Seed and Grow: An Attack Against Anonymized Social Networks

Wei Peng*, Feng Li†, Xukai Zou* and Jie Wu‡
*Department of Computer and Information Science
†Department of Computer, Information and Leadership Technology
Indiana University-Purdue University, Indianapolis, Indianapolis, IN, U.S.A.
‡Department of Computer and Information Science
Temple University, Philadelphia, PA, U.S.A.

***Abstract*—Digital traces left by a user of an online 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. In this paper, we 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. Our work identifies and relaxes implicit assumptions taken by previous works, eliminates arbitrary parameters, and improves identification effectiveness and accuracy. Experiments on real-world collected datasets further corroborate our expectation and claim.**

***Keywords*-anonymity, privacy, social networks, topology**

I. 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 characteristic of online social networking services is their emphasis on user and their connections, in addition to the content as in traditional Internet services. Online social networking services, while providing conveniences to users, accumulate a treasure of user-generated contents and users' social connections, which were only available to large telecommunication service providers and intelligence agencies a decade ago.

Online social networking data, once published, are of great interest to a large audience. For instance, with the massive social networking 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 a user's demographic and preference profile and can thus issue targeted advertisements. 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".

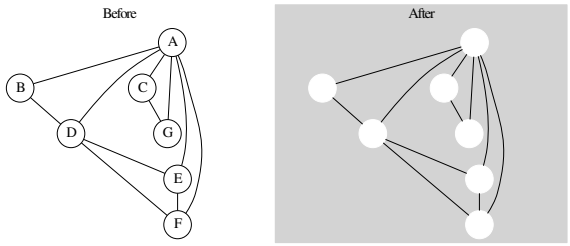


Fig. 1. Each node represents a user, with the user's ID attached. Naive anonymization simply removes the ID, but retains the network structure.

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 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., removing plainly identifying labels such as name, social security number, postal or e-mail address, and retaining the structure of the network as published data. Figure 1 illustrates 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. In several high-profile cases, anonymity has been unquestioningly interpreted as equivalent to privacy [1].

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 [2]. A few privacy attacks have been proposed to circumvent the naive anonymization protection [1, 2]. Meanwhile, more sophisticated anonymization techniques have been proposed to provide better privacy protection [3, 4, 5, 6, 7]. Nevertheless, research in this area is still in its infancy and a lot of work, both in attacks and defenses, remains to be done.

In this paper, we describe 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 breaching privacy.

More concretely, our contributions include:

- We propose an efficient seed construction and recovery algorithm (Section III-A). More specifically, we drop the assumption that the attacker has complete control over the connection between the seed and the rest of the graph (Section III-A1); the seed is constructed in a way which is only visible to the attacker (Section III-A1); the seed recovery algorithm examines at most the two-hop local neighborhood of each node, and thus is efficient (Section III-A2).
- We 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 require arbitrary parameters for probing aggressiveness, our algorithm automatically finds a good balance between identification effectiveness and accuracy (Section III-B).
- We demonstrate the significant improvements in identification effectiveness and accuracy of our algorithm over previous works with real-world social-network datasets (Section IV).

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*. Our work calls for a re-evaluation of the current privacy-protection practices in publishing social-network data.

II. Background and Related Work

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 attributes.

In this context, *privacy* can be modeled as the knowledge of existence or absence of vertices, edges, or labels. An extension is to model privacy in terms of metrics, such as betweenness, closeness, and centrality, which originate from *social network analysis* studies [8].

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 [9]. However, the information conveyed in edges and its associated labels is susceptible to privacy breaches. Backstrom et al. [2] proposed an identification attack against anonymized graph and coined the term *structural steganography*.

Beside privacy, other dimensions in formulating privacy attacks against anonymized social networks, as identified in numerous previous works [4, 5, 7, 10], 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 that is lost or distorted, the less useful published data is. Existing anonymization schemes [3, 4, 5, 7, 10] are all based on the trade-off between the usefulness of the published data and the 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.

