Sharing-Habits Based Privacy Control in Social Networks

Silvie Levy, Ehud Gudes, Nurit Gal-Oz

Sharing-habits based privacy control in social networks

Silvie Levy[1], silvie.levy@gmail.com, Ehud Gudes[1], ehud@cs.bgu.ac.il, and Nurit Gal-Oz[2], galoz@cs.bgu.ac.il

Department of Mathematics and Computer Science, The Open University, Ra'anana, Israel
Department of Mathematics and Computer Science, Sapir College, Sederot, Israel

Abstract. We study users behavior in online social networks (OSN) as a means to preserve privacy. People widely use OSN for a variety of objectives and fields. Each OSN has different characteristics, requirements, and vulnerabilities of the private data shared. *Sharing-habits* refers to users' patterns of sharing information. These sharing-habits implied by the communication between users and their peers hides a lot of additional private information. Most users are not aware that the sensitive private information they share might leak to unauthorized users. We use several different well-known strategies from graph flows, and the sharing-habits of information flow among OSN users to define efficient and easy to implement algorithms for ensuring privacy preservation with a predefined privacy level.

1 Introduction

Online Social networks (OSN) are websites enabling users to build connections and relationships among each other. The OSN structure represents social interactions and relationships between entities which are the users of the OSN. Social networks are widely used by members for information sharing with the purpose of reaching as many friends as possible. The shared-information spread, is influenced by human decisions, and users are not fully aware of the possible consequences of their preferences when specifying access rules to their shared data. It is the responsibility of OSN administrators to effectively control the shared information, reduce the risk of information leakage, and constantly evaluate the potential risks of shared-information leakage. Most access rules are defined in terms of the degree of relationship required to access ones data. These rules are not refined enough to allow for dynamic denial of content from certain peers of the community.

We propose a model for access control that works with minimal user intervention. The model is based on users' patterns of sharing information denoted as *Sharing-habits*. Naturally some users are more likely to share information with others. To minimize the probability of information leakage, the social network is analyzed to determine based on these habits, the probability of information flow through network connections. In a graph representation of the network, where edges indicate relationship between users, the challenge is to select the set of edges that should be blocked to prevent leakage of the shared information to unwanted recipients. We review some methods for handling and preserving privacy in social networks, and present our new privacy preserving approach, based on sharing-habits data. Our model combines algorithms that use graph flow methods such as max-flow-min-cut, and contract. Experimental results show the effectiveness of these algorithms in controlling the flow of information sharing to allow sharing with friends while hiding from others. The paper is structured as follows: in the next section we review related work, in section 3 we define the privacy assurance in OSN problem, and in section 4 we present our method for dealing with this problem. We explain our evaluation method and primary results in section 5 and conclude by summarizing our contribution and discussing directions for future work in section 6.

2 Related Work

There are various types of Online Social Networks, each with different properties. Privacy preservation can be viewed and handled from various aspects. Carmagnola et al [5] present a research about the factors that help users identification, and information leakage in social networks, based on entity resolution. They conducted a study on the possible factors that make users vulnerable to identification, and of personal information leakage, and the perception of users about privacy related to the spreading of their public data. To find the risk factors, they studied the relations between the user behavior (habits) on OSNs and the probability of users' identification. Kleinberg and Ligett [7] describe the social network as a graph where nodes represent users, and an edge between two nodes indicates that those two users are enemies that do not wish to share information. The problem of information sharing is described as the graph coloring problem, Kleinberg and Ligett [7] analyze the stability of solutions for this problem, and the incentive of users to change the set of partners with whom they are willing to share information. Tassa and Cohen [11], handle the information release problem, and present algorithms to compute an anonymization of the released data to a level of k-anonymity; the algorithm can be used in sequential and distributed environments, while maintaining high utility of the anonymized data. Vatsalan et al [3] conducted a survey of privacy-preserving record linkage (PPRL) techniques, with an overview of techniques that allow the linking of databases between organizations while at the same time preserving the privacy of these data. In this paper Vatsalan et al [3] present taxonomy of PPRL which characterize the known PPRL techniques along 15 dimensions, highlight shortcomings of current techniques avenues for future research. Jaehong and Ravi [6] present the ORIGIN CONTROL access control model where every piece of information is associated with its creator forever. Ranjbar and Maheswaran [1], describe the social network as a graph where nodes represent users, and an edge between two nodes indicates that those two users are friends that wish to share information. They present algorithms for defining communities among users, were the information is shared among users within the community, and algorithms for defining a set of users that should be blocked in order to prevent the shared information from reaching the adversaries, and leaking outside the community. In OSN, communities are subsets of users connected to each other; the community members have common interests and high levels of mutual trust, it can be described by a connected graph, where each user is a node in the graph, and an edge connecting two nodes indicates a relationship between two users. A community is defind by Ranjbar and Maheswaran [1] from the view point of an individual user. *myCommunity* is defined as the largest sub-graph of users who are likely to receive and hold the information without leaking. In other words, myCommunity is the subset of an individual users friends that have intense and frequent interactions and describes a grouping abstraction of a set of users that surrounds an individual based on the communication patterns used for information sharing. Our study is based on the ideas described in their paper; while they only share information within the defined community, and block users that might leak information to adversaries, we relax the limitation defined in their study, and block only edges on the path to the adversaries, instead of blocking all the information from the source user to the users that might leak the information.

