Ragini Gupta
CS 563
(In-class presentation)

Papers-

1. Analyzing Graphs with Node Differential Privacy
2. "Generating Synthetic Decentralized Social Graphs with Local Differential Privacy"

Paper-1

Analyzing Graphs with Node Differential Privacy

Venue: TCC'13

Introduction

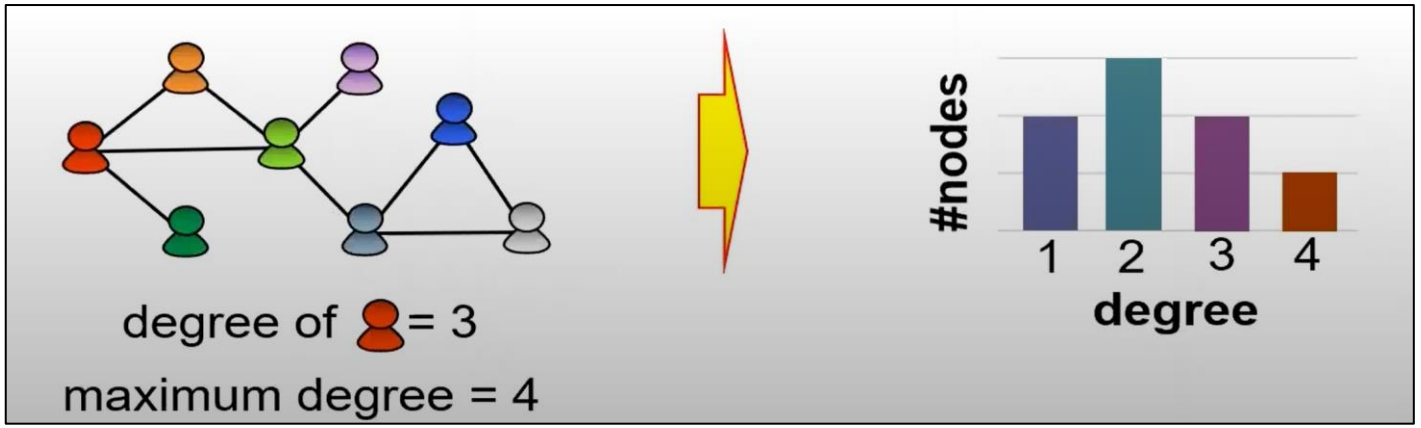
Many types of data can be represented as graphs

- Friendship in online social networks
- Financial transactions
- Email communications
- Health Network
- Relationships

Outline

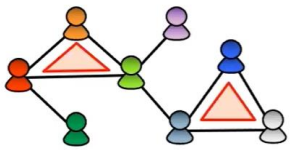
- Graph Statistics
  - Is important to understand connection patterns in social network graph

- E.g. Degree Distribution
  - Degree = # Edges connected to a node
  - Degree Distribution = Distribution of friends in a social network



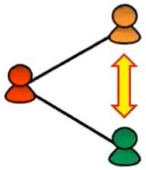
Outline

- E.g. Subgraph Counts
  - Triangle is a set of 3 nodes with 3 edges
  - K-star consists of a central node connected to k other nodes



Shape	Name	Count
	Triangle	2
	2-star	15
	3-star	6

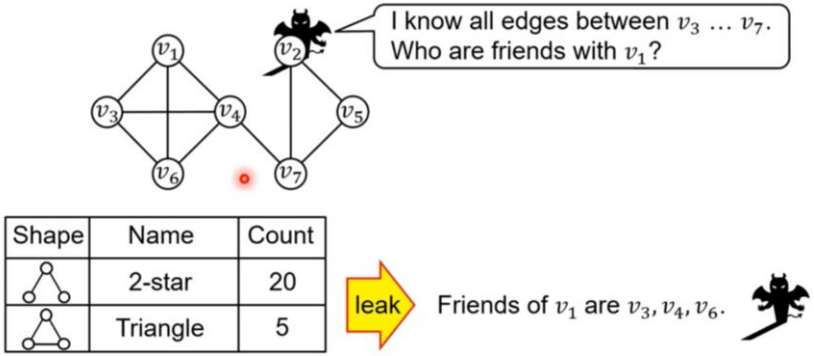
- E.g. Clustering Coefficient
  - Probability that two friends of a user will also be friends
  - = 3 * #triangles/ #2-stars (40% in the above graph)



Will be a friend (after friend suggestion)?

