimation on the number of essentially different (i.e., with different ordered internal degree sequence $\mathcal{S}_D$) constructions produced by Algorithm 1.

For a flag graph G_F with n vertices, there are $n-1$ vertices beside the head node v_h. By Algorithm 1, v_h is connected with each of the $n-1$ vertices. So we only need to consider the connections between the $n-1$ non-head vertices to estimate the number of constructions.

There are $(n-1)(n-2)/2$ pairs among the $n-1$ vertices; the edge between each pair of vertices can be either present or absent. Therefore, there are $2^{(n-1)(n-2)/2}$ different flag graphs.

However, some of them are considered the same by Algorithm 1. For example, the ordered internal degree sequence $\mathcal{S}_D = \langle 2,2,2,3,3,4\rangle$ in Figure 3.2. There are 3, 2, and 1 vertices with an internal degree of 2, 3, and 4, respectively; hence, there are $\binom{6}{3}\binom{3}{2}\binom{1}{1}$ different flag graphs with the same ordered internal degrees sequence.

For any ordered internal degree sequence $\mathcal{S}_D$, there are *at most*

$$\binom{n-1}{1}\binom{n-2}{1}\cdots\binom{2}{1}\binom{1}{1} = (n-1)!$$

flag graphs with n vertices. The ordered internal degree sequence divides all flag graphs into equivalent classes. Therefore, there are at least

$$\frac{2^{(n-1)(n-2)/2}}{(n-1)!}.$$

essentially different constructions produced by Algorithm 1.

Figure 4.1 shows this estimate for a few different flag graph sizes. From this, we can understand the reason for the high probability for successful flag graph recovery, even in a large graph like `Livejournal` with 5.2×10^6 vertices: there are so many ways to construct essentially different flag graphs.

Table 4.1
The estimate of essentially different constructions for a flag graph G_F with n vertices produced by Algorithm 1.

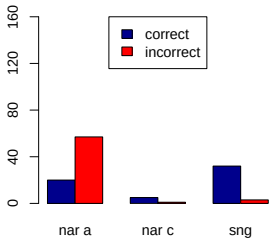
n	10	11	12	13
estimate	1.89×10^6	9.70×10^7	9.03×10^8	1.54×10^{11}

4.3 Grow

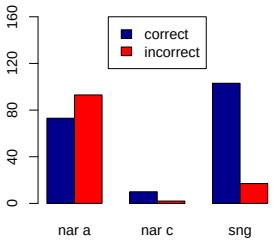
The grow algorithm is compared with the one proposed by Narayanan and Shmatikov[1]. There is a mandatory threshold parameter, which controls the probing aggressiveness, in their algorithm. Lacking a quantitative guideline to choose this parameter[1], I experimented with different values and found that, with increasing threshold, more nodes were identified but the accuracy decreased accordingly. Therefore, I used two different thresholds, which established a performance envelop for the Narayanan algorithm. The result was two variants of the algorithm: an aggressive one (with a threshold of 0.0001) and a conservative one (with a threshold of 1). The difference of the two variants lies in the tolerance to the ambiguities in matching: the aggressive variant might produce a mapping in a case whereas the conservative variant would deem too ambiguous.

I perceive such an arbitrary parameter, lacking a quantitative guideline, as a major drawback of the Narayanan algorithm: a user of the algorithm *must* decide on the parameter without knowing how much accuracy is sacrificed for better effectiveness (the number of identified nodes). In contrast, the Seed-and-Grow algorithm has no such parameter and, as demonstrated by the experiments, finds a good balance between effectiveness and accuracy.

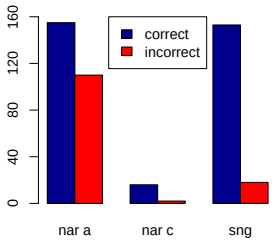
To account for the bias on the performance measurement of a particular graph setting, for each target/background graph pair, multiple runs of simulations were conducted with different initial seeds; the average was taken as the performance result. I focused the simulations on graphs with hundreds of vertices, which were big enough



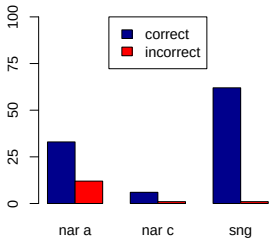
(a) `Livejournal`, seed: 5, pertb: 0.5%



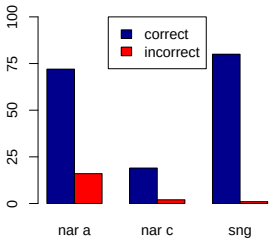
(b) `Livejournal`, seed: 10, pertb: 0.5%