3 The Privacy Assurance in OSN Problem

In this section we define the general problem of privacy assurance in OSN and our proposed method that uses information from users sharing-habits.

Let $G = (V, E)$ be a directed graph that describes a social network, where V is the set of network's users, and E is the set of directed and weighted edges representing the users' information flow relationships. An edge $(u_i, u_j) \in E$ exists only if u_i shares information with u_j. The distance between two vertices,$dist_G(u_i, u_j)$ is the length of the shortest path from u_i to $u_j$in G. *Ego* is an individual focal node, it is the specific user from which we consider the information flow. A network has as many egos as it has nodes, *ego-community* is the collection of ego and all nodes to whom ego has a connection at some path length. The δ-community of a user, represented by the ego vertex u_i is the sub-graph $G_\delta(u_i)$=$(V_\delta(u_i)$,$E_\delta(i))$, where for each $v_i \in V_\delta(u_i)$, $v_i \neq u_i$, $dist_G(u_i, v_i) \leq \delta$.

The following definitions are as defined by Ranjbar et al. [1]: p_i is the probability that user u_i is willing to share the information with some of his friends.

$$p_i = \begin{cases} (outflow/inflow) & (outflow < inflow), \\ 1 & (outflow \geq inflow). \end{cases} \tag{1}$$

- Outflow is the number of sharing interactions from u_i to his friends.
- Inflow is the number of sharing interactions from $u_i's$ friends to u_i.

The likelihood of u_i sharing information with u_j along the edge (u_i, u_j) is represented by $w_{i,j}$, the weight on the edge (u_i, u_j); This weight is derived from the relationship between u_i and u_j, it is a fixed number indicating the willingness of u_i to share information with u_j, it does not change or change very infrequently, and may be set by the user. The probability of flow between two neighbor users, u_i and u_j is denoted as p_{ij}, and calculated by $p_{i,j} = p_i \times w_{i,j}$. Since the flow may change quite often this probability may also changes with it. We assume that the user behavior is consistent; user u_i shares all the data with user u_j with probability $p_{i,j}$. This probability can change with time, but it does not depend on the content of the shared information. The Probability of Information Flow (PIF), is the maximum probability of information flow throughout the entire paths between u_i and u_j. A path probability flow between u_i and u_j is the flow of the edge with the minimum $p_{i,j}$. It is denoted as $PATH_{i,j}$. The PIF is the maximum among of all paths between u_i and u_j of $PATH_{i,j}$. The function f which denotes flow, is computed by using the log of the edges' probabilities on a path between u_i and u_j. To prevent information flow from one user to another we search for the minimal set of edges that when removed from the community graph, or blocked, disables the flow. We denote this set of blocked edges as B. Note that after edges are removed, the PIF and therefore f should be recomputed.

3.1 Problem goal

Our aim is to enable a user u_i to share information with as many friends and acquaintances as possible, while preventing information leakage to adversaries within the user's community. Ranjbar et al. [1] describe a method for sharing information within the source user u_i defined community, while blocking users (friends and acquaintances) that might leak information to adversaries. We relax the limitation due to blocking friends, and instead of blocking all the information from the source user u_i to the users that might leak the information, block only edges on the path from u_i to his adversaries. We use the following criteria to define and evaluate the resulting u_i ego-community graph:

1. Minimum Friends Information Flow: the minimum information flow from u_i to every user within his community must preserve a certain percentage of the original information flow to every user denoted by α.
   Let $G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$ be the δ-community of $u_i, v \in V(u_i)$
   $$f(u_i, v) \geq \alpha \cdot f_{original}(u_i, v) \tag{2}$$
2. Close Friends Distance: Close friends are defined by their distance from u_i. $G_\beta(u_i) = (V_\delta(u_i), E_\delta(u_i))$ is the β-community of $u_i, v \in V(u_i)$, $\beta < \delta$. This criteria reflects the requirement that all the users within $u_i's$ β-community must receive the entire information from u_i, and cannot be blocked.
   Let B be the set of blocked edges, than
   $$B \subset \{(u_s, u_t) | d_{G_\delta}(u_i, u_s) \geq \beta, u_s, u_t, u_i \in V_{G_\delta}(u_i)\} \tag{3}$$
   We assume that there are no adversaries within $u_i's$ β-community, otherwise the above condition can never be fulfilled.
3. Maximum Adversaries Information Flow: the maximum information flow from u_i to each of his adversaries cannot be more than γ from the original

information flow to each adversary.

$$f(u_i, u_{adv}) \leq \gamma \cdot f_{original}(u_i, u_{adv}) \tag{4}$$

For example the threshold parameters can be: $\alpha = 0.9$, $\beta = 2$, and $\gamma = 0.1$. The problem goal is to remove the least number of edges such that the three equations 2,3,4 will be satisfied.