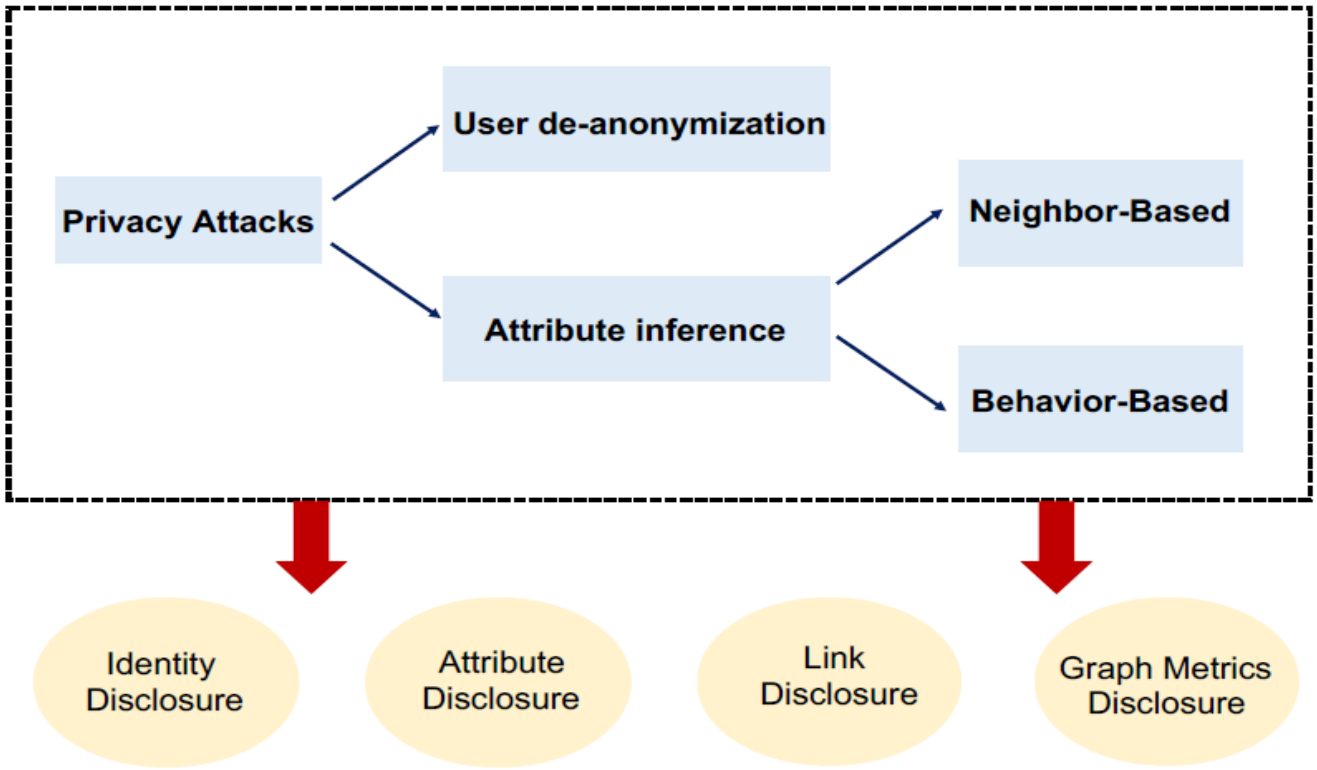
Challenges

- **Privacy Issues:**
  - Subgraph counts (triangles/k-stars, cliques) can reveal sensitive friendship information
  - Suppose V_2 is an honest but curious adversary