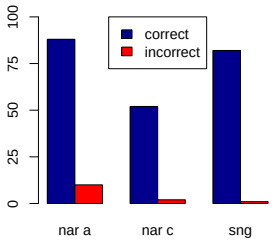
(c) `Livejournal`, seed: 15, pertb: 0.5%



(d) `emailWeek`, seed: 5, pertb: 0.5%



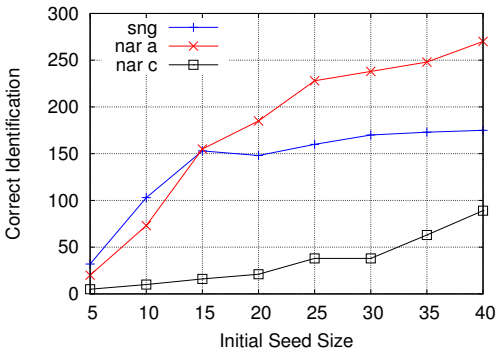
(e) `emailWeek`, seed: 10, pertb: 0.5%



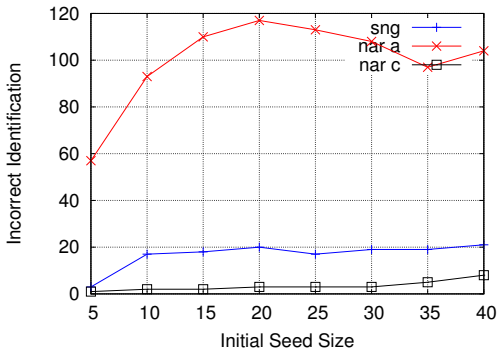
(f) `emailWeek`, seed: 15, pertb: 0.5%

Figure 4.1. Grow performance with different initial seed sizes. The Seed-and-Grow (sng) algorithm is compared with two variants of the identification algorithm proposed by Narayanan and Shmatikov[1]: "aggressive" (nar a; with a threshold of 0.0001) and "conservative" (nar c; with a threshold of 1). An edge perturbation of 0.5% is introduced to simulate a more realistic scenario. (a), (b), and (c) are from `Livejournal`; (d), (e), and (f) are from `emailWeek`.

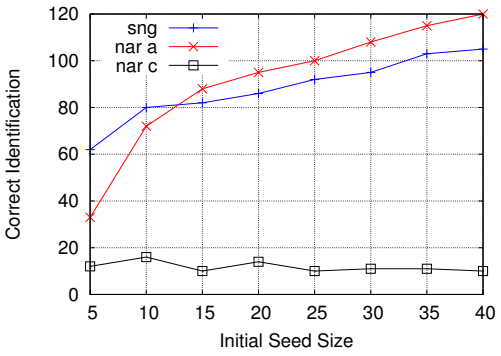
to make the identification non-trivial. More precisely, I chose ($N_C = 400 + N_S, N_T = 600 + N_S, N_B = 600 + N_S$) for `Livejournal` and ($N_C = 100 + N_S, N_T = 125 + N_S, N_B = 125 + N_S$) for `emailWeek`. In other words, the ideal result is to correctly identify $400 + N_S$ nodes for `Livejournal` and $100 + N_S$ nodes for `emailWeek`, where N_S is the size of initial seed.



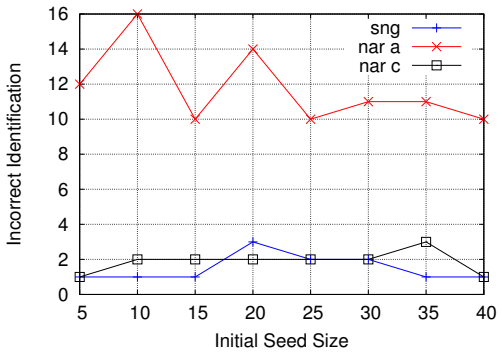
(a) `Livejournal`, correct identifications.



(b) `Livejournal`, incorrect identifications.



(c) `emailWeek`, correct identifications.