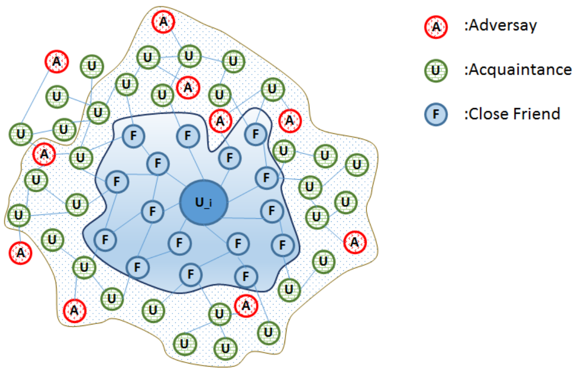

Fig. 1: $u_i's$ Community Graph

Figure 1 describes a δ-community graph for u_i. The dotted area surrounds $u_i's$ δ-community graph with $\delta = 4$, i.e all acquaintances within distance ≤ 4. The blue area surrounds $u_i's$ β-community, i.e all friends within distance ≤ 2.
As shown by the figure the δ-community of friends is much larger than the β-community of close friends.

3.2 Cuts in graphs

A cut in a graph is a set of edges between two subsets of a graph, one containing u_i, and the other containing $u_i's$ adversaries, such that when removed, prevents information flow from one subset to the other.
A naive algorithm for solving the problem would be an algorithm that finds any cut between the adversaries' set and $u_i's$ community, and defines this cut as the blocked edges list. Algorithm 1 is a naive algorithm for blocked users.

The naive algorithm is not suitable for our problem, since it doesn't comply with the (1) Minimum Friends Information Flow, (2) Close Friends Distance-criteria of our problem. Condition (1) requires minimum information flow from u_i to all members in $u_i's$ community, the naive algorithm doesn't handle this requirement. Condition (2) defines close friends by their distance from u_i, the naive algorithm doesn't handle this requirement. While the naive algorithm is not sufficient to our problem, it is important for understanding the theoretical problem defined here.

Algorithm 1 Naive algorithm for blocked users

Input: $G = (V, E)$ a directed graph that describes the social network.
u_i the ego user.
δ the community distance criteria.
$AdversariesList$: the list of $u_i's$ adversaries.

Output: B:the set of blocked edges.

```
1: set $B = \emptyset$
2: for all $u_j \in V$ and $(u_j \notin AdversariesList)$ and $(dist_G(u_i, u_j) \leq \delta)$ do
3:     insert $u_j$ to $V_\delta(u_i)$
4: for all $u_j \in AdversariesList$ do
5:     insert $u_j$ to $V_\delta(adversaries)$
6: Choose any cut between the community graph, $V_\delta(u_i)$ and the adversaries
   $V_\delta(adversaries)$.
7: for all $e_{ij} \in$ {the cut between $V_\delta(u_i)$ and $V_\delta(adversaries)$} do
8:     insert $e_{ij}$ to $B$
9: return $B$
```

4 The Sharing-habits based privacy assurance in OSN Solution

In our solution we propose a model for finding the set of edges that should be blocked in order to achieve maximum information sharing among the community of the information source with minimum information leak. Our model uses two methods for defining candidate sets for blocked edges, along with the evaluation method for choosing the best set to be blocked. Our method consists of two major steps, the first is the initialization step that creates a multi-graph with a super-vertex s_1 containing $u_i's$ β-community, this step is described in sub-section 4.1. The second step described in sub-section 4.2, uses two methods to find candidates-sets for blocked edges.
Algorithm 2 warps these steps to construct the set of edges to be blocked.

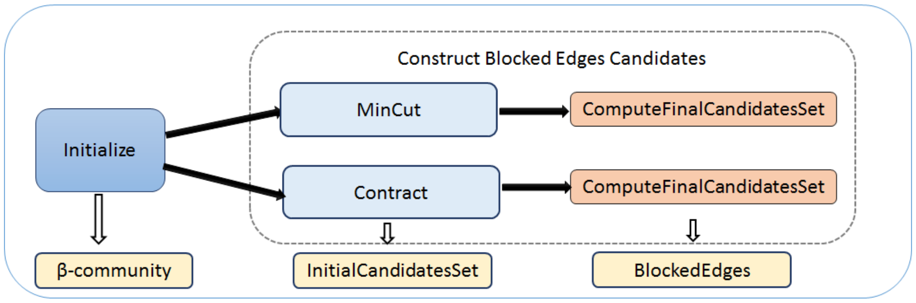
Construct Blocked Edges Candidates
Initialize
MinCut
Contract
ComputeFinalCandidatesSet
ComputeFinalCandidatesSet
β-community
InitialCandidatesSet
BlockedEdges

Fig. 2: Construct Blocked Edges main building blocks

Figure 2 describes the main building blocks of the algorithm for defining the edges to be removed from $u_i's$ δ-community in order to prevent information leakage to $u_i's$ adversaries.
Next we detail each one of these building blocks.