Although trade-off between utility and privacy is necessary, 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.

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 of 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 that a user of one service would simultaneously use another service. As a person registers to different social networking services, her social connections in these services, which relate 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 the user IDs 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 III-A). By using a graph (with the 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 by carefully measuring structural similarity of these graphs, he manages to identify 100 more persons from the anonymized graph from F-net (the "Dissimilarity" interlude in Section III-B will illustrate how to do this). By doing so, Bob defeats his employer's attempt to protect the customers' privacy.

We conclude this section with a brief comment on our choice of model. We use the *undirected* graph model to explain the proposed deanonymization attack on social networks. Undirected graphs arise naturally in scenarios where the social

relation under investigation is *mutual*, e.g., friend requests must be confirmed in Facebook; *directed* graphs are more appropriate in other cases such as fans follow a movie star in Twitter. A undirected graph could be seen as a special case of directed graphs, in which the every relationship is reciprocal. As explained in Section III, the algorithms used in the proposed deanonymization attack do not rely on the fact that the target and background graphs are undirected; they work on directed graphs without any change. The choice of undirected graph model is for specificity and ease of presentation.

III. Seed and Grow: The Attack

This section describes an attack that identifies users from an anonymized social graph. Let an undirected graph $G_T = \{V_T, E_T\}$ represent the *target* social network after anonymization. We assume that the attacker has an 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.

We assume that, *before* the release of G_T, the attacker obtains (either by creating or stealing) a few accounts and connects 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 relationships between other users in G_T. Furthermore, unlike previous works, we *do not assume that 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 [11], 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 reads as "flag" or "fingerprint") into G_T before its release. After the anonymized graph is 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 comprised of 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.

A. 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 2, 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.

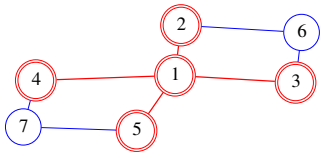


Fig. 2. A randomly generated graph G_F may be symmetric. Vertices in $G_F = \{v_1, \ldots, v_5\}$ are double-circled.

- G_F is *asymmetric*, i.e., G_F does not have any non-trivial automorphism. For example, in Figure 2, 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 of other nodes in the network is unknown to the attacker before its release, the uniquely identifiable property is not realizable. However, as was proved by Backstrom et al. [2], with a large enough size and randomly generated edges under the Erdös-Rényi model [12], 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. 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 $V_F(v)$), and once G_F is recovered from G_T, V_S can be identified uniquely.

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

1) Construction: The construction of G_F starts with a *star* structure. 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 the 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 2, 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 $V_F - \{v_h\}$

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

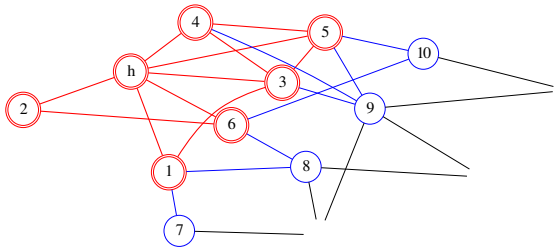


Fig. 3. An illustration of the seed stage. Vertices in the flag $G_F = \{v_h, v_1, \ldots, v_6\}$ are double-circled. The ordered internal degree sequence $\mathcal{S}_D = \langle 2, 2, 2, 3, 3, 4\rangle$. The internal degree subsequence for the neighboring vertices $V_S = \{v_7, \ldots, v_{10}\}$ of 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.

until it does (which it eventually will since we assume that $V_F(u) \neq V_F(v)$ for any pair u and v from V_S). 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 a while, 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. 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 was 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 III-A2, 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 which characterizes G_F due to the random nature in seed construction. Therefore, G_F is *visible only to the attacker*.

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 the 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 the 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 neighbors are identified with V_S by subsequence secret comparison. Algorithm 2 has the details.