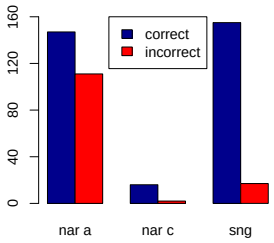


(d) `emailWeek`, incorrect identifications.

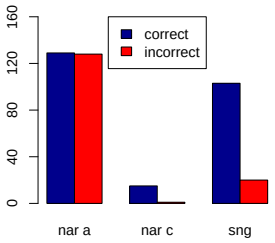
Figure 4.2. Grow performance with different initial seed sizes on a larger scale than Figure 4.1. The Seed-and-Grow (sng) algorithm is compared with two variants of the identification algorithm proposed by Narayanan and Shmatikov[1]: "aggressive" (nar a; with a threshold of 0.0001) and "conservative" (nar c; with a threshold of 1). An edge perturbation of 0.5% is introduced to simulate a more realistic scenario. (a) and (b) are from `Livejournal`; (c) and (d) are from `emailWeek`.

4.3.1 Initial Seed Size

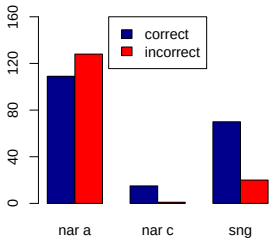
Recent literature[20] on interaction-based social graph (e.g., the social graph in the motivating scenario) singles out attacker's interaction budget as the major limitation to attack effectiveness. The limitation translates to 1) the initial seed size and 2) number of links between the flag graph and initial seed. The seed algorithm resolves the latter issue by guaranteeing unambiguous identification of initial seed regardless



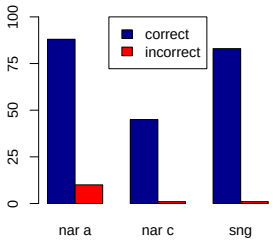
(a) `Livejournal`, seed: 15, pertb: 0.5%



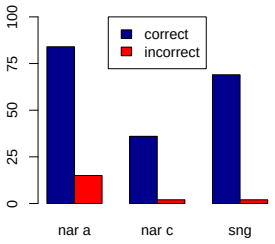
(b) `Livejournal`, seed: 15, pertb: 1%



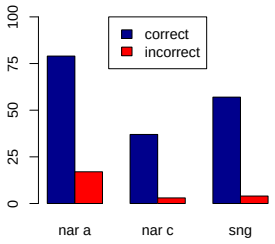
(c) `Livejournal`, seed: 15, pertb: 1.5%