Algorithm 2 Construct blocked edges

Input: u_i: the ego vertex.

$G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: $u_i's$ δ-community graph.

α, γ: Flow thresholds.

β: β-community distance.

$AdversariesList$: the list of $u_i's$ adversaries.

Output: B:a set with edges to be blocked.

```
1: $MultiSet$ function Initialize($u_i, G_\delta(u_i), \beta, AdversariesList$)
2: set $s_1 = \{u_i\}$
3: for all ($u_j \in G_\delta(u_i)$) and ($dist_{G_\delta}(u_i, u_j) \leq \beta$) do
4:     if ($u_j \notin AdversariesList$) then
5:         insert $u_j$ to $s_1$
6:     else
7:         return $\emptyset$
8: return $s_1$
   {————Main——————————}
9: $s_1$ =Initialize($u_i, G_\delta(u_i), \beta, AdversariesList$)
10: if ($s_1 \neq \emptyset$) then
11:     $InitialCandidatesSet$=CondtructBlockedEdgesCandidates($u_i, G_\delta(u_i), s_1, AdversariesList, \alpha, \gamma$)
12: $B$ = SelectBestBlockedEdges($InitialCandidatesSet$)
13: return $B$
```

4.1 Initialization

The δ-community of $u_i's$ consists of all users u_j connected to u_i with a path with distance $\leq \delta$. The β parameter defines the size of the community of close friends. Therefore, a β-community of u_i would be a sub-graph contained in δ-community were $\beta \leq \delta$, as demonstrated in figure 1. The privacy criteria that is defined in sub-section 3.1 requires that the entire information shared by u_i must be shared with $u_i's$ close friends (2). In order to comply with (2), the Initialization step creates a multi-graph with one super-vertex s_1 containing u_i and his close friends with distance $\leq \beta$. This step ensures that the algorithm won't define edges for blocking on paths between u_i and his close friends, since u_i and his close friends are in the same super-vertex,s_1, see figure 3.

Figure 3(a) describes a δ-community graph for u_0, with 10 members, δ=3, 4 are close friends with distances=1 (blue vertices), 4 acquaintances (green vertices), and 2 adversaries (red vertices). Figure 3(b) describes the graph after initialization.

4.2 Construct Blocked Edges Candidates

We use two methods derived from flow problems, to find the initial candidates-set of edges to be blocked. This candidates-set is a cut between two sets of vertices, one set containing u_i, $u_i's$ β-community, and some vertices from of $u_i's$ δ-community. The other set containing the remaining part of $u_i's$ δ-community, and $u_i's$ adversaries.

This candidates-set is evaluated to filter out the final candidates-sets by selecting a set that complies with the required privacy criteria. This process is described in section 4.3; the two methods we use for finding the initial candidates-sets of edges to be blocked are:

1. *Min-Cut*: based on Ford-Fulkerson [4], Max-flow-min-cut algorithm, to find the minimum cut between super-vertex s_1 that contains u_i and his close friends, and each of $u_i's$ adversaries. This process is described in paragraph 4.2.
2. *Contract*: based on Karger et al. [9], contract algorithm, to find any cut between super-vertex s_1 that contains u_i and his close friends, and each of $u_i's$ adversaries, .This process is described in subsection 4.2.

4.2.1 Block edges by Min-Cut

Algorithm 3 implements the Sharing-habits privacy assurance based on the max-flow min-cut method by Ford and Fulkerson [4], and then checks for privacy criteria compliance:

1. Find a minimum cut between super-vertex s_1 and $u_i's$ adversaries [4].
2. Check if the cut complies with the required privacy criteria as defined in sub-section 3.1, and select the final candidates-set. This process is described in section 4.3.

Algorithm 3 Block edges by Min-Cut

```
Input: $u_i$: the ego vertex.
       $G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: $u_i's$ $\delta$-community graph, after the initialization step.
       $\alpha, \gamma$: Flow threshold.
       $AdversariesList$: the list of $u_i's$ adversaries.
Output: $B$:a set with edges to be blocked.
1: set $B = \emptyset$
2: $InitialBlockedEdges$= FindMinCut($u_i$,$G_\delta(u_i)$,$AdversariesList$)
3: if ($InitialBlockedEdges \neq \emptyset$) then
4:     $B$=ComputeFinalCandidatesSet($u_i$,$G_\delta(u_i)$,$AdversariesList$,$InitialBlockedEdges$,$\alpha, \gamma$)
5: return $B$
```

4.2.2 Block edges by Contract

The minimum cut between $G_\beta(u_i)$, and $u_i's$ adversaries, found by BlockEdges-ByMinCut algorithm, might not be the optimal solution for our problem, since the edges in this cut may not satisfiy the privacy criteria. Thus, we use the contract algorithm, that finds a variety of other cuts possibly complying with the required privacy criteria.
Algorithm 4 implements the Sharing-habits privacy assurance based on the contract method by Karger and Stein [8, 9].
In each iteration, the contract algorithm finds a different cut between the super-vertex containing $G_\beta(u_i)$ and the super-vertex containing $u_i's$ adversaries. The contract algorithm repeatedly contract vertices to super-vertices until it gets two super-vertices connected by a set of edges that defines a cut between the two sets of vertices contained in each super-vertex.

