Privacy in Social Networks: Structural identity disclosure

1

Methods based on k-anonymity

- k-candidate
- k-degree
- k-neighborhood
- k-automorphism

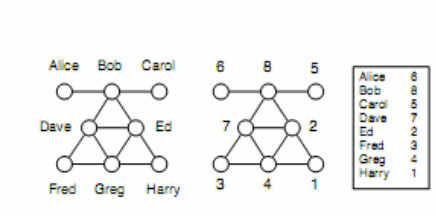
2

k-candidate Anonymity

M Hay et al, *Resisting Structural Re-identification in Anonymized Social Networks* VLDB 2008

3

G_a the naive anonymization of G through an anonymization mapping f



An individual x ∈ V called the target has a candidate set, denoted cand(x) which consists of the nodes of G_a that could possibly correspond to x

Given an uninformed adversary, each individual has the same risk of re-identification, cand(x) = V_a

In practice, background knowledge, examples:

Bob has three or more neighbors, cand(Bob) =?

Greg is connected to at least two nodes, each with degree 2, cand(Greg) =?

4

Focus on

- (background knowledge) structural re-identification where the information of the adversary is about graph structure

- (utility) analysis about structural properties: finding communities, fitting power-law graph models, enumerating motifs, measuring diffusion, accessing resiliency

Two factors

- descriptive power of the external information – background knowledge
- structural similarity of nodes – graph properties

5

Knowledge Acquisition in Practice

External information may be acquired through

- malicious actions by the adversary (active attacks) or
- through public information sources

An adversary may be a participant in the network with some innate knowledge of entities and their relationships

Radius - neighborhood

(locality) Adversary knowledge about a targeted individual tends to be local to the targeted nodes

6

Knowledge Acquisition in Practice

Closed-World vs Open-World Adversary

Assumption: External information sources are **accurate**, but **not necessarily complete**

- Closed-world: absent facts are false
- Open-world: absent facts are simply unknown

7

Anonymity through Structural Similarity

[automorphic equivalence]. Two nodes x, y ∈ V are automorphically equivalent (denoted x ≡ y) if there exists an isomorphism from the graph onto itself that maps x to y.

Example: Fred and Harry, but not Bob and Ed

Automorphic equivalence induces a partitioning on V into sets whose members have identical structural properties.

An adversary —even with exhaustive knowledge of the structural position of a target node — cannot identify an individual beyond the set of entities to which it is automorphically equivalent.

8

Anonymity through Structural Similarity

- Some special graphs have large automorphic equivalence classes.
  - E.g., complete graph, a ring
- In general, an extremely strong notion of structural similarity.

9

Adversary Knowledge (model)

An adversary access a source that provides answers to a restricted knowledge query Q evaluated for a single target node of the original graph G.

knowledge gathered by the adversary is accurate.

For target x, use Q(x) to refine the candidate set.

[CANDIDATE SET UNDER Q]. For a query Q over a graph, the candidate set of x w.r.t Q is candQ(x) = {y $\in V_a$ | Q(x) = Q(y)}.

10

Adversary Knowledge

1. Vertex Refinement Queries
2. Subgraph Queries
3. Hub Fingerprint Queries

11

Vertex Refinement Queries

A class of queries of increasing power which report on the local structure of the graph around a node.

- The weakest knowledge query, H_0, simply returns the label of the node.
- $H_1(x)$ returns the degree of x,
- $H_2(x)$ returns the multiset of each neighbors' degree,
- $H_i(x)$ returns the multiset of values which are the result of evaluating H_{i-1} on the nodes adjacent to x

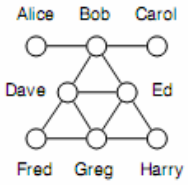
$$\mathcal{H}_i(x) = \{\mathcal{H}_{i-1}(z_1), \mathcal{H}_{i-1}(z_2) \ldots, \mathcal{H}_{i-1}(z_m)\}$$

where $z_1 \ldots z_m$ are the nodes adjacent to x.

H* Iterative computation of H until no new vertices are distinguished.

12

Vertex Refinement Queries II



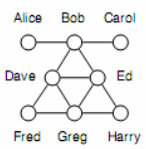
(a) graph

Node ID	$\mathcal{H}_0$	$\mathcal{H}_1$	$\mathcal{H}_2$
Alice	ϵ	1	$\{4\}$
Bob	ϵ	4	$\{1,1,4,4\}$
Carol	ϵ	1	$\{4\}$
Dave	ϵ	4	$\{2,4,4,4\}$
Ed	ϵ	4	$\{2,4,4,4\}$
Fred	ϵ	2	$\{4,4\}$
Greg	ϵ	4	$\{2,2,4,4\}$
Harry	ϵ	2	$\{4,4\}$

(b) vertex refinements

Vertex Refinement Queries III

DEFINITION 2 (RELATIVE EQUIVALENCE). Two nodes x, y in a graph are equivalent relative to H_i, denoted $x \equiv_{Hi} y$, if and only if $H_i(x) = H_i(y)$.



(a) graph

Node ID	$\mathcal{H}_0$	$\mathcal{H}_1$	$\mathcal{H}_2$
Alice	ϵ	1	$\{4\}$
Bob	ϵ	4	$\{1,1,4,4\}$
Carol	ϵ	1	$\{4\}$
Dave	ϵ	4	$\{2,4,4,4\}$
Ed	ϵ	4	$\{2,4,4,4\}$
Fred	ϵ	2	$\{4,4\}$
Greg	ϵ	4	$\{2,2,4,4\}$
Harry	ϵ	2	$\{4,4\}$

(b) vertex refinements

Equivalence Relation	Equivalence Classes
$\equiv_{\mathcal{H}_0}$	$\{A,B,C,D,E,F,G,H\}$
$\equiv_{\mathcal{H}_1}$	$\{A,C\}\ \{B,D,E,G\}\ \{F,H\}$
$\equiv_{\mathcal{H}_2}$	$\{A,C\}\{B\}\{D,E\}\{G\}\{F,H\}$
$\equiv_A$	$\{A,C\}\{B\}\{D,E\}\{G\}\{F,H\}$

(c) equivalence classes

Subgraph Queries

Limitation of vertex refinement:

- always provide complete information about the nodes adjacent to the target (closed-world).
- arbitrarily large subgraphs around a node if that node is highly connected
  E.g.,, if $H_1(x) = 100$ vs $H_1(y) = 2$

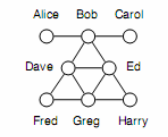
class of queries about the existence of a subgraph around the target node.

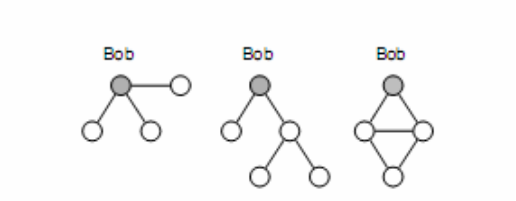
Measure their descriptive power by counting *edge facts* (# edges in the subgraph).