(d) `emailWeek`, seed: 15, pertb: 0.5%



(e) `emailWeek`, seed: 15, pertb: 1%

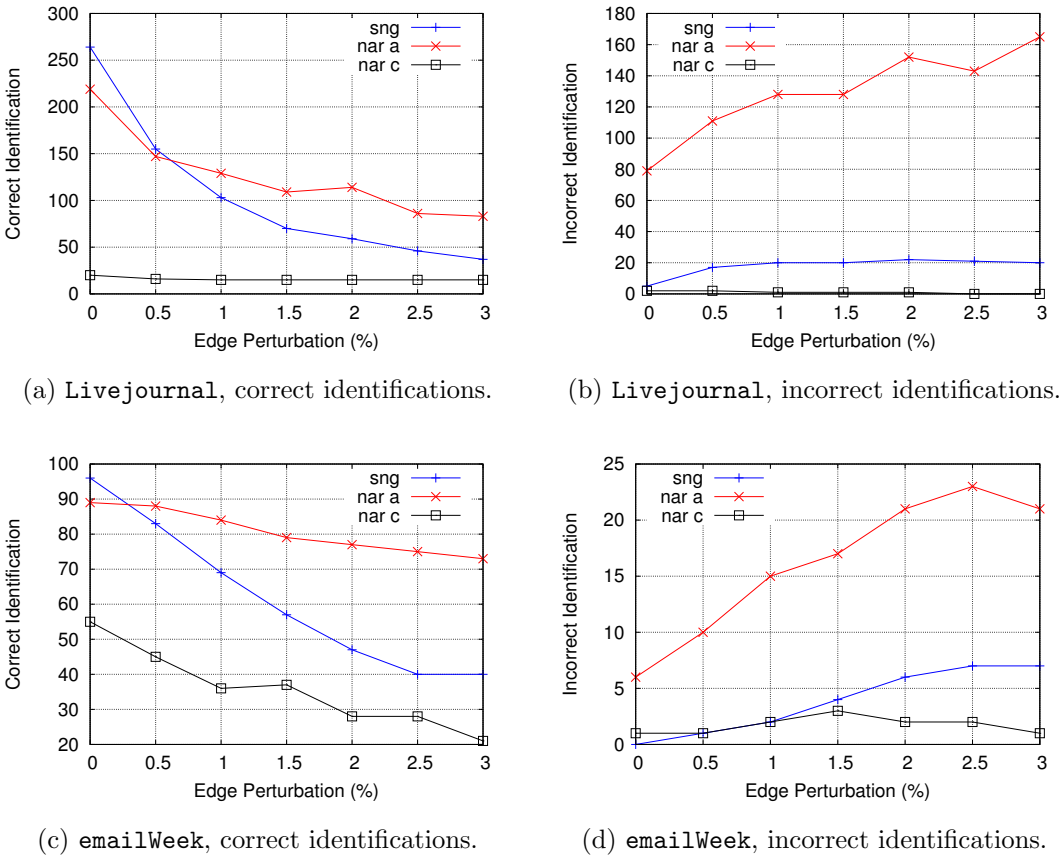


(f) `emailWeek`, seed: 15, pertb: 1.5%

Figure 4.3. Grow performance with different edge perturbation percentage. The Seed-and-Grow (sng) algorithm is compared with two variants of the identification algorithm proposed by Narayanan and Shmatikov[1]: "aggressive" (nar a; with a threshold of 0.0001) and "conservative" (nar c; with a threshold of 1). The initial seed size is 15 for both datasets. (a), (b), and (c) are from `Livejournal`; (d), (e), and (f) are from `emailWeek`.

of link numbers. As shown below, the grow algorithm resolves the former issue by working well with a small initial seed.

Figure 4.1 shows the grow performance with different initial seed sizes. To simulate the more realistic case that the target and background graphs are from different sources and therefore might be differ even among the same group of vertices, I in-



(a) `Livejournal`, correct identifications.

(b) `Livejournal`, incorrect identifications.

(c) `emailWeek`, correct identifications.

(d) `emailWeek`, incorrect identifications.

Figure 4.4. Grow performance with different edge perturbation percentage on a larger scale than Figure 4.3. The Seed-and-Grow (sng) algorithm is compared with two variants of the identification algorithm proposed by Narayanan and Shmatikov[1]: "aggressive" (nar a; with a threshold of 0.0001) and "conservative" (nar c; with a threshold of 1). The initial seed size is 15 for both datasets. (a) and (b) are from `Livejournal`; (c) and (d) are from `emailWeek`.

troduced an *edge perturbation* of 0.5%, i.e., I added 0.5% of the all the edges in the target graph's complement to the target graph.

A few points are noted in Figure 4.1.

- More nodes are correctly identified with increasing initial seed size for both Seed-and-Grow and Narayanan.

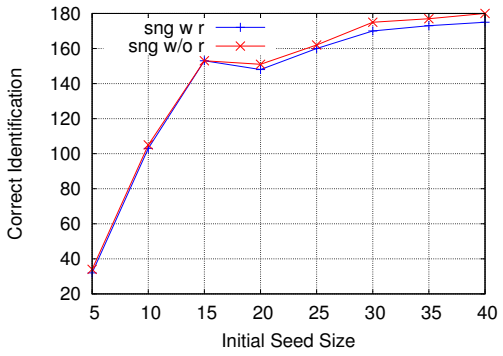
- Seed-and-Grow is better (or at least comparable) to the aggressive Narayanan in terms of number of correct identifications, and is superior when comparing with conservative Narayanan. For `Livejournal`, conservative Narayanan stops almost immediately (the correct identification statistics shown in Figure 4.1 includes the initial seed). In contrast, even for very small initial seed of 5 nodes, Seed-and-Grow correctly identifies averagely 32 nodes for `Livejournal` and 62 nodes for `emailWeek` while only incorrectly identifies 1 nodes on average.

- Though aggressive Narayanan correctly identifies more nodes as seed size grow, the number of incorrect identification grows accordingly. This is especially evident in `Livejournal`. In contrast, the incorrect identification number for Seed-and-Grow remains constant in `emailWeek` and grows very slowly in `Livejournal`; in either case, the percentage of correct identification, as defined by the number of correct identifications over the total number of identified nodes, is much higher for Seed-and-Grow than for aggressive Narayanan.