Algorithm 4 is composed of the following main steps:

1. Find a cut between super-vertex $G_\beta(u_i)$ and $u_i's$ adversaries; this step uses the contract algorithm presented [8, 9]
2. Check if the cut complies with the required privacy criteria as defined in sub-section 3.1, and select the final candidates-set. This process is described in 4.3.

Algorithm 4 Block edges by Contract

Input: u_i : the source.

$G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: $u_i's$ δ-community graph, after the initialization step.

α, γ: Flow thresholds.

$AdversariesList$: the list of $u_i's$ adversaries.

Output: B:the set with the blocked edges.

```
1: set $B = \emptyset$
2: $InitialBlockedEdges$= ContractFindCut($u_i$,$G_\delta(u_i)$, $AdversariesList$)
3: **if** ($InitialBlockedEdges \neq \emptyset$) **then**
4:    $B$=ComputeFinalCandidatesSet($u_i$,$G_\delta(u_i)$,$AdversariesList$,$InitialBlockedEdges$,$\alpha, \gamma$)
5: **return** $B$
```

Algorithm 5 is called by algorithm 4 to find a cut between two vertices by randomly selecting an edge and contracting the two vertices connected by the selected edge into one super-vertex.

Algorithm 5 ContractFindCut

Find a cut in a graph by repeatedly contracting vertices into two super vertices

Input: $G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: δ-community multi-graph, after the initialization step.

u_i: the source.

$AdversariesList$: the list of $u_i's$ adversaries.

Output: $CutSet$: the resulting cut.

```
1: set $CutSet = \emptyset$
2: **repeat**
3:    **if** (all edges $(u, v)$ are tested) **then**
4:       **return** $CutSet$
5:    **else**
6:       choose an edge $(u, v)$ uniformly at random from $E \setminus testededges$
7:       **if** ($u$ and $v$ don't contain each others' adversaries) **then**
8:          contract the vertices $u$ and $v$ to a super vertex $w$
9:          keep parallel edges, remove self loops
10: **until** ($G$ has only two super-vertices)
11: set $CutSet$ = the edges between the two vertices
12: **return** $CutSet$
```

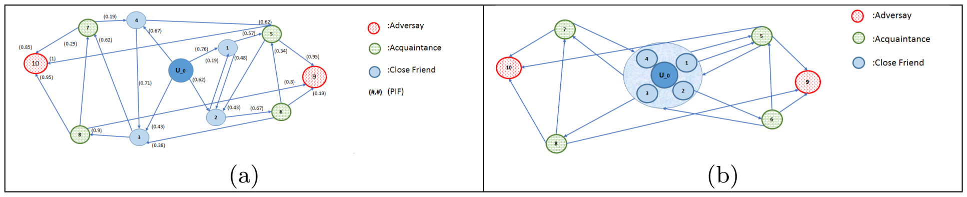

Fig. 3: $u_0's$ δ-community graph: (a) $u_0's$ comuunity (b) after initialization

Figures 4- 5 describe a simple community graph and some steps of one run of the contract algorithm.

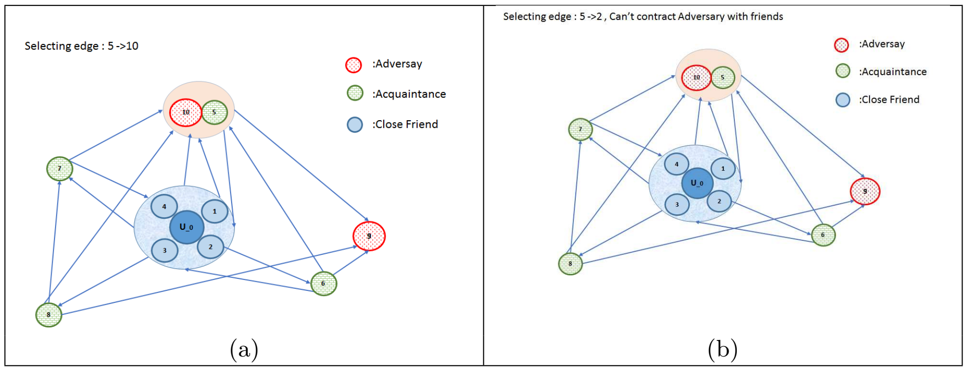

Fig. 4: Contract: (a) Edge (5,10) was randomly selected, (b) Edge (5, 2) cannot be selected, since the algorithm can't contract a super-vertex containing u_0 with a super-vertex containing $u_0's$ adversary.