15

Subgraph Queries

Example: Three subgraph queries centered around Bob.





may correspond to different strategies of knowledge acquisition by the adversary.
including breadth-first exploration, induced subgraphs of radius 1 and 2. etc -- For a given number of edge facts, some queries are more effective at distinguishing individuals.

may be incomplete (open-world)

16

Hub Fingeprint Queries

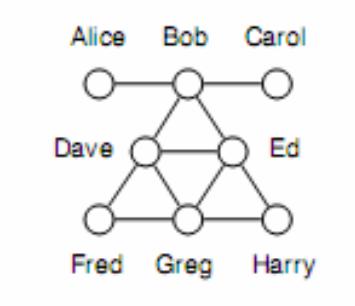
A **hub** is a node with high degree and high betweenness centrality (the proportion of shortest paths in the network that include the node)

A **hub fingerprint** for a target node x is a description of the connections of x to a set of designated hubs in the network.

$F_i(x)$ hub fingerprint of x to a set of designated hubs, where i limit on the maximum distance

17

Hub Fingeprint Queries



Hubs: Dave and Ed
F_1(Fred) = (1; 0)
F_2(Fred) = (1; 2)

both an open and a closed world.

Example:
open world, if the adversary knows F_1(Fred) = (1; 0) then nodes in the anonymized graph with F_1 fingerprints of (1; 0) or (1; 1) are both candidates for Fred.

18

Comparison of the Knowledge Models

Expressiveness:
Vertex refinement queries provide complete information about node degree.
A subgraph query can never express H_i because subgraph queries are existential and cannot assert exact degree constraints or the absence of edges in a graph.

Complexity Computing:
H* is linear in the number of edges,
Subgraph queries can be NP-hard in the number of edge facts, (requires finding all isomorphic subgraphs in the input graph)

Both have well-studied *logical foundations*:
H_i knowledge corresponds to first order logic with counting quantifiers, restricted to i variables.

Subgraph queries can be expressed as conjunctive queries with inequalities. The number of edge facts corresponds to the number of subgoals in the query

19

Disclosure in Real Networks

- Study three networked data sets, drawn from diverse domains.
- For each data set, consider each node in turn as a target.
- Assume the adversary computes a vertex refinement query, a subgraph query, or a hub fingerprint query on that node, and then compute the corresponding candidate set for that node.
- Report the distribution of candidate set sizes across the population of nodes to characterize how many nodes are protected and how many are identifiable.

20

Disclosure in Real Networks

Hep-Th database: papers and authors in theoretical high-energy physics, from the arXiv archive, linked if at least two papers together.

Enron dataset: from a corpus of email sent to and from managers at Enron Corporation -- Two individuals connected if they corresponded at least 5 times.

Net-trace dataset: from an IP-level network trace collected at a major university. monitors traffic at the gateway; a bipartite graph between IP addresses internal to the institution, and external IP addresses.
187 internal addresses from a single campus department and the 4026 external addresses to which at least 20 packets were sent on port 80 (http traffic).

undirected edges, self-loops removed, eliminated a small percentage of disconnected nodes.

21

Reidentification: Vertex Refinement

very low percentage of high-risk nodes under a reasonable assumption about adversary knowledge.
Two datasets meet that requirement for H1 (Hep-Th and Net-trace), but no datasets meet that requirement for H2.

significant variance across different datasets in their vulnerability to different adversary knowledge.

the most significant change in re-identification is from H1 to H2,

Re-identification tends to stabilize after H3 — more information in the form of H4 does not lead to an observable increase in re-identification

a substantial number of nodes are not uniquely identified even with H4

22

Reidentification: Subgraph Queries

disclosure is substantially lower than for vertex refinement queries.
To select candidate sets of size less than 10 requires a subgraph query of size 24 for Hep-Th, size 12 for Enron, and size 32 for Net-trace.

The smallest subgraph query resulting in a unique disclosure was size 36 for Hep-Th and 20 for Enron. The smallest candidate set witnessed for Net-trace was size 2, which resulted from a query consisting of 88 edge facts.

Breadth-first exploration led to selective queries across all three datasets.
asserts lower bounds on the degree of nodes.
In Enron, the most selective subgraph queries witnessed;
for Hep-Th and Net-trace, the more selective subgraph queries asserted the existence of two nodes with a large set of common neighbors.

23

Reidentification: Hub Fingerprints

disclosure is low using hub fingerprints.
At distance 1, 54% of the nodes in Enron were not connected to any hub and therefore hub fingerprints provide no information.
This was 90% for Hepth and 28% for Net-trace.

connectivity to hubs was fairly uniform across individuals.
For example, the space of possible fingerprints at distance 1 for Hepth and Net-trace is 2^{10}= 1024.
Of these, only 23 distinct fingerprints were observed for Hepth and only 46 for Net-trace.

hubs themselves stand out, but have high-degrees, thus connections to a hub are shared by many.

24

Anonymization in Random Graphs

Erdos-Renyi (RE) random graphs
n nodes by sampling each edge independently with probability p
sparce p = c/n, dense = clogn/n, super-dense p = c (c is a constant)

c>1,
include a giant connected component of size Θ(n), and a collection of smaller components (sparse)
completed connected (dense)

25

Reindification in Random Graphs

THEOREM 1 (SPARSE ER RANDOM GRAPHS). *Let G be an ER random graph containing n nodes with edge probability given by $p = c/n$ for $c > 1$. With probability going to one, the expected sizes of the equivalence classes induced by $\mathcal{H}_i$ is $\Theta(n)$, for any $i \geq 0$.*

THEOREM 2 (SUPER-DENSE ER RANDOM GRAPHS). *Let G be an ER random graph on n nodes with edge probability $p = 1/2$. The probability that there exist two nodes $x, y \in V$ such that $x \equiv_{\mathcal{H}_3} y$ is less than 2^{-cn} for constant value $c > 0$.*

For dense, nodes cannot be identified for H_1 for any c>0, but all nodes are re-identifiable for H_2 for any c>1

26

Reindification in Random Graphs

ω(G) the number of nodes in the largest clique

PROPOSITION 2. *Let G be any graph, and $Q(x)$ a subgraph query around any node x. If $Q(x)$ contains fewer than $\omega(G)$ nodes, then $|cand_Q(x)| \geq \omega(G)$.*

Any subgraph query matching fewer than ω(G) nodes, will match any node in the clique

27

Anonymization Algorithms

Partition/Cluster the nodes of Ga into disjoint sets

In the generalized graph,
supernodes: subsets of Va
edges with labels that report the density

Partitions of size at least k

DEFINITION 3 (GENERALIZATION OF GRAPH). *Let $\mathcal{V}$ be the supernodes of V_a. $\mathcal{G}$ is a* generalization of G_a under $\mathcal{V}$ *if, for all* $X, Y \in \mathcal{V}$, $d(X,Y) = |\{(x,y) \in E_a | \, x \in X, y \in Y\}|$.

Extreme cases: a singe super-node with self-loop, Ga

Again: Privacy vs Utility

28

Anonymization Algorithms

Find a partition that best fits the input graph

Estimate fitness via a maximum likelihood approach

Uniform probability distribution over all possible worlds

Searches all possible partitions using simulated annealing

Each valid partitions (minimum partition of at least k nodes) is a valid state

Starting with a single partition with all nodes, propose a change of state:

- split a partition
- merge two partitions, or
- move a node to a different partition

Stop when fewer than 10% of the proposals are accepted

29

Anonymization Algorithms

Next, we see 3 concrete examples:

- Know the degree, and
- Neighborhood,
- Any structural query

30

k-degree Anonymity

K. Liu and E. Terzi, *Towards Identity Anonymization on Graphs,* SIGMOD 2008

31

Identity anonymization on graphs

- Question
  - How to share a network in a manner that permits useful analysis without disclosing the identity of the individuals involved?
- Observations
  - Simply removing the identifying information of the nodes before publishing the actual graph does not guarantee identity anonymization.