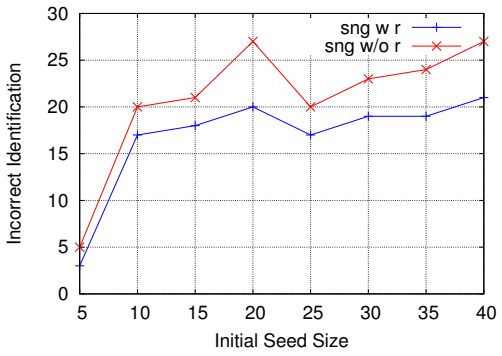
An ideal grow algorithm should be both effective and accurate. Effectiveness is measured by the number of correct identification; accuracy is measured by the percentage of correct identification. Figure 4.1 shows Seed-and-Grow is 1) comparable to aggressive Narayanan in terms of effectiveness while better in terms of accuracy; 2) comparable to conservative Narayanan in terms of accuracy while better in terms of effectiveness.

Figure 4.2 shows, on a larger scale, the comparison in Figure 4.1 in a slightly different form. The previous observations on algorithm performance hold for larger seeds.

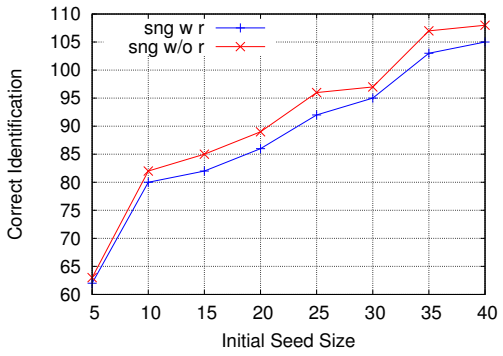
It is arguable that, with a "proper" threshold, Narayanan will show the same or even superior performance than Seed-and-Grow. However, lacking any quantitative guideline, such a proper threshold is hard, if not impossible, to find for the vast array of graphs the identification algorithm applies to. Even one can find such a threshold, it is unclear that its performance will be superior to that of Seed-and-Grow. In contrast,



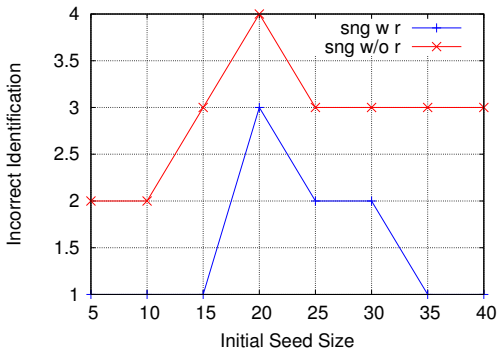
(a) `Livejournal`, correct identifications.



(b) `Livejournal`, incorrect identifications.



(c) `emailWeek`, correct identifications.