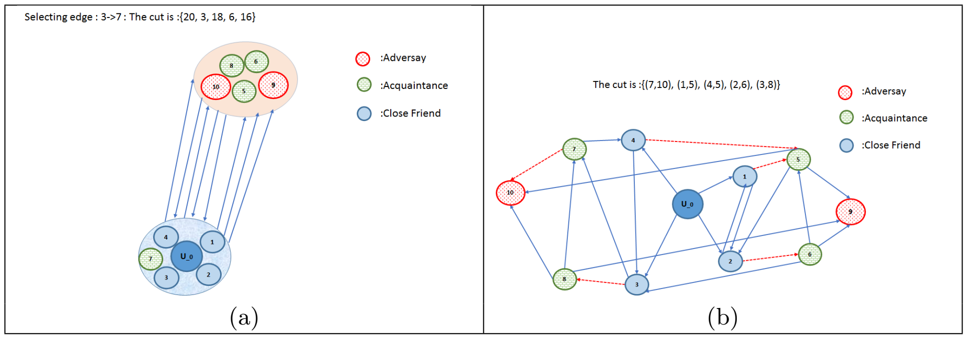
Selecting edge : 3->7 : The cut is :{20, 3, 18, 6, 16}
:Adversay
:Acquaintance
:Close Friend
(a)
The cut is :{(7,10), (1,5), (4,5), (2,6), (3,8)}
:Adversay
:Acquaintance
:Close Friend
(b)

Fig. 5: Contract: (a) Edge (3, 7) is randomly selected (b) The obtained cut from one run of Contract algorithm

4.3 Compute Final Candidates Set

After selecting the initial candidates-set of edges to be blocked, each method uses algorithm 6 for selecting the final candidates-set of edges that should be removed from $u_i's$ δ-community graph. In the first step of the algorithm, we check if by removing the initial-candidates-set of edges from $u_i's$ δ-community graph, the remaining δ-community graph for user u_i complies with the required privacy criteria. If it doesn't comply with the required privacy criteria, we try to remove edges from the initial blocked candidates-set, and insert them back into $u_i's$ δ-community graph, until the remaining community graph complies with the required criteria, or until we tested the entire edges in the initial candidate-set, and couldn't find a set of edges to be blocked. We propose three methods for selecting and removing an edge from the initial candidates-set, and insert the selected edge back to δ-community graph:

1. Randomize: select an edge randomly.
2. Maximum PIF: select the edge with the maximum probability of information flow.
3. Minimum PIF: selecting the edge with the minimum probability of information flow.

Algorithm 6 implements the three methods and algorithm 7 tests the criteria.

Algorithm 6 Compute final candidates-set

Input: u_i : the source.
G_{u_i}: $u_i's$ δ-community multi-graph, after the initialization step.
$AdversariesList$: the list of $u_i's$ adversaries.
$InitialBlockedEdges$: the list of edges to be blocked.
α, γ: Flow thresholds.
$EdgeMethod$: the method for selecting the next edge for unblocking.
Output: $BlockedEdges$: the final set of edges to be blocked.

```
1: while ((ComputeCriteria($u_i, G_{u_i}, AdversariesList, InitialBlockedEdges, \alpha, \gamma$) $\neq$
   $TRUE$) and ($InitialBlockedEdges \neq \emptyset$)) do
2:   switch ($EdgeMethod$)
3:   case Random:
4:     $e \leftarrow$ select random $(u,v) \in E_{u_i}$
5:   case MaxPIF:
6:     $e \leftarrow \arg\max_{(u,v)\in E_{u_i}} PIF$
7:   case MinPIF:
8:     $e \leftarrow \arg\min_{(u,v)\in E_{u_i}} PIF$
9:   end switch
10:  InitialBlockedEdges = InitialBlockedEdges $\setminus(u,v)$
11: $BlockedEdges = InitialBlockedEdges$
12: return $BlockedEdges$
```

5 Evaluation

In this section we describe the evaluation method we use for the proposed algorithm, and the results we obtained using real data [10]. We first demonstrate our methods and the difference between them using a toy community.

Algorithm 7 Compute the required criteria

Input: u_i : the source.

G_{u_i}: $u_i's$ δ-community multi-graph, after the initialization step.

$AdversariesList$: the list of $u_i's$ adversaries.

$BlockedEdges$: the list of edges to be removed.

α, γ: Flow thresholds.

Output: $ComplyCriteria$: indicator whether the community graph without the blocking-set complies with the required privacy criteria.

```
1: for all $v \in G_{u_i}$ do
2:    if $(f(u_i, v) < \alpha \cdot f_{original}(u_i, v))$ then
3:       return $FALSE$
4: for all $a_{adv} \in AdversariesList$ do
5:    if $(f(u_i, u_{adv}) > \gamma \cdot f_{original}(u_i, u_{adv}))$ then
6:       return $FALSE$
7: return $TRUE$
```

5.1 Demonstration on a synthetic community

We demonstrate our algorithms on a small graph representing a synthetic community that we built from the example in [2], containing 11 vertices, and 23 edges. We selected community distance $\delta = 3$, close friends distance $\beta = 1$, and assigned 2 adversaries. The algorithms were tested with different probabilities of information flow from source user U_0 to the community members. In the following example, Figure 6 describes the synthetic community graph with high probability of information flow on the edges to adversaries. This situation simulates a collision, and it is hard to select α and γ such that we get minimum leakage of information flow to $u_i's$ adversaries, and maximum information flow to $u_0's$ community.