Algorithm 2 Seed recovery.

for all $u \in G_T$ **do**
  if $\deg(u) = |V_F| - 1$ **then**
    $U \leftarrow$ exact 1-hop neighborhood of u
    for all $v \in U$ **do**
      $d(v) \leftarrow$ number of v's neighbors in $U \cup \{u\}$
    end for
    $s(u) \leftarrow \text{sort}(d(v)|v \in U)$
    if $s(u) = \mathcal{S}_D$ **then**
      $V \leftarrow$ exact 2-hop neighborhood of u
      for all $w \in V$ **do**
        $U(w) \leftarrow w$'s neighbors in U
        $s(w) \leftarrow \text{sort}(d(v)|v \in U(w))$
      end for
      if $\langle s(w)|w \in V\rangle = \langle \mathcal{S}_D(v)|v \in V_S\rangle$ **then**
        $\{w \in V$ is identified with $v \in V_S$ if $s(w) = \mathcal{S}_D(v)\}$
      end if
    end if
  end if
end for

SEED RECOVERY. Bob started to check the anonymized graph G_T to find the flag. He did this by examining all of the vertices in G_T for one with degree 6. After he reached a *candidate head* v_c with degree 6, he isolated it along with its candidate flag graph (red in Figure 3), and the internal degrees for each of the neighbors. He found that the ordered internal degree sequence $\langle 2, 2, 2, 3, 3, 4\rangle$ match with that of V_F. He then proceeded to isolate v_c's exact 2-hop neighbors 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 is 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., $V_F \cup V_S$) is 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 $O(N|V_T|)$.

B. 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.

We have 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 [13]. 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. Besides, the overlapping between G_T and G_B may well be *partial*, so a *full* mapping is either impossible or undesirable. Therefore, the grow algorithm adopts a progressive and self-reinforcing strategy, mapping multiple vertices for each round.

Figure 4 shows a small example. v_7 to v_{10} have already been identified in the seed stage (recall Figure 3). The task is to identify other vertices in the target graph G_T.

1) Dissimilarity: 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, we introduce a *greedy heuristic* with *revisiting* into the algorithm.

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 4, $\mathcal{N}_m^T(u_{*1}) = \{u_7, u_8, u_9\}$ and $\mathcal{N}_u^T(u_{*1}) = \{u_{*4}\}$.

For the background graph G_B, we can make similar definitions. 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 let $\mathcal{N}_u^B(v)$ be the other (i.e., unmapped) neighbors. Hence, in Figure 4, $\mathcal{N}_m^B(v_{12}) = \{v_9, v_{10}\}$ and $\mathcal{N}_u^B(v_{12}) = \{v_{11}, v_{16}\}$.

We identify the mapped vertices in V_S and V_S^* so that $\mathcal{N}_m^T(u_{*1}) - \mathcal{N}_m^B(v_{12}) = \{u_7, u_8\} = \{v_7, v_8\}$ in Figure 4.

For a pair of nodes, $u \in V_T$ and $v \in V_B$, we define the following pair of dissimilarity metrics:

$$\Delta_T(u, v) = \frac{|\mathcal{N}_m^T(u) - \mathcal{N}_m^B(v)|}{|\mathcal{N}_m^T(u)|}, \tag{1}$$

and:

$$\Delta_B(u, v) = \frac{|\mathcal{N}_m^B(v) - \mathcal{N}_m^T(u)|}{|\mathcal{N}_m^B(v)|}, \tag{2}$$

in which $|\cdot|$ is the number of set elements, i.e., set cardinality. We have $\Delta_T(u_{*1}, v_{12}) = |\{u_7, u_8\}|/|\{u_7, u_8, u_9\}| = 2/3 \approx 0.667$, 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 1 and 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 that 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 we want to choose those mappings which are the mutually best choices for the graphs. Again, a concrete example helps.