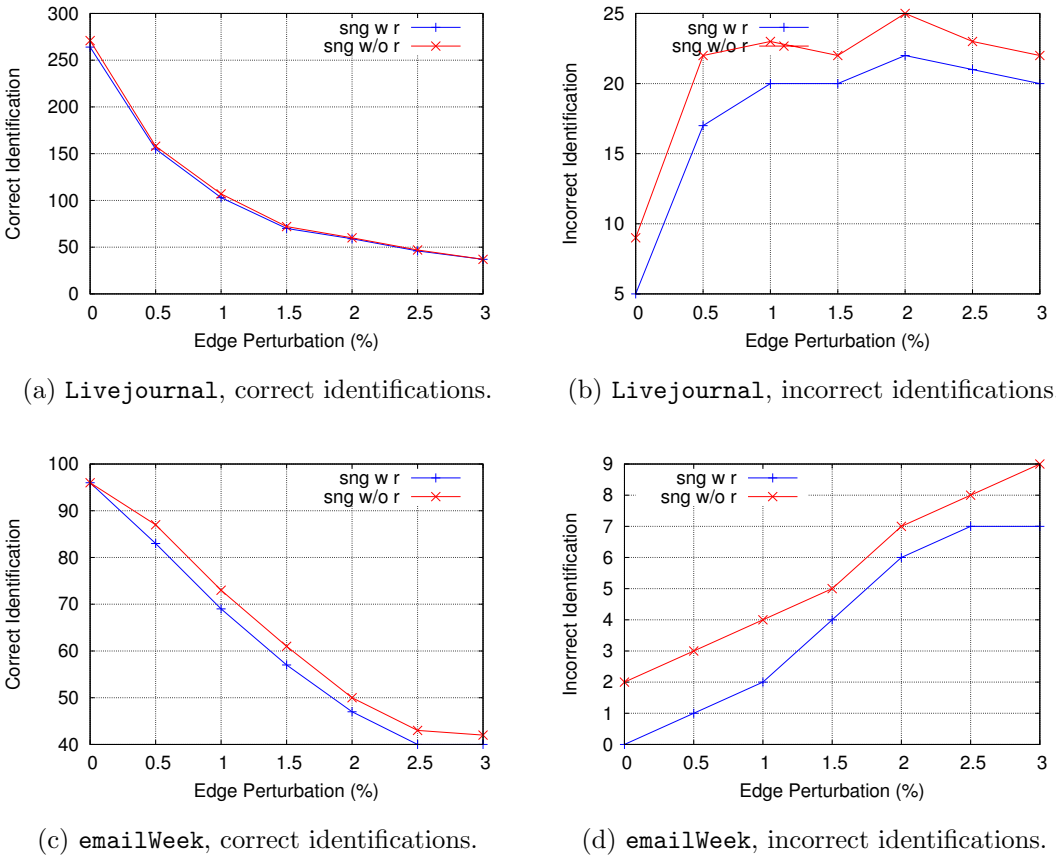
(d) `emailWeek`, incorrect identifications.

Figure 4.5. Grow performance with different initial seed sizes. Seed-and-Grow algorithm is compared with (sng w r) and without (sng w/o r) revisiting. An edge perturbation of 0.5% is introduced to simulate a more realistic scenario. (a) and (b) are from `Livejournal`; (c) and (d) are from `emailWeek`.

Seed-and-Grow has no such arbitrary parameter. The point is that Seed-and-Grow *automatically* finds a sensible balance between effectiveness and accuracy.

4.3.2 Edge Perturbation

The impact of edge perturbations on the grow performance is shown in Figure 4.3. The initial seed size was 15.



(a) `Livejournal`, correct identifications.

(b) `Livejournal`, incorrect identifications.

(c) `emailWeek`, correct identifications.

(d) `emailWeek`, incorrect identifications.

Figure 4.6. Grow performance with different edge perturbation percentage. Seed-and-Grow algorithm is compared with (sng w r) and without (sng w/o r) revisiting. The initial seed size is 15 for both datasets. (a) and (b) are from `Livejournal`; (c) and (d) are from `emailWeek`.

Correct identifications decreased with a larger edge perturbation percentage for all algorithms. Incorrect identifications increased with edge perturbation for aggressive Narayanan while remaining at a constant level for Seed-and-Grow and conservative Narayanan.

Seed-and-Grow is more effective than conservative Narayanan in all settings. Although aggressive Narayanan is more effective than Seed-and-Grow for larger perturbation percentage, it comes with a price of much lower accuracy: for `Livejournal`, aggressive Narayanan made more incorrect identifications than correct ones. In con-

trast, the number of incorrect identification for Seed-and-Grow remained almost constant for different perturbation percentage.

A high accuracy (i.e., high percentage of correct identification) is desirable, even at a reasonable cost of effectiveness (fewer nodes identified). This is because, lacking the knowledge whether an identification is correct, accuracy corresponds to the user's *confidence* in the identification result. For example, in Figure 4.3c, even though aggressive Narayanan correctly identifies 109 nodes on average while Seed-and-Grow only correctly identifies 70 nodes on average, the former incorrectly identifies 128 nodes on average while the latter only incorrectly identifies 20 nodes on average. Without knowing which nodes are correctly identified, a user has less than 50% confidence in the results of aggressive Narayanan while having more than 75% confidence in the results of Seed-and-Grow.

Figure 4.4 shows, on a larger scale, the comparison in Figure 4.3 in a slightly different form. The discussions on algorithm performance hold for larger edge perturbation percentages.