    L. Backstrom, C. Dwork, and J. Kleinberg, "Wherefore art thou R3579X?: Anonymized social netwoks, hidden patterns, and structural steganography," In WWW 2007.

    J. Kleinberg, "Challenges in Social Network Data: Processes, Privacy and Paradoxes, " KDD 2007 Keynote Talk.
- Can we borrow ideas from ***k***-anonymity?

32

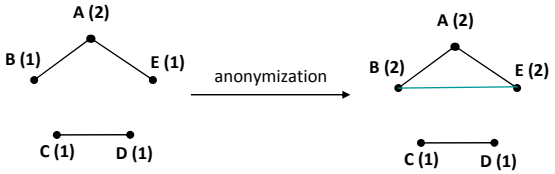
What if you want to prevent the following from happening

- Assume that adversary **A** knows that **B** has 327 connections in a social network!
- If the graph is released by removing the identity of the nodes
  - **A** can find all nodes that have degree 327
  - If there is only one node with degree 327, **A** can identify this node as being **B**.

33

Privacy model

k-degree anonymity A graph ***G(V, E)*** is ***k-degree*** *anonymous* if every node in ***V*** has the same degree as ***k-1*** other nodes in ***V***.



[Properties] It prevents the re-identification of individuals by adversaries with *a priori* knowledge of the degree of certain nodes.

34

Degree-sequence anonymization

[k-anonymous sequence] A sequence of integers *d* is *k-anonymous* if every distinct element value in *d* appears at least *k* times.

[100,100, 100, 98, 98,15,15,15]

A graph G(V, E) is *k*-degree anonymous if its degree sequence is *k*-anonymous

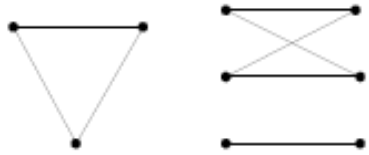


Figure 1: Examples of a 3-degree anonymous graph (left) and a 2-degree anonymous graph (right).

35

Problem Definition

Given a graph *G(V, E)* and an integer *k*, modify G via a set of edge addition or deletion operations to construct a new graph *k-degree anonymous* graph G' in which every node u has the same degree with at least *k*-1 other nodes

Why not simply transform G to the complete graph?

Prop 1: If G is k1-degree anonymous, then it is also k2-degree anonymous, for every k2 ≤ k1

36

Problem Definition

Given a graph *G(V, E)* and an integer *k*, modify *G* via a minimal set of edge addition or deletion operations to construct a new graph *G'(V', E')* such that

1) *G'* is *k*-degree anonymous;

2) *V' = V*;

3) The *symmetric difference* of *G* and *G'* is as small as possible

$$\mathrm{SymDiff}(G', G) = (E' \setminus E) \cup (E \setminus E')$$

Assumption: G: undirected, unlabeled, no self-loops or multiple-edges

Only edge **additions** -- SymDiff(G', G) = |E'| - |E|

There is always a feasible solution (ΠΟΙΑ;)

37

Degree-sequence anonymization

Increase/decrease of degrees correspond to additions/deletions of edges

[degree-sequence anonymization] Given degree sequence *d*, and integer *k*, construct *k*-anonymous sequence *d'* such that ***||d'-d||*** (i.e., $L_1(d' - d)$) **is minimized**

$|E'| - |E| = \frac{1}{2} L_1(d' - d)$

Relax graph anonymization: E' not a supergraph of E

38

Graph Anonymization algorithm

Two steps

Input: Graph ***G*** with degree sequence ***d***, integer ***k***
Output: ***k***-degree anonymous graph ***G'***

[STEP 1: **Degree Sequence Anonymization**]:
Construct an (optimal) k-anonymous degree sequence ***d'*** from the original degree sequence ***d***

[STEP 2: **Graph Construction**]:
[**Construct**]: Given degree sequence ***d'***, construct a new graph $G^0(V, E^0)$ such that the degree sequence of G^0 is ***d'***
[***Transform***]: Transform $G^0(V, E^0)$ to $G'(V, E')$ so that ***SymDiff***(G',G) is minimized.

39

DP for degree-sequence anonymization

$d(1) \geq d(2) \geq \ldots \geq d(i) \geq \ldots \geq d(n)$: original degree sequence.
$d'(1) \geq d'(2) \geq \ldots \geq d'(i) \geq \ldots \geq d'(n)$: k-anonymized degree sequence.

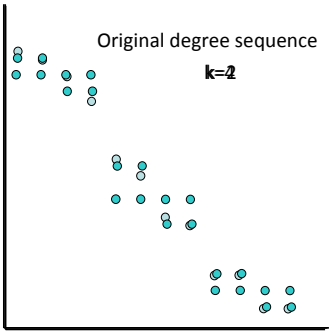
If we only add edges, d'(i) ≥ d(i)
Observation 1, if d'(i) = d'(j) with i < j, then d'(i) = d'(i+1) = .. . d'(j-1) = d(j)

I(i, j): anonymization cost when all nodes i, i+1, …, j are put in the same anonymized group

$$I(i, j) = \sum_{\ell=i}^{j} \left(d(i) - d^*\right)$$

40

Algorithm for degree-sequence anonymization



41

DP for degree-sequence anonymization

DA(1, j): the optimal degree anonymization of subsequence d(1, j)

DA(1, n): the optimal degree-sequence anonymization cost

I(i, j): anonymization cost when all nodes i, i+1, …, j are put in the same anonymized group

For i < 2k (impossible to construct 2 different groups of size k)

$$DA(1,i) = I(1,i)$$

For i ≥ 2k

$$DA(1,i) = \min\ \{\min_{k \le t \le i-k} \{DA(1,t) + I(t+1,i)\}, I(1,i)\}$$

42

DP for degree-sequence anonymization

$$DA(1,i) = I(1,i)$$

$$DA(1,i) = \min\ \{\min_{k \le t \le i-k} \{DA(1,t) + I(t+1,i)\}, I(1,i)\}$$

Can be improved, no anonymous groups should be of size larger than 2k-1

We do not have to consider all the combinations of I(i, j) pairs, but for every i, only j's such that $k \le j - i + 1 \le 2k-1$

$O(n^2)$ -> (Onk)

$$DA(1,i) = \min_{\max\{k, i-2k+1\} \le t \le i-k} \{DA(1,t) + I(t+1,i)\}$$

Additional bookkeeping -> Dynamic Programming with **O(nk)**

Greedy

Form a group with the first k, for the k+1, consider

$C_{merge} = (d(1) - d(k+1)) + I(k+2, 2k+1) - C_{new}(k+1, 2k)$

43

GraphAnonymization algorithm

Input: Graph ***G*** with degree sequence ***d***, integer ***k***

Output: ***k***-degree anonymous graph ***G'***

[**Degree Sequence Anonymization**]:

- Contruct an anonymized degree sequence ***d'*** from the original degree sequence ***d***

[Graph Construction]:

[**Construct**]: Given degree sequence ***d'***, construct a new graph $G^0(V, E^0)$ such that the degree sequence of G^0 is ***d'***

[**Transform**]: Transform $G^0(V, E^0)$ to $G'(V, E')$ so that ***SymDiff***(G', G) is minimized.