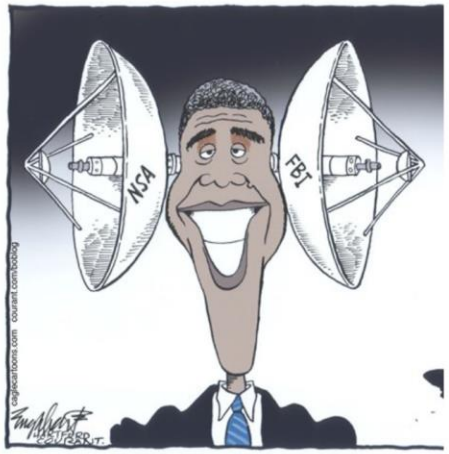


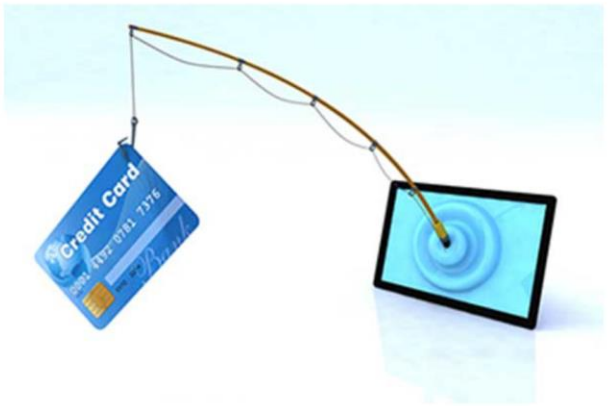
  - We need to obfuscate subgraph count to protect user privacy

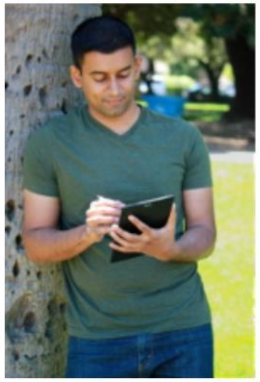
Challenges


Privacy Attacks
User de-anonymization
Attribute inference
Neighbor-Based
Behavior-Based
Identity Disclosure
Attribute Disclosure
Link Disclosure
Graph Metrics Disclosure

Who'd want to de-anonymize social network graph?