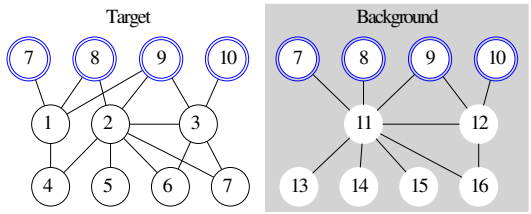


Fig. 4. An illustration of the grow stage. Vertices in the initial seed $V_S = \{v_7, \ldots, v_{10}\}$ are double-circled. Those vertices in the target graph G_T with labels starting with an asterisk are yet to be identified. The task of the grow stage is to identify these vertices.

TABLE I
DISSIMILARITY METRICS FOR PAIRS OF UNMAPPED VERTICES IN FIGURE 4. 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)

DISSIMILARITY Bob applied the dissimilarity metrics defined in Equations 1 and 2 to Figure 4 and got the results shown in Table I.

Bob first identified the tuples in Table I 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.

2) Greedy Heuristic: Bob's story suggested a way of using the dissimilarity metrics defined in Equations 1 and 2 to iteratively grow the seed.

Since smaller dissimilarity implies better match, we identify those tuples in the table like Table I which has *smallest* Δ_T and Δ_B in both its row and column; these tuples are the mutually best matches between the target graph and the background graph. We then add the mappings corresponding to these tuples to the seed and move on to the next iteration.

We 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 we choose? Rather than randomly selecting a tuple, we 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, we refrain from 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". We define an *eccentricity* metric for this purpose in our algorithm. Let X be a group of numbers (the same number can occur multiple

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

times). The *eccentricity* of a number $x \in X$ is defined as:

$$\mathcal{E}_X(x) = \begin{cases} \frac{\Delta_X(x)}{\sigma(X)\#_X(x)} & \text{if} \quad \sigma(X) \neq 0 \\ 0 & \text{if} \quad \sigma(X) = 0 \end{cases}. \tag{3}$$

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.

Therefore, if there are conflicts *in a row*, these tuples have the same Δ_T and Δ_B. For each such tuple, we collect the Δ_T and Δ_B *in the same column* 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 the *largest* $\mathcal{E}_{X_T}(\Delta_T)$ and $\mathcal{E}_{X_B}(\Delta_B)$, we pick it and add the corresponding mapping to the seed; otherwise, no mapping is added to the seed.

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; we are very 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.

We address this problem by providing a way to re-examine previous mapping decisions given new evidences in the grow algorithm; we call this *revisiting*. More concretely, for each iteration, we consider all vertices which have at least one seed neighbor, i.e., those pairs of vertices on which the dissimilarity metrics in Equations 1 and 2 are well-defined.

We expect the revisiting technique will increase the accuracy of the algorithm. The greedy heuristic with revisiting is summarized in Algorithm 3.

IV. Experiments

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

A. Setup

We used two datasets collected from different real-world social networks in our study.

The `Livejournal` dataset, which was collected from the friend relationship of the online journal service, LiveJournal, on 9–11 December 2006 [14], consists of 5.2 million vertices and 72 million links. The links are directed. As previously discussed at the end of Section II, we conducted the experiments with the more difficult setting of an undirected graph. We retained an undirected link between two vertices 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. We derived the target and background graphs from each dataset and used their shared vertices as the *ground truth* to measure against.

More precisely, we derived the graphs with the following procedure. First, we chose a connected subgraph with $N_\cap$ vertices from the dataset, which served as a *shared portion* of the background and target graphs. We then picked other two sets of vertices (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.

B. 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.

We randomly generated a number of modest-sized flag graphs with 10 to 20 vertices and planted them into the `Livejournal` dataset with Algorithm 1. We were able to uniquely recover them from the resulted graph with Algorithm 2 without exception.

To explain this result, we 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. 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.

[1] http://www.infovis-wiki.net/index.php/Social_Network_Generation.

TABLE II
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}

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. 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, the number of essentially different constructions produced by Algorithm 1 is:

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

Table II 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.