44

Are all degree sequences realizable?

- A degree sequence ***d*** is **realizable** if there exists a simple undirected graph with nodes having degree sequence ***d.***
- Not all vectors of integers are realizable degree sequences
  - d = {4,2,2,2,1} ?
- How can we decide?

45

Realizability of degree sequences

[**Erdös and Gallai**] A degree sequence ***d*** with ***d***(*1*) ≥ ***d***(*2*) ≥… ≥ ***d***(*i*) ≥… ≥ ***d***(*n*) and Σ***d****(i)* even, is realizable if and only if

$$\sum_{i=1}^{l} \boldsymbol{d}(i) \le l(l-1) + \sum_{i=l+1}^{n} \min\{l, \boldsymbol{d}(i)\}, \text{ for every } 1 \le l \le n-1.$$

For each subset of the *l* highest degree nodes, the degrees of these nodes can be "absorbed" within the nodes and the outside degrees

46

Realizability of degree sequences

Input: Degree sequence ***d'***
Output: Graph ***$G^0(V, E^0)$*** with degree sequence ***d'*** or ***NO!***

General algorithm, create a graph with degree sequence d'

Algorithm 1 The `ConstructGraph` algorithm.

Input: A degree sequence $\mathbf{d}$ of length n.
Output: A graph $G(V, E)$ with nodes having degree sequence $\mathbf{d}$ or "No" if the input sequence is not realizable.

$V \leftarrow \{1, \ldots, n\}$, $E \leftarrow \emptyset$
if $\sum_i \mathbf{d}(i)$ is odd **then**
  Halt and return "No"
while 1 **do**
  if there exists $\mathbf{d}(i)$ such that $\mathbf{d}(i) < 0$ **then**
    Halt and return "No"
  if the sequence $\mathbf{d}$ are all zeros **then**
    Halt and return $G(V, E)$
  Pick a random node v with $\mathbf{d}(v) > 0$
  Set $\mathbf{d}(v) = 0$
  $V_{\mathbf{d}(v)} \leftarrow$ the $\mathbf{d}(v)$-highest entries in $\mathbf{d}$ (other than v)
  for each node $w \in V_{\mathbf{d}(v)}$ **do**
    $E \leftarrow E \cup (v, w)$
    $\mathbf{d}(w) \leftarrow \mathbf{d}(w) - 1$

In each iteration,

pick an arbitrary node u

add edges from u to d(u) nodes of highest residual degree, where d(u) is the residual degree of u

Is an oracle

We also need G' such that E' ⊇ E

Thus, we start with the edges of E already in

Is not an oracle

47

Realizability of degree sequences

Input: Degree sequence ***d'***
Output: Graph ***$G^0(V, E^0)$*** with degree sequence ***d'*** or ***NO!***

→If the degree sequence ***d'*** is NOT realizable?

- Convert it into a realizable and ***k***-anonymous degree sequence

Algorithm 2 The `Probing` scheme.

Input: Input graph $G(V, E)$ with degree distribution $\mathbf{d}$ and integer k.
Output: Graph $\hat{G}(V, \hat{E})$ with k-anonymous degree sequence $\hat{\mathbf{d}}$, such that $E \subseteq \hat{E}$.

$\hat{\mathbf{d}} = \texttt{DP}(\mathbf{d})$ /* or $\texttt{Greedy}(\mathbf{d})$ */
$\left(\text{realizable}, \hat{G}\right) = \texttt{Supergraph}(\hat{\mathbf{d}})$
while realizable = "No" or "Unknown" **do**
  $\mathbf{d} = \mathbf{d} + \text{random\_noise}$
  $\hat{\mathbf{d}} = \texttt{DP}(\mathbf{d})$ /*or $\texttt{Greedy}(\mathbf{d})$ */
  $\left(\text{realizable}, \hat{G}\right) = \texttt{Supergraph}(\hat{\mathbf{d}})$
Return $\hat{G}$

Slightly increase some of the entries in d via the addition of uniform noise

In the implementation, examine the nodes in increasing order of their degrees, and slightly increase the degrees of a single node at each iteration (in real graph, few high degree nodes – rarely any two of these exactly the same degree)

48

GraphAnonymization algorithm

Input: Graph ***G*** with degree sequence ***d***, integer ***k***
Output: ***k***-degree anonymous graph ***G'***

[**Degree Sequence Anonymization**]:

- Contruct an anonymized degree sequence ***d'*** from the original degree sequence ***d***

[**Graph Construction**]:

[**Construct**]: Given degree sequence ***d'***, construct a new graph $G^0(V, E^0)$ such that the degree sequence of G^0 is ***d'***

[**Transform**]: Transform $G^0(V, E^0)$ to $G'(V, E')$ so that ***SymDiff***(G',G) is minimized.

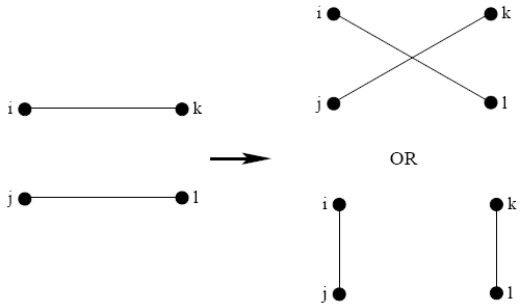
49

Graph-transformation algorithm

- GreedySwap transforms $G^0 = (V, E^0)$ into $G'(V, E')$ with the same degree sequence ***d'***, and min symmetric difference ***SymDiff(G',G)*** .
- GreedySwap is a greedy heuristic with several iterations.
- At each step, GreedySwap swaps a pair of edges to make the graph more similar to the original graph ***G,*** while leaving the nodes' degrees intact.

50

Valid swappable pairs of edges



A swap is ***valid*** if the resulting graph is simple

51

GreedySwap algorithm

Input: A pliable graph $G^0(V, E^0)$, fixed graph $G(V,E)$

Output: Graph $G'(V, E')$ with the same degree sequence as $G^0(V,E^0)$

i=0

Repeat

find the valid swap in G^i that most reduces its symmetric difference with G , and form graph G^{i+1}

i++

52

Experiments

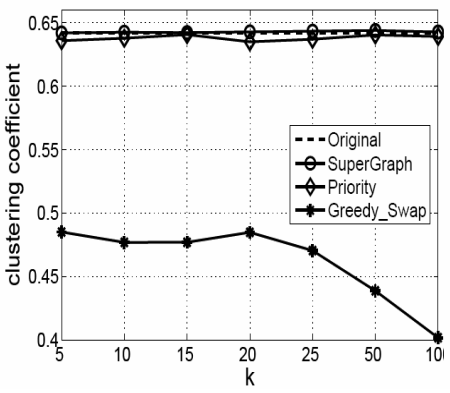
- **Datasets:**
  - Co-authors (7995 authors of papers in db and theory conference),
  - Enron emails (151 users, edge if at least 5 times),
  - powergrid (generators, transformers and substations in a powergrid network, edges represent high-voltage transmission lines between them),
  - Erdos-Renyi (random graphs with nodes randomly connected to each other with probability p),
  - small-world large clustering coefficient (average fraction of pair of neighbors of a node that are also neighbors) and small average path length (average length of the shortest path between all pairs of reachable nodes),
  - power-law or scale graphs (the probability that a node has degree d is proportional to $d^{-\gamma}$, $\gamma = 2, 3$)