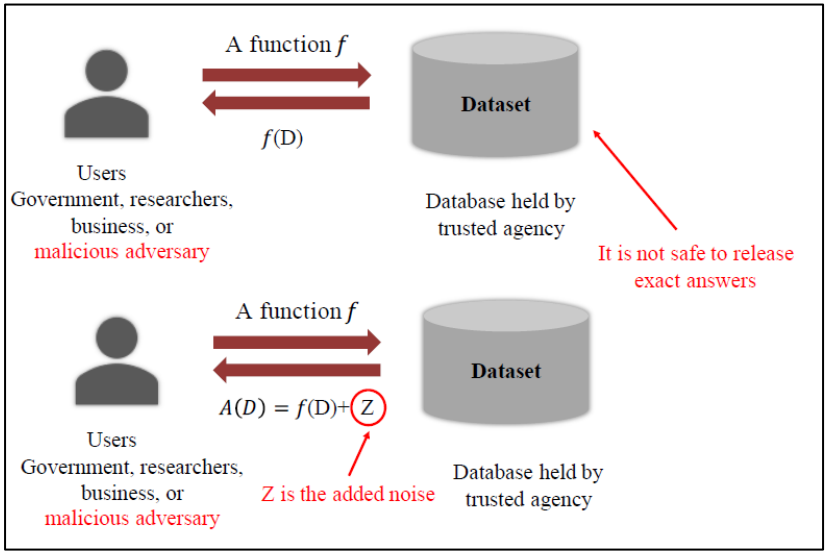




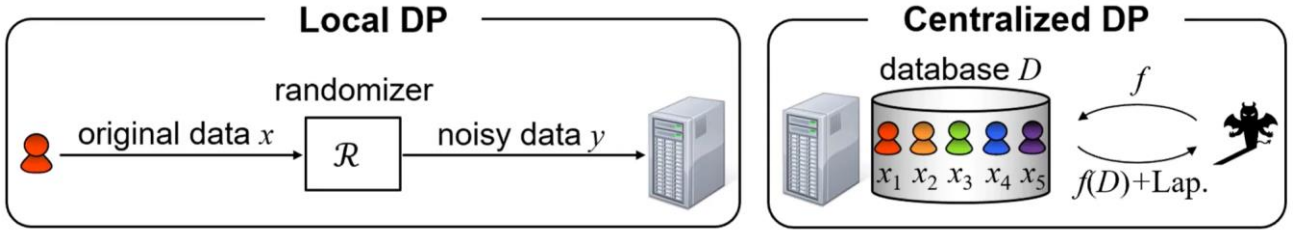
Differential Privacy

- Differential privacy addresses adversarial attacks that queries datasets differing by only a small number of entries

- **DP:** *Adding noise to query results*
  1. How to add noise
  2. How much noise to be added



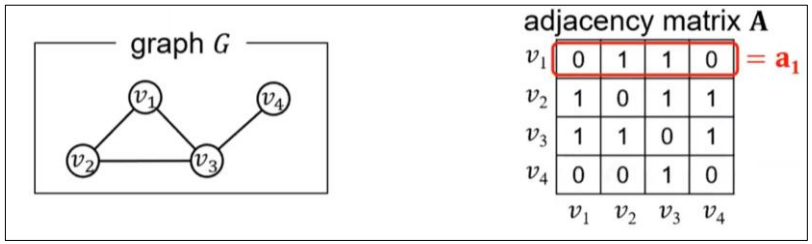
- Obfuscating user data
  - individuals/third party

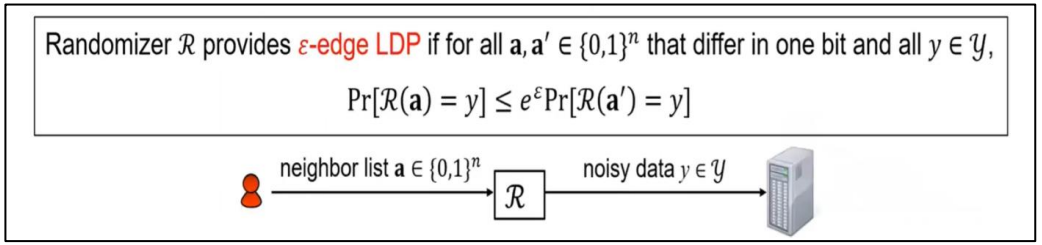


Differential Privacy on Graphs

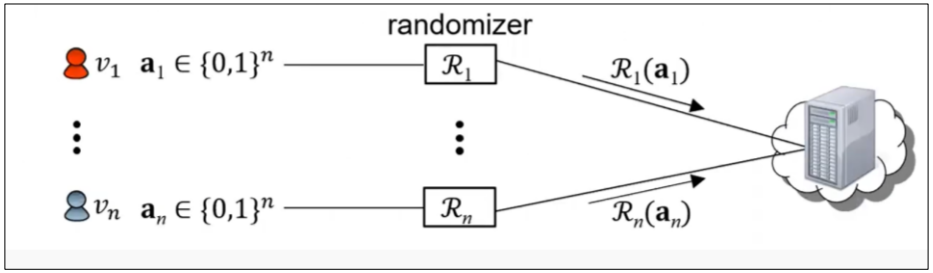
- **Edge DP:** Algorithm should not reveal inclusion/removal of edge in a graph

  - Graph: Represented as an adjacency matrix **A** (1: edge, 0: no edge)
  - Use v_i knows her neighbor list a_i (*i-th row* of **A**).