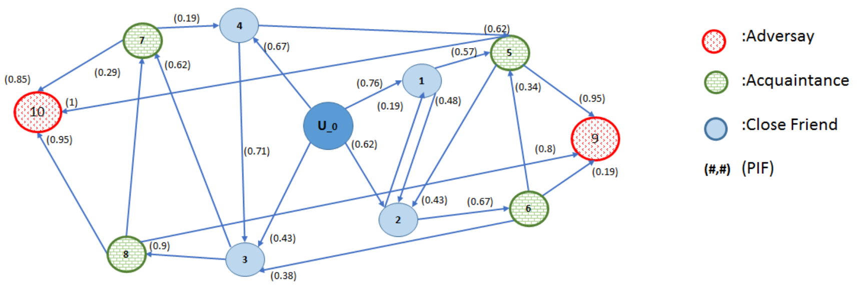

Fig. 6: Synthetic community graph with collision

In this community graph U_0 is the source, U_0 has four close friends: $1, 2, 3, 4$, four acquaintances: $5, 6, 7, 8$, and two adversaries: $9, 10$.
Each adversary has three incoming edges.
$\{(6,9), (5,9), (8,9)\}$ with probabilities $(0.19, 0.95, 0.8)$ respectively.
$\{(5,10), (7,10), (8,10)\}$ with probabilities $(1, 0.85, 0.95)$ respectively.

The maximum probability of information flow from U_0 to the members of his community graph is depicted in table 1.

User	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
MAX PIF	0.76	0.62	0.43	0.67	0.4332	0.4154	0.2949	0.4281	0.4115	0.4332

Table 1: PIF from U_0 to his community

Next, using this example we show why the contract approach has better chance of finding a good set of edged that can be blocked while satisfying the privacy criteria.

Edge	**PIF**
(5,9)	0.95
(6,9)	0.19
(8,9)	0.8
(3,8)	0.9
(3,7)	0.62
(5,10)	1

Table 2: Min-Cut Candidates

Edge	**PIF**
(4,5)	0.62
(1,5)	0.57
(2,6)	0.67
(3,8)	0.9
(7,10)	0.85

Table 3: Contract Candidates

Block edges by Min-Cut method
The Minimum cut found by Min-Cut method is depicted in table 2. If we remove the initial candidates-set edges from $u_0's$ community graph, the probability of information flow to 7 and 8 will be 0, meaning no flow at all. In the final step of algorithm 6, we try unblocking each edge from the initial candidtes-set, and reach the required privacy criteria, which is computed by algorithm 7; in this example the only edge that improves the PIF to community without increasing the information leakage to $u_0's$ adversaries is $(3, 7)$, thus the final candidates-set is $\{(3, 7)\}$. In this example we can't define α, and γ with values that comply with the required privacy criteria, which is computed by algorithm 7.

Block edges by Contract method
A Cut found by an iteration of contract method is depicted in table 3.

If we remove the initial candidates-set edges, the probability of information flow to 5, 6, and 8 will be 0, meaning no flow at all. Algorithm 6 tries unblocking each edge from the initial candidtes-set, and reach the required privacy criteria, which is computed by algorithm 7; the final candidates-set is empty, since each edge we unblock not only improves the information flow to $u_0's$ community, but also increases the information leakage to $u_0's$ adversaries.
It is obvious that when the edges to the adversaries have high probabilities, the max-flow-min-cut methods might not select those edges, and might not find a solution that comply with the required privacy criteria, while the contract method might find the trivial cut that contains only the edges to the adversaries, and thus comply with the required privacy criteria.

5.2 Test on SNAP Database

We evaluated our algorithms on the Facebook network data from Stanford Large Network Data-set Collection [10]. The SNAP library is being actively developed since 2004 and is organically growing as a result of Stanford research pursuits in analysis of large social and information networks. The website was launched in July 2009. The social network graph describes the Social circles from Facebook (anonymized) and consists 4,039 nodes (users), and 88,234 edges, it describes the Social circles from Facebook (anonymized). We took the structure and relationship from the SNAP database, and assigned random probabilities to the edges in the network graph in the following way. We defined four types of users, the type reflects the user's willing to share information: very high sharing users, medium, sometimes, and very low. For each user in the graph we randomly assigned a type. To conform the edges' probabilities to the users' types, we randomly assigned probabilities to the users' edges according to their types, from the following ranges: very high sharing users (probability 0.75-1), medium (0.5-0.75), sometimes(0.25-0.5), very low (0-0.25). The four types were generated uniformly among all the network users.