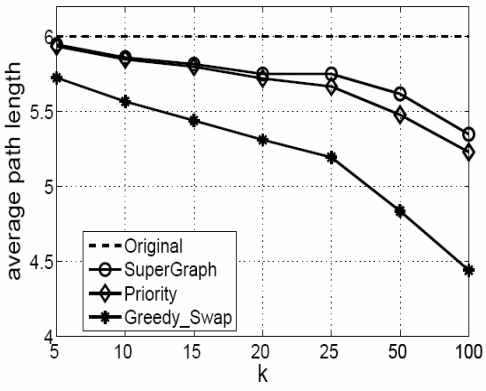
- **Goal (Utility):** degree-anonymization does not destroy the structure of the graph
  - Average path length
  - Clustering coefficient
  - Exponent of power-law distribution

53

Experiments: Clustering coefficient and Avg Path Length

- **Co-author** dataset
- APL and CC do not change dramatically even for large values of ***k***





Experiments: Edge intersections

Edge intersection achieved by the GreedySwap algorithm for different datasets.

Parenthesis value indicates the original value of edge intersection

Synthetic datasets	
Small world graphs*	0.99 (0.01)
Random graphs	0.99 (0.01)
Power law graphs**	0.93 (0.04)
Real datasets	
Enron	0.95 (0.16)
Powergrid	0.97 (0.01)
Co-authors	0.91(0.01)

(*) L. Barabasi and R. Albert: Emergence of scaling in random networks. *Science 1999.*

(**) Watts, D. J. Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology* 1999

55

Experiments: Exponent of power law distributions

Original	2.07
k=10	2.45
k=15	2.33
k=20	2.28
k=25	2.25
k=50	2.05
k=100	1.92

Co-author dataset

Exponent of the power-law distribution as a function of ***k***

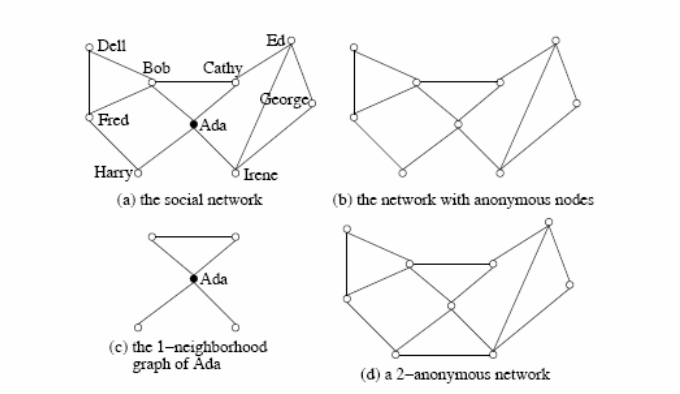
56

k-neighborhood Anonymity

B. Zhou and J. Pei, *Preserving Privacy in Social Networks Against Neighborhood Attacks,* ICDE 2008

57

Motivation



An adversary knows that:
Ada has two friends who know each other, and has another two friends who do not know each other (1-neighborhood graph)
Similarly, Bob can be identified if the adversary knows its 1-neighborhood graph

58

1-neighborhood attacks

The neighborhood of $u \in V(G)$ is the induced subgraph of the neighbors of u, denoted by $Neighbor_G(U) = G(N_u)$ where $N_u = \{v \mid (u,v) \in E(G)\}$.

59

Graph Model

Graph G= (V, E, L, F),
V is a set of vertices,
$E \subseteq V \times V$ is a set of edges,
L is a set of labels, and
F a labeling function $F: V \rightarrow L$ assigns each vertex a label.

edges do not carry labels

Items in L form a hierarchy.
E.g., if occupations are used as labels of vertices, L contains not only the specific occupations [such as dentist, general physician, optometrist, high school teacher, primary school teacher, etc] but also general categories [such as, medical doctor, teacher, and professional}.

$* \in L$ -> most general category generalizing all labels.

60

Graph Model

Given a graph = (V_H, E_H, L, F) and a social network G = (V, E, L, L), an instance of H in G is a tuple (H', f) where H' = ($V_{H'}$,$E_{H'}$,L, F) is a subgraph in G and f: $V_H \rightarrow V_{H'}$, is a bijection function such that

(1) for any $u \in V_H$, $F(f(u)) \leq F(u)$, /* the corresponding labels in H' are more general */ and

(2) $(u, v) \in E_H$ if and only if $(f(u), f(v)) \in E_{H'}$.

61

G -> G' through a bijection (isomorphism) A

[*k*-neighborhood anonymity] A vertex $u \in V(G)$, u is k anonymous in G' if there are at least (k − 1) other vertices $u_1, \ldots, u_{k-1} \in V(G)$ such that $Neighbor_{G'}(A(u))$, $Neighbor_{G'}(A(u_1))$, . . ., $Neighbor_{G'}(A(u_{k-1}))$ are isomorphic.

G' is k-anonymous if every vertex in G' is k-anonymous.

Property 1 (k-anonymity) Let G be a social network and G' an anonymization of G. If G' is k-anonymous, then with the neighborhood background knowledge, any vertex in G cannot be re-identified in G' with confidence larger than 1/k .

62

Given a social network G, the k-anonymity problem is to compute an anonymization G′ such that

(1) G′ is *k*-anonymous;
(2) each vertex in G is anonymized to a vertex in G′ and G′ does not contain any fake vertex; (no node addition)
(3) every edge in G is retained in G′; and (no node deletion)
(4) the number of edges to be added is minimized.

63

Utility

Aggregate queries:
compute the aggregate on some paths or subsgraphs satisfying some given conditions
E.g., Average distance from a medical doctor to a teacher

Heuristically, when the number of edges added is as small as possible, G′ can be used to answer aggregate network queries accurately

64

Anonymization Method

Two steps:

STEP 1
Extract the neighborhoods of all vertices in the network
Encode the neighborhood of each node (to facilitate the comparison between neigborhoods)

STEP 2
Greedily, organize vertices into groups and anonymize the neighborhoods of vertices in the same group

65

Step 1: Neighborhood Extraction and Coding

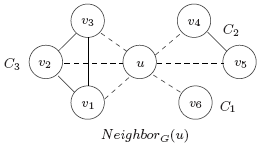
General problem of determining whether two graphs are isomorphic is NP-complete

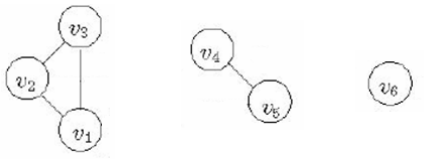
Goal: Find a coding technique for neighborhood subgraphs so that whether two neighborhoods are isomorphic can be determined by the corresponding encodings

66

Step 1: Neighborhood Extraction and Coding

A subgraph C of G is a **neighborhood component** of u ∈ V (G), if C is a maximal connected subgraph in $Neighbor_G(u)$.