C. Grow

We compared our grow algorithm with the one proposed by Narayanan and Shmatikov [1]. There is a mandatory threshold parameter, which controls the probing aggressiveness, in their algorithm. Narayanan and Shmatikov [1] did not give a quantitative guideline to choose this parameter; we experimented with different values and found that, with increasing threshold, more nodes were identified but the accuracy decreased accordingly. Therefore, we used two different thresholds, which established a performance envelope 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 lay in the tolerance to the ambiguities in matching: the aggressive one might declare a mapping in a case where the conservative one would deem as too ambiguous.

To account for the bias on the performance measurement of a particular graph setting, for each target/background graph pair, we ran multiple simulations with different initial seeds and took the average as the performance measurement. We focused our simulations on graphs with hundreds of vertices, which are big enough to make the identification nontrivial. More precisely, we 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.

1) Initial Seed Size: Recent literature [15] on interaction-based social graphs (e.g., the social graph in the motivating scenario) singles out the attacker's interaction budget as the major limitation to attack effectiveness. The limitation translates to 1) the initial seed size and 2) the number of links between the flag graph and the initial seed. Our seed algorithm resolves the latter issue by guaranteeing unambiguous identification of the initial seed regardless of link numbers. As shown below, our grow algorithm resolves the former issue by working well with a small initial seed.

Figure 5 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 differ even among the same group of vertices, we introduced an *edge perturbation* of 0.5%, i.e., we added 0.5% of the all of the edges in the target graph.

We note a few points for Figure 5.

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

2). 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 5 include the initial seed). In contrast, even for very small initial seed of 5 nodes, Seed-and-Grow correctly identifies an average of 32 nodes for `Livejournal` and 62 nodes for `emailWeek` while only incorrectly identifying 1 node on average.

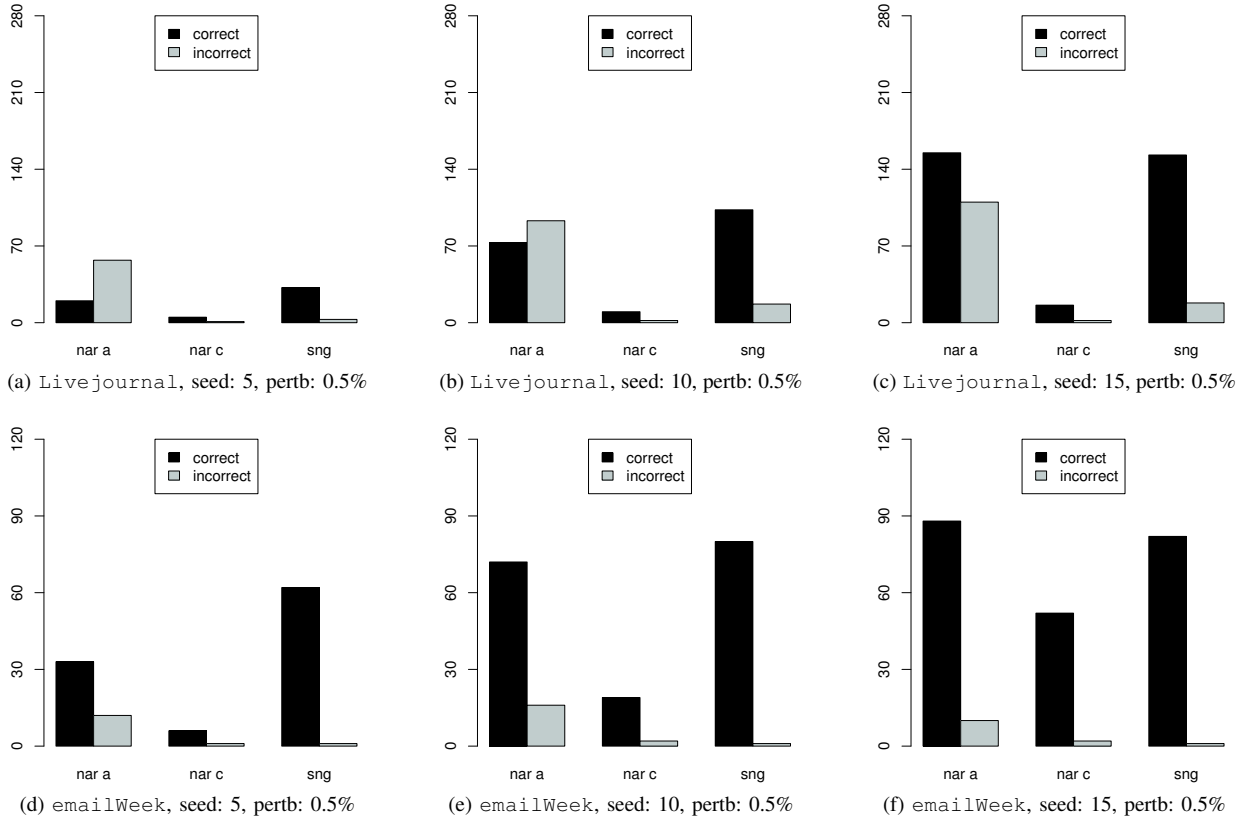
3). Though aggressive Narayanan correctly identifies more nodes as the seed size grows, the number of incorrect identification grows accordingly. This is especially evident in `Livejournal`. In contrast, the incorrect identification number for Seed-and-Grow remains constant in `emailWeek` and grows very 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 5 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.