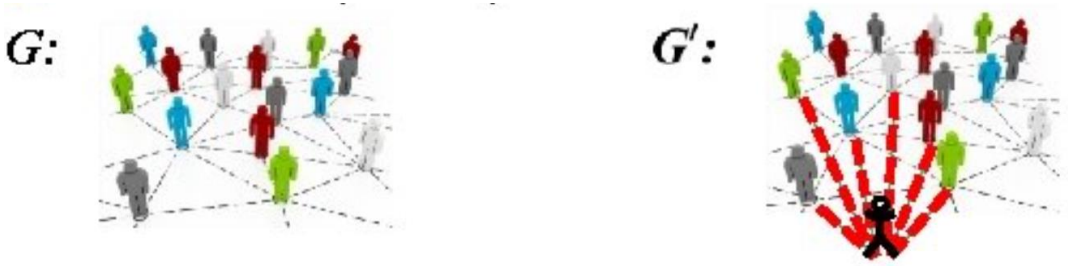


- Protects a single bit in a neighbor list a ε $\{0,1\}^n$ with privacy budget ε.

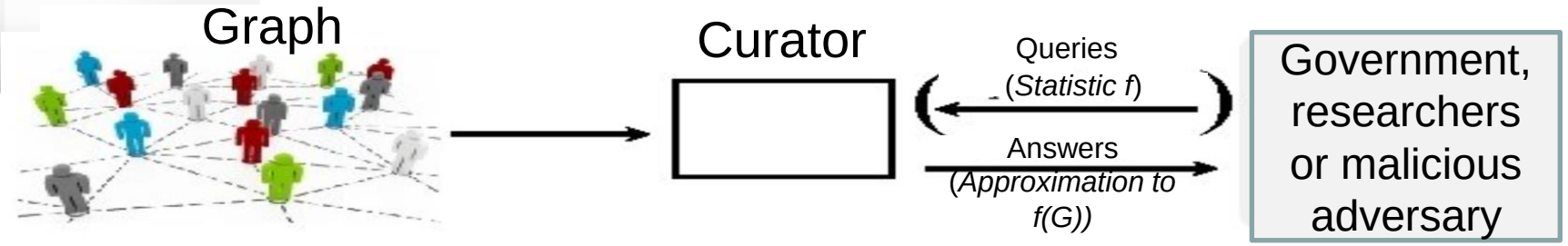


Differential Privacy on Graphs

- **Node DP:** A node DP algorithm will have similar output distributions on any pair of graphs that differ in one node and edges adjacent to it
- Notion of node neighbors is used



Motivation



- **Conflicting goals:** Utility (accurate answers) Vs. Privacy
  - Too much noise → reduce utility
  - Too little noise → cannot suffice privacy guarantee

- **Design node-differentially private algorithm that compute accurate graph characteristics on large family of realistic graphs**

  - **Research Question:** How accurately can an ε-differentially private algorithm release f(G)
    - » i.e. graph characteristics such as subgraph count, number of degrees, degree distribution?

Challenge for Node Privacy: High Sensitivity

- **Sensitivity: Change in query result caused by adding/removal of edge/node in graph**
- Laplace mechanism for adding noise:

Randomized value — Sensitivity

$$A(D) = f(D) + Lap(\frac{\Delta f}{\varepsilon})$$

Real value

Magnitude of noise is proportional to function *f* which measures the maximum changes of adding or deleting changes on output

- Global Sensitivity of a function *f* is:

$\Delta f = \max |f(G) - f(G')|$ (maximum difference in function *f* for two neighboring datasets)

Node neighbors G, G'

- Laplace is difficult: functions on graphs are

→ highly sensitive to insertion/removal of well-connected node

  - **Example: Statistics**
    - » f_(G) is the number of edges in G
    - » f Δ(G) is the number of triangles in G

G': Addition of a new node in G with n nodes and edge with each n

δf_ (Sensitivity)= n
δfΔ (Sensitivity) = nC2

Too High!