Neighborhood components of u

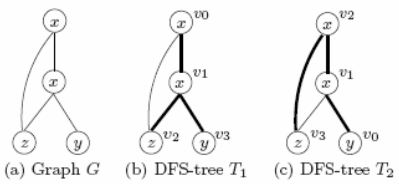
- Divide the neighborhood of v into neighborhood components
- To code the whole neighborhood, first code each component.

67

Step 1: Neighborhood Extraction and Coding

Encode the edges and vertices in a graph based on its depth-first search tree (DFS-tree).

All the vertices in G can be encoded in the pre-order of T .



Thick edges are those in the DFS-trees (forward edges),
Thin edges are those not in the DFS-trees (backward edges)

vertices encoded u0 to u3 according to the pre-order of the corresponding DFS-trees.

The DFS-tree is generally not unique for a graph -> minimum DFS code (based on an ordering of edges) – select the lexically minimum DFS code – DFS(G)

- Two graphs G and G' are isomorphic, if and only if, DFS(G) = DFS(G')

68

Step 1: Neighborhood Extraction and Coding

Combine the code of each component to produce a single code for the neighborhood

The neighborhood component code of NeighborG(u) is a vector NCC(u) = (DFS(C_1)}.... DFS(C_m)) where C_1,...,C_m
are the neighborhood components of NeighborG(U), where components are ordered

Theorem (Neighborhood component code): For two vertices u, v $\in$ V(G) where G is a social network, $Neighbor_G(u)$ and $Neighbor_G(v)$ are isomorphic if and only if NCC(u) = NCC(v).

69

Step 2: Social Network Anonymization

Each vertex must be grouped with a least (k-1) other vertices such their anonymized neighborhoods are isomorphic

For a group S with the same neighborhoods, all vertices in S have the same degree

Vary few nodes have high degrees, process them first to keep information loss for them low

Many vertices of low degree, easier to anonymize

1. Define Quality Measures
2. Anonymize Two Neighborhoods
3. Anonymize a Social Network

70

Step 2: Quality Measures

Generalize vertex labels

l_1 (leaf level)-> more general l_2 (penalty or loss as in relational) size(*)= #leafs

$$NCP(l_2) = \frac{size(l_2)}{size(*)}$$

Add Edges

Total number of edges added +

Number of vertices that are not in the neighborhood of the target vertex and are linked for anonymization

$$Cost(u,v) = \alpha \cdot \sum_{v' \in H'} NCP(v') + \beta \cdot |\{(v_1,v_2)|(v_1,v_2) \notin E(H), (\mathcal{A}(v_1),\mathcal{A}(v_2)) \in E(H')\}| + \gamma \cdot (|V(H')| - |V(H)|)$$

71

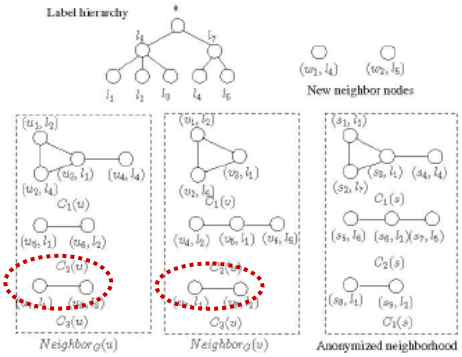
Step 2: Anonymizing 2 neighborhoods

First, find all perfect matches of neighborhood components (perfectly match=same minimum DFS code)

For unmatched, try to pair "similar" components and anonymize them

How: greedily, starting with two vertices with the same degree and label in the two components to be matched (if ties, start from the one with the highest degreeIf there are no such vertices: choose the one with minimum cost

Then a BFS to match vertices one by one, if we need to add a vertex, consider vertices in V(G)



72

Step 2: Social Network Anonymization

Input: a social network $G = (V, E)$, the anonymization requirement parameter k, the cost function parameters α, β and γ;
Output: an anonymized graph G';
Method:
1: initialize $G' = G$;
2: mark $v_i \in V(G)$ as "unanonymized";
3: sort $v_i \in V(G)$ as *VertexList* in neighborhood size descending order;
4: WHILE (*VertexList* $\neq \emptyset$) DO
5: let *SeedVertex* = *VertexList.head*() and remove it from *VertexList*;
6: FOR each $v_i \in$ *VertexList* DO
7: calculate *Cost*(*SeedVertex*, v_i) using the anonymization method for two vertices;
END FOR
8: IF (*VertexList.size*() $\geq 2k - 1$) DO
let *CandidateSet* contain the top $k - 1$ vertices with the smallest *Cost*;
9: ELSE
10: let *CandidateSet* contain the remaining unanonymized vertices;
11: suppose *CandidateSet*= $\{u_1, \ldots, u_m\}$, anonymize *Neighbor*(*SeedVertex*) and *Neighbor*(u_1) as discussed in Section III-B.2;
12: FOR $j = 2$ to m DO
13: anonymize *Neighbor*(u_j) and {*Neighbor*(*SeedVertex*), *Neighbor*(u_1), . . . ,*Neighbor*(u_{j-1})} as discussed in Section III-B.2, mark them as "anonymized";
14: update *VertexList*;
END FOR
END WHILE

Fig. 5. Anonymizing a Social Network.

Maintain a list VertexList of unanonymized vertices in descending order of neighborhood size

73

Co-authorship data from KDD Cup 2003 (from arXiv, high-energy physics)
Edge – co-authored at least one paper in the data set.
57,448 vertices
120,640 edges
average number of vertex degrees about 4.

k	Removing labels	Generalizing to affiliations
5	1.3%	12.7%
10	3.9%	16.1%
15	7.1%	19.4%
20	12.0%	23.2%

TABLE I
THE PERCENTAGES OF VERTICES VIOLATING k-ANONYMITY IN THE CO-AUTHORSHIP DATA.

74

3-level anonymization, author, affiliations-countries, *

Anonymized for different k

Aggregate queries: the average distance from vertex with level l1 to each nearest neighbor with label l2

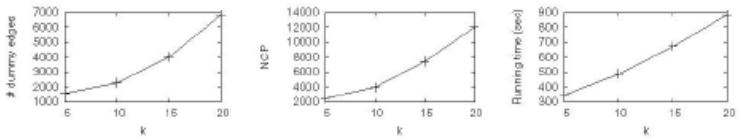


Fig. 9. Anonymizing the KDD cup 2003 co-authorship data set.

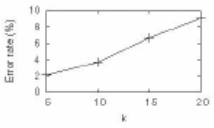


Fig. 10. Query answering on the KDD Cup 2003 co-authorship data set.

For 10 random label pairs

75

k-Automorphism

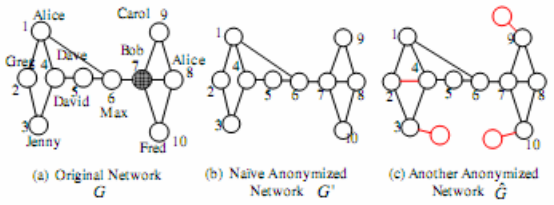
L. Zhu, L. Chen and M. Tamer Ozsu, *k-automorphism: a general framework for privacy preserving network publication,* PVLDB 2009

76

K-Automorphism

Considers any subgraph query - any structural attack

At least k symmetric vertices no structural differences

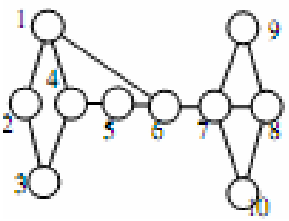


77

K-Automorphism

DEFINITION 2.1. ***Graph Isomorphism***. *Given two graphs* $Q = \langle V_Q, E_Q \rangle$ *and* $G = \langle V_G, E_G \rangle$, *Q is isomorphic to G, if and only if there exists at least one bijective function* $f: V_Q \rightarrow V_G$ *such that for any edge* $(u, v) \in E_Q$, *there is an edge* $(f(u), f(v)) \in E_G$.