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 if one can find such a threshold, it is unclear that its performance will be superior to that of Seed-and-Grow. In contrast, Seed-and-Grow has no such arbitrary parameter. The point is that Seed-and-Grow *automatically* finds a sensible balance between effectiveness and accuracy.

2) Edge Perturbation: The impact of edge perturbations on the grow performance is shown in Figure 6. The initial seed size was 15.

Correct identifications decreased with a larger edge pertur-



(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%

Fig. 5. 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`.

bation 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 at a much higher cost in accuracy; for `Livejournal`, aggressive Narayanan makes more incorrect identifications than correct ones. In contrast, the number of incorrect identifications for Seed-and-Grow remain almost constant with different perturbation percentages.

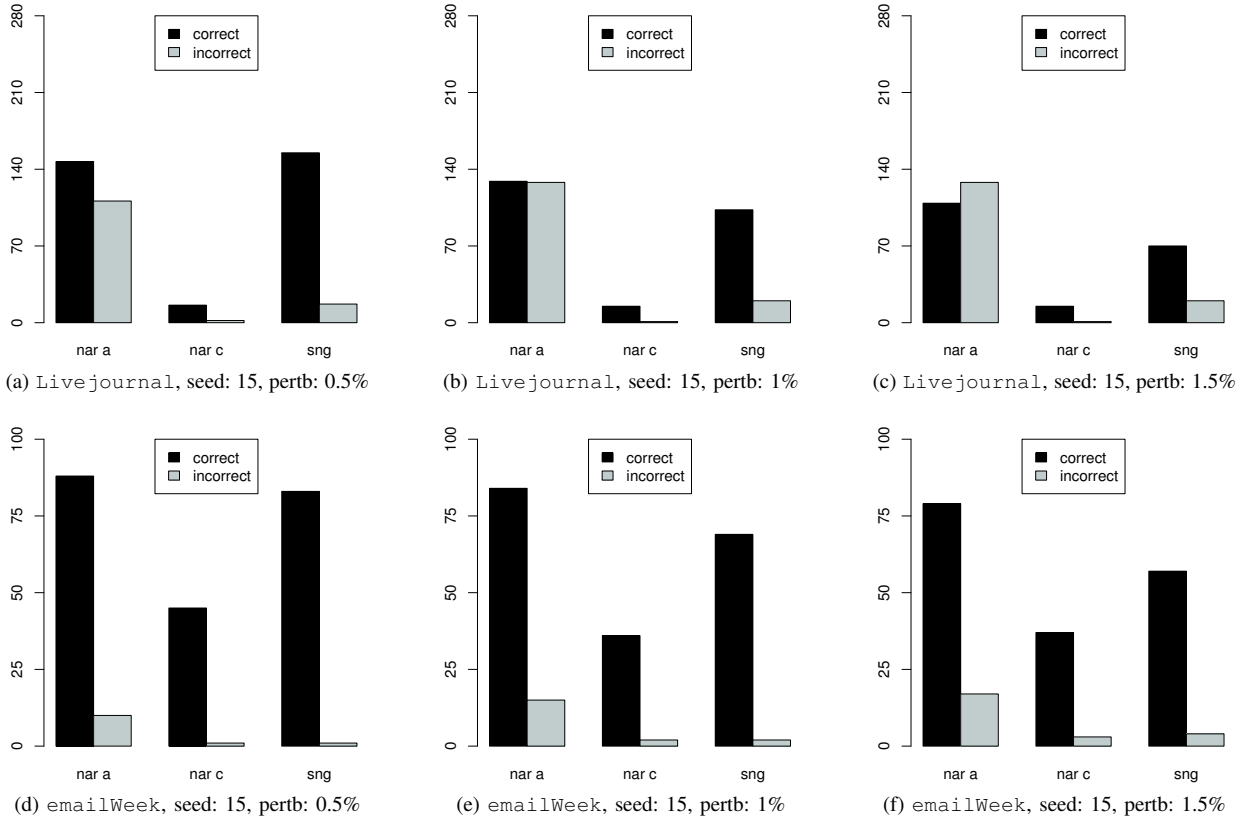
A high accuracy (i.e., a high percentage of correct identifications) is desirable, even at a reasonable cost of effectiveness (fewer nodes identified). This is because, lacking the knowledge about whether or not an identification is correct, accuracy corresponds to the user's *confidence* in the identification result. For example, in Figure 6c, 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.

On reflection, we attribute the relatively high accuracy of Seed-and-Grow to the conservative design in our grow algorithm (Algorithm 3). More specifically, we add a mapping 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.

V. Conclusion

We propose an algorithm, *Seed-and-Grow*, to identify users from an anonymized social graph. Our algorithm 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 attacker's existing knowledge of the users' social relations. We 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 simulations on real-world-collected social-network datasets.