On reflection, I attribute the relatively high accuracy of Seed-and-Grow to its conservative design in the grow algorithm (Algorithm 3). More specifically, a mapping is added to the seed (i.e., grow the seed) if and only if 1) it is the mutually best choice for the pair of nodes under the dissimilarity metric and 2) it stands out among alternative choices in the sense that it has no tie under the eccentricity metric. Besides, the algorithm further improves accuracy by revisiting earlier mappings in light of new mappings.

4.3.3 Revisiting

To verify the expectation that the revisiting technique improves performance, I ran simulations of the Seed-and-Grow algorithm *without* revisiting , and compared it with the full version *with* revisiting. The results are shown in Figures 4.5 and 4.6.

The results indicate that revisiting reduces incorrect identifications for both datasets while, in the case of `emailWeek`, also reduces correct identifications by around the same number. Because Seed-and-Grow has the desirable property that correct identifications far outnumber incorrect ones, proportionally speaking, the increase in accuracy far outweighs the sacrifice in effectiveness. This confirms the conjecture that it is worthwhile to revisit previous mappings.

5 CONCLUSION

With the increasing popularity of on-line social network services, digital traces left by a user of such services might be collected and abused by a malicious party to compromise that user's privacy.

To demonstrate the feasibility of abuse and raise public awareness on this issue, I propose an algorithm, *Seed-and-Grow*, to identify users from an anonymized social graph. Seed-and-Grow exploits the increasing overlapping user-bases among services and is based solely on social graph structure. The algorithm first identifies a seed sub-graph either planted by an attacker or divulged by collusion of a small group of users, and then grows the seed larger based on the attackers existing knowledge of the users social relations. I identify and relax implicit assumptions for unambiguous seed identification taken by previous works, eliminate arbitrary parameters in grow algorithm, and demonstrate the superior performance over previous works in terms of identification effectiveness and accuracy by simulation on real-world-collected social-network datasets.

As an extension to the current work, I plan to incorporate tagging (known mappings from external source) into Seed-and-Grow. Although Seed-and-Grow is based solely on graph structure, with auxiliary information often abounded in real-world dataset, I expect tagging will further increase identification effectiveness and accuracy.

Another potential extension is to adapt the grow algorithm to work with multiple seeds. A challenge is to resolve conflicting grow result from different seeds. Another challenge is to design a multiple-seed grow algorithm with a super-linear (with regard

to seed number) effectiveness (i.e., high number of correct identification) without sacrificing accuracy (i.e., low number of incorrect identifications).

LIST OF REFERENCES

LIST OF REFERENCES

[1] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Proc. of IEEE Symposium on Security and Privacy*, 2009.

[2] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proc. of ACM International Conference on World Wide Web (WWW)*, 2007.

[3] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava. Anonymizing social networks. Technical report, University of Massachusetts, Amherst, 2007.

[4] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *Proc. of ACM SIGKDD International Conference on Privacy, Security, and Trust in KDD*, 2007.

[5] A. Korolova, R. Motwani, S.U. Nabar, and Y. Xu. Link privacy in social networks. In *Proc. of ACM Conference on Information and Knowledge Management*, 2008.

[6] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, 2008.

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

[8] B. Zhou, J. Pei, and W.S. Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM SIGKDD Explorations Newsletter*, 10(2):12–22, 2008.

[9] J. Scott. *Social network analysis: a handbook*. SAGE Publications, 2000.

[10] S. Wasserman. *Social network analysis: Methods and applications*. Cambridge university press, 1994.

[11] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *Proc. of ACM SIGMOD International Conference on Management of Data*, 2005.

[12] D. Agrawal and C.C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proc. of ACM SIGMOD*, 2001.

[13] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proc.of ACM SIGMOD*. ACM, 2003.

[14] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. *Link Mining: Models, Algorithms, and Applications*, pages 337–357, 2010.

[15] S. Sorlin and C. Solnon. Reactive tabu search for measuring graph similarity. *Lecture Notes in Computer Science*, 3434:172–182, 2005.

[16] P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6(26):290–297, 1959.

[17] E.M. Wright. Graphs on unlabelled nodes with a given number of edges. *Acta Mathematica*, 126(1):1–9, 1971.

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

[19] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2007.

[20] C. Wilson, B. Boe, A. Sala, K.P.N. Puttaswamy, and B.Y. Zhao. User interactions in social networks and their implications. In *Proc. of ACM European Conference on Computer systems*, 2009.