DEFINITION 2.2. ***Graph Automorphism.*** *An automorphism of a graph* $G = \langle V, E \rangle$ *is an automorphic function* f *of the vertex set V, such that for any edge* $e = (u, v)$, $f(e) = (f(u), f(v))$ *is also an edge in G, i.e., it is a graph automorphism from G to itself under function* f. *If there exist k automorphisms in G, it means that there exists k-1 different automorphic functions.*



map each node of graph G to (another) node of graph G

78

K-Automorphism

DEFINITION 3.1. ***K-automorphic Network.*** *Given a network G, (a) if there exist k-1 automorphic functions F_a (a=1,...,k-1) in G, and (b) for each vertex v in G, $F_{a_1}(v) \neq F_{a_2}(v)$ $(1 \leq a_1 \neq a_2 \leq k-1)$, then G is called a* k-automorphic network.

any k-1 automorphic functions?

79

K-Automorphism

DEFINITION 3.2. ***Different Matches.*** *Given a sub-graph query Q and two matches m_1 and m_2 of Q in a social network G', where m_1 and m_2 are isomorphic to Q under functions f_1 and f_2, respectively, if there exists no vertex v (in query Q) whose match vertices in m_1 and m_2 are identical, (i.e. $f_1(v) = f_2(v)$), we say that m_1 and m_2 are* different matches.

DEFINITION 3.3. ***k-different match principle.*** *Given a released network G^* and any sub-graph query Q, if (a) there exist at least k matches of Q in G^*, and (b) any two of the k matches are different matches according to Definition 3.2, then G^* is said to obey* k-different match principle.

80

K-Automorphism: Cost

DEFINITION 2.6. ***Anonymization Cost.*** *Given an original network G and its anonymized version G^*, the anonymization cost in G^* is defined as*

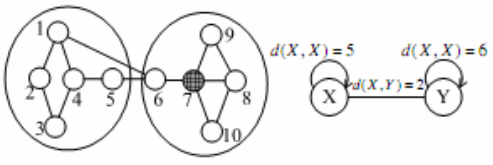
$$Cost(G, G^*) = (E(G) \cup E(G^*)) - (E(G) \cap E(G^*))$$

where $E(G)$ is the set of edges in G.

81

K-Automorphism: Algorithm

compare with Hay et al

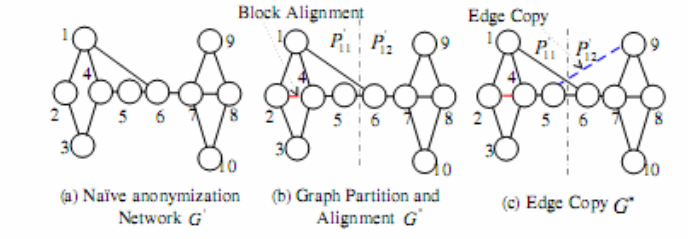


(a) Naïve Anonymization Network G'

(b) Generalized Network

82

K-Match (KM) Algorithm



Step 1: Partition the original network into blocks

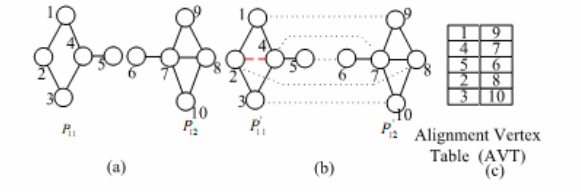
Step 2: Align the blocks to attain isomorphic blocks (add edge (2,4))

Step 3: Apply the "edge-copy" technique to handle matches that cross the two blocks

K-Match (KM) Algorithm: Alignment

DEFINITION 4.1. ***Alignment Vertex Instance.*** *Given a group U_i with blocks P_{ij}, $j = 1, \ldots, k$, assume that alignment blocks $P'_{ij} = (V'_{ij}, E'_{ij})$ are the blocks obtained after graph alignment, namely, $\forall j$ P_{ij} is a sub-graph of P'_{ij} and all P'_{ij} are isomorphic to each other.*

Due to graph isomorphism, given an alignment block P'_{ij}, for each vertex v in P'_{ij}, there must exist $k-1$ symmetric vertices in the other $k-1$ blocks respectively. The set containing v and v's symmetric vertices form alignment vertex instance *I where $|I| = k$. All alignment vertex instances are collected to form* alignment vertex table *(AVT).*



K-Match (KM) Algorithm: Alignment

Heuristic for finding a good alignment

Algorithm 3 constructAVT(U_i), where $j = 1, ..., k$: Built AVT for a group with k blocks P_{ij}, where $j = 1, ..., k$

1: Set all vertices in each P_{ij} as "un-visited", initialize AVT
2: Find v_{ij} in each block P_{ij}, where all $degree(v_{ij}) = d$. If there are multiple choices for d, choose d with the largest value. If there are no choices for d, choose v_{ij} with the largest degree from block P_{ij} respectively
3: The set of all v_{ij} form the initial alignment vertex I instance in AVT.
4: Perform breath-first search (BFS) starting from v_{ij} in each P_{ij} in parallel.
5: During BFS, k vertices from k blocks with similar vertex degrees are collected to form an alignment vertex instance in AVT.
6: Report AVT.

Find k vertices with the same vertex degree

If many, start with those with high degree

If none, choose the one with the largest degree

This set -> initial alignment

BFS in each block in parallel,

pairing nodes with similar degree (if there is no corresponding vertex, introduce dummy with the same label as the corresponding)

85

K-Match (KM) Algorithm: Edge Copy

DEFINITION 4.4. ***Boundary Vertex*** *and* ***Crossing Edge.*** *Given a vertex v in a block P, v is a* boundary vertex *if and only if v has at least one neighbor vertex that is outside of block P. An edge $e = (v, u)$ is called a* crossing edge *if and only if v and u are boundary vertices in two different blocks.*

Algorithm 4 Edge Copy Algorithm

Require: **Input**: The original network: G; The network after graph partition and block alignment: G''; Alignment Vertex Table: AVT
Output: The anonymized network G^*.
1: Duplicate G'' into G^* and remove all crossing edges in G^*.
2: **for** each crossing edge (v, u) in the original network G **do**
3: Add edge (v, u) and $(F_a(v), F_a(u))$ $(a = 1, ..., k-1)$ into G^*.
4: Report G^* as release network.

Duplicate all crossing edges using the AVT

86

K-Match (KM) Algorithm: Graph Partitioning

How many blocks to add a small number of edges?