In the future, we plan to incorporate tagging (known mappings from external information) into our structural-based



(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%

Fig. 6. Grow performance with different edge perturbation percentage. (a), (b), and (c) are from `Livejournal`; (d), (e), and (f) are from `emailWeek`

grow algorithm, which we expect will further increase identification effectiveness and accuracy.

REFERENCES

[1] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *Proc. Symp. on Security and Privacy (S&P)*. IEEE, 2009.

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

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

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

[5] A. Korolova, R. Motwani, S. Nabar, and Y. Xu, "Link privacy in social networks," in *Proc. Conf. on Information and Knowledge Management (CIKM)*. ACM, 2008.

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

[7] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting structural re-identification in anonymized social networks," *VLDB Endowment*, vol. 1, no. 1, pp. 102–114, 2008.

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

[9] K. LeFevre, D. DeWitt, and R. Ramakrishnan, "Incognito: efficient full-domain k-anonymity," in *Proc. Intl. Conf. on Management of Data (ICMD)*. ACM SIGMOD, 2005.

[10] B. Zhou, J. Pei, and W. Luk, "A brief survey on anonymization techniques for privacy preserving publishing of social network data," *SIGKDD Explorations Newsletter*, vol. 10, no. 2, pp. 12–22, 2008.

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

[12] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, no. 26, pp. 290–297, 1959.

[13] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proc. Symp. on Security and Privacy (S&P)*. IEEE, 2008.

[14] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. Internet Measurement Conference (IMC)*. ACM SIGCOMM, 2007.

[15] C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao, "User interactions in social networks and their implications," in *Proc. European Conf. on Computer Systems*. ACM, 2009.