Preliminary Results

Tables 4- 5 summarize the results of four different evaluation runs, for different communities.

Run	**Vertices**	**Edges**	**Friends**	**Adversaries**
1	334	968	15	2
2	1036	2428	26	3
3	1495	6886	40	10
4	206	3755	29	2

Table 4: Data size

Table 4 presents four runs with the four different sub-communities. The community size is derived by the user selected as the sharing user. Friends column refers to the amount of first degree friends. Table 5 present the results obtained by the four runs. Columns 2-3 and 4-5 present the initial set of edges to be blocked and the final set of edges found by min-cut and contract algorithm respectively. Columns 6-8 present the threshold parameters used for the run. The difference between the two algorithms is the method for finding the initial candidates set, min-cut versus contract. Both algorithms use the same method for computing the privacy criteria. For each community graph we performed the algorithms with extreme thresholds, ($\alpha = 0, \beta = 1, \gamma = 1$, and $\alpha = 1, \beta = 1, \gamma = 0$), and with random thresholds. The remark indicates which edges were found as candidates for blocking. We can see that in the simple case (e.g., run 1 and 2)the solution is trivial and the blocked edges were the edges to the adversaries.

Run	**MinCut Initial Edges**	**MinCut Final Edges**	**Contract Initial Edges**	**Contract Final Edges**	α	β	γ	**Remark**
1	2	2	7	7	0	0	1	All edges to adv
1	2	2	7	7	1	0	0	All edges to adv
1	2	2	7	7	0.783	1	0.5654	All edges to adv
2	2	2	2	2	0	0	1	contract=mincut=edges to adv
2	2	2	2	2	1	0	0	contract=mincut=edges to adv
2	2	2	2	2	0.056	1	0.4266	contract=mincut=edges to adv
3	29	29	5	5	0	0	1	mincut=adv, contract=mix
3	29	29	5	0	1	0	0	mincut=adv, contract=mix
3	29	29	12	0	0.8867	1	0.0376	mincut=adv, contract=mix
4	2	0	34	34	0	0	1	mincut=mix, contract=mix
4	2	0	51	19	1	0	0	mincut=mix, contract=mix
4	2	0	286	181	0.0846	1	0.6478	mincut=mix, contract=mix

Table 5: Evaluation Runs Results

While both algorithm are complete, in the non trivial cases, min-cut finds the the best solution with respect to blocking adversaries, while contract may return a compromised solution that is less efficient in blocking adversaries but allows more sharing with friend. However, the time performance of the contract is much better.

It is important to note that the contract algorithm if executed multiple times, is guaranteed to eventually find the optimal solution with respect to the threshold criteria. In the case where there is no solution, the contract algorithm will provide the best cut that satisfies the threshold.

6 Conclusion

The problem of uncontrolled information flow in social network is a true concern to ones privacy. In this paper we address the need to follow the social trend of information sharing while enabling the owner to prevent their information from flowing to undesired recipients. The goal of the suggested method is to find the minimal set of edges that should be excluded from ones community graph to allow sharing of information while blocking adversaries. To reduce side effect of limiting legitimate information flow, we minimize this impact according to the flow probability. Our algorithms can be used within the ORIGIN CONTROL access control model [6]. In this model every piece of information is associated with its creator forever. The set of cut edges found by our algorithms, is stored for each user and can be checked when the origin controlled information is accessed. This way the administrator can check whenever this information is access by a certain user, if the edge between them was cut for the originator user. In future work, we intend to expand the evaluation and test our algorithms on different types of social networks (e.g., twitter). We intend to further explore more approaches to identifying the edges to be blocked, such as genetic algorithm.

References

1. Muthucumaru Maheswaran Amin Ranjbar. Using community structure to control information sharing in online social networks. *Computer Communications*, 41:11–21, 2014.
2. Rivest Ronald L. Cormen Thomas H., Leiserson Charles E. *Introduction to Algorithms*, chapter 27, page 581. MIT Press, Cambridge, Massachusetts 02142, 1990 (2009).
3. Vassilios S.Verykios Dinusha Vatsalan, Peter Christen. A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38:946969, 2013.
4. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
5. Ilaria Torre Francesca Carmagnola, Francesco Osborne. Escaping the big brother: An empirical study on factors influencing identification and information leakage on the web. *Journal of Information Science*, 40(2):180–197, 2014.
6. Ravi Sandhu Jaehong Park. Originator control in usage control. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 60–66, Washington DC, 2002. IEEE.
7. Katrina Ligett Jon Kleinberg. Information-sharing in social networks. *Games and Economic Behavior*, 82:702–716, 2013.
8. D. R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, New York, 1993. ACM.
9. D. R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43 (4):601–604, 1996.
10. Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.
11. Dror J. Tassa, Tamir; Cohen. Anonymization of centralized and distributed social networks by sequential clustering. *IEEE Transactions on Knowledge & Data Engineering*, 25 Issue 2:311–324, Feb2013.