Few -> fewer crossing edges, but larger groups (more edges for aligning)

NP complete -> heuristics

DEFINITION 4.5. *Given a group U_i with blocks P_{ij}, $j = 1, \ldots, k$, anonymization cost of group U_i is defined as follows:*

$$Cost(U_i) = AlCost(U_i) + 0.5 * (k-1) * \sum_{j-1}^{k} |CrossEdge(P_{ij})|$$

where $AlCost(U)$ is defined in Definition 4.2 and $|CrossEdge(P_{ij})|$ is the number of crossing edges associated with block P_{ij}.

DEFINITION 4.2. ***Alignment Cost.*** *Given a group U_i with blocks P_{ij}, $j = 1, \ldots, k$, assume that P'_{ij} are the blocks obtained after block alignment. The cost of block alignment in group U_i is defined as follows:*

$$AlCost(U_i) = \sum_{j-1}^{k} Min(EditDist(P_{ij}, P'_{ij}))$$

where $EditDist(P_{ij}, P'_{ij})$ is defined as the number of graph edit operations (insert vertex/edge, delete vertex/edge) required to transform P_{ij} into P'_{ij}.

87

K-Match (KM) Algorithm: Graph Partitioning

THEOREM 4.2. *Assume that a network G is partitioned into n blocks that are clustered into m groups U_i, where each group U_i has k blocks. Let G^* be an anonymized network produced by KM algorithm. Then*

$$Cost(G, G^*) = \sum_{i-1}^{m} Cost(U_i)$$

where $Cost(G, G^*)$ and $Cost(U_i)$ are defined in Definitions 2.6 and 4.5, respectively.

88

K-Match (KM) Algorithm: Graph Partitioning

Find all frequent subgraphs (first group!)

Try to expand them until the cost becomes worst, in which case start a new group

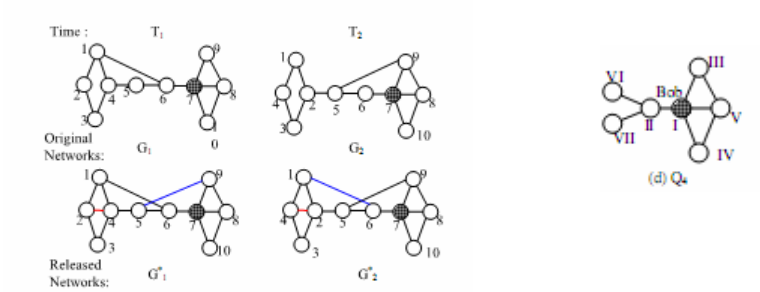
Algorithm 5 Graph Partitioning and Block Clustering

Require: **Input**: The naive anonymized G' and k.
Output: a set of groups $S = \{U_i\}, (i = 1, ..., m)$, where each group U_i has k blocks $P_{ij}, (j = 1, ..., k)$.

```
repeat
    Find frequent sub-graphs {g_f} in G' by setting minimal support
    min_sup = k. Find the frequent sub-graph g_f with the largest
    number of edges. Each match of g_f is extracted from G' as one
    block P_ij.
    The set of all blocks P_ij from one group U'_i.
    repeat
        set U_i = U'_i.
        for each block P_ij in U_i do
            Expand block P_ij by one hop.
        The set of expanded blocks form group U'_i.
    until Cost(U_i) < Cost(U'_i)
    G' = G' − U_i, and insert U_i = {P_ij} into answer set S.
until |E(G)|=0
Report S = {U_i}, i = 1, ..., m.
```


Dynamic Releases

Example:
Individually satisfy 2-automorphism
Assume that an adversary knows that sub-graph Q4 exists around target Bob at both time T1 and T2.
At time T1, an adversary knows that there are two candidates vertices (2, 7)
Similarly, at time T2, there are still two candidates (4, 7)
Since Bob exists at both T1 and T2, vertex 7 corresponds to Bob



Dynamic Releases

Remove all vertex IDs, or permute vertex IDs randomly (so, a given vertexID does not correspond to the same entity in different publications).
Impossible to conduct proper data analysis.
Instead, **vertex ID generalization**

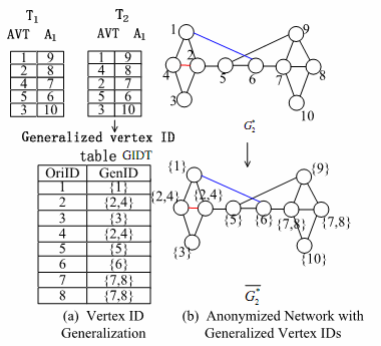
For simplicity, no vertex insertions or deletions in different releases (set of all vertex IDs remains unchanged)

91

Vertex ID Generalization

Given a series of s publications, vertex *v* cannot be identified with a probability higher than 1/k if:

$$Res(v, G_1^*) \cap Res(v, G_2^*) \cap \ldots \cap Res(v, G_s^*) = Res(v, G_1^*)$$
$$where\ |Res(v, G_1^*)| = k.$$



$$Res(7, G_1^*) = \{4, 7\} \text{ and } Res(7, G_2^*) = \{2, 7\}$$
$$Res(7, G_1^*) \cap Res(7, G_2^*) = Res(7, G_1^*) = \{4, 7\}$$
$$2.GenID = \{2, 4\}$$

92

Vertex ID Generalization: Algorithm

Algorithm 6 Generalize Vertex ID For Released Network G_t^*

Require: **Input**: AVT A_1 for the network G_1^*, and AVT A_t for the network G_t^*.
Output: The anonymized network after vertex ID generalization: $\overline{G_t^*}$.

1: Initialize table GIDT.
2: Based on A_1, define $k-1$ automorphic functions F_a^1 in G_1^*, $a = 1, ..., k-1$.
3: Based on A_t, define $k-1$ automorphic functions F_a^t in G_2^*, $t = 1, ..., k-1$.
4: **for** each vertex v in G_t^* **do**
5: **for** $a = 1, ..., k-1$ **do**
6: **if** $F_a^1(v) \neq F_a^t(v)$ **then**
7: Insert $F_a^1(v)$ into $F_a^t(v).GenID$.
8: **for** each vertex v in G_t^* **do**
9: Replace $v.OriID$ by its generalized vertex ID $v.GenID$.
10: Report $\overline{G_t^*}$.

Vertex ID Generalization: Cost

DEFINITION 5.2. *Given a released network $\overline{G_t^*}$ produced by* GenID algorithm, average generalized vertex ID size, *denoted by* $AvgIDSize(\overline{G_t^*})$, *is defined as follows:*

$$AvgIDSize(\overline{G_t^*}) = \frac{\sum_{v \in V(\overline{G_t^*})} |v.GenID|}{|V(\overline{G_t^*})|}$$

where $V(\overline{G_t^*})$ is the set of vertices in $\overline{G_t^*}$.

Vertex Insertion and Deletion

(Deletion) There is a vertex ID v that exists in G'_1 but not in G'_t
Find an arbitrary vertex ID u that exists in both
Insert v in the generalized vertex ID of u

(Insertion) There is a vertex ID v that exists in G'_t but not in G'_1
Assume that instance I contains v in AVT A_t
For each vertex u in I, insert v in the generalized vertex ID of u

95

Evaluation

Prefuse (129 nodes, 161 edges)
Co-author graph (7995 authors in database and theory, 10055 edges)

Synthetic
Erdos Renyi 1000 nodes
Scale free, $2 < \gamma < 3$

All k = 10 degree anonymous, but no sub-graph anonymous

96

Questions?

97