"Projections" on Graphs of Small Degree

- Let G= family of all graphs,
  G_d= family of graphs of degree <= *d*

Notation: δf = Global sensitivity of *f* over G

$\delta_d f$ = Global sensitivity of *f* over G_d

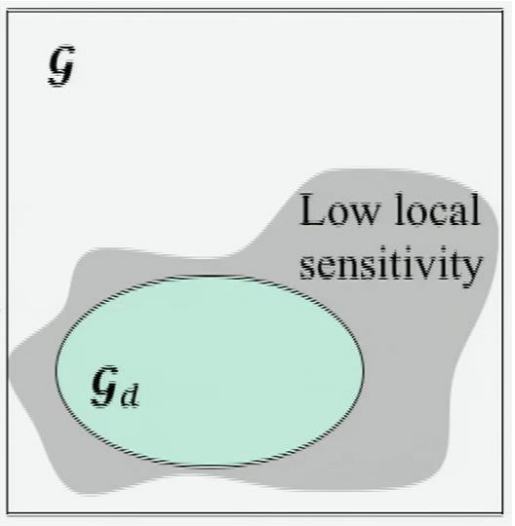
Observation: $\delta_d f$ is low for many useful *f*

(function f has low/ bounded sensitivity, Lipschitz constant, on G_d)

Examples:

$\delta_d f\_ =$ d (compared to δf_ = n and d <<n)

$\delta_d f\ \Delta$ = dC2 (compared to δfΔ = nC2)



Goal: Privacy for all graphs!

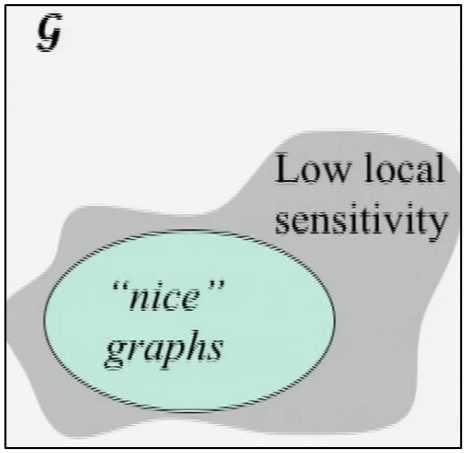
Idea: ***Knowing the input lay for sub-class of graphs G_d, we can design more accurate differentially private algorithm by adding noise proportional to restrictive notion of sensitivity***

General Technique for node-DP

1. Identify a set of “nice” graphs
   - Example: Graphs of maximum degree at most *d*
   - *Should include graphs you care about*

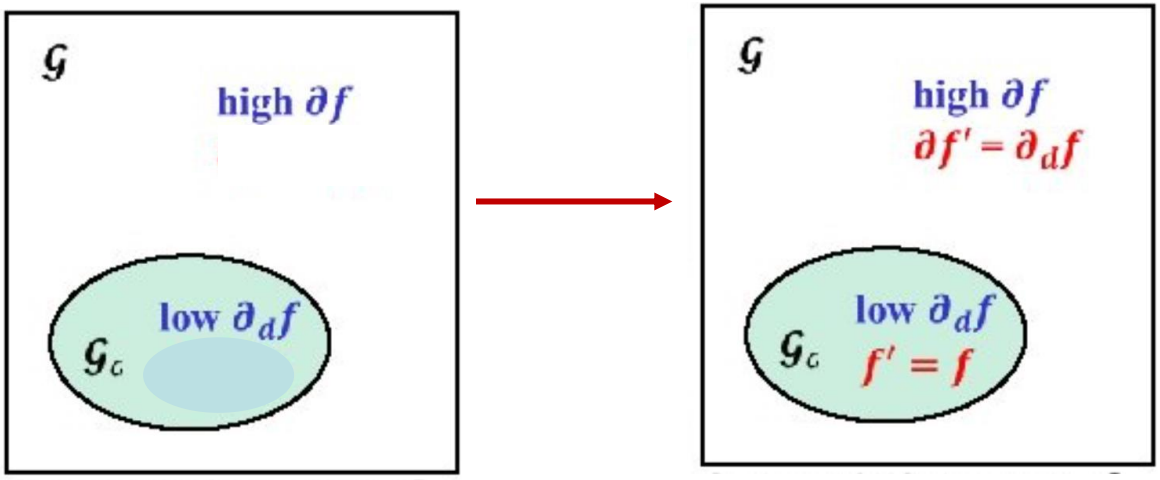
2. Design an algorithm that is differentially private on “nice” inputs

3. “Extend” algorithm on all possible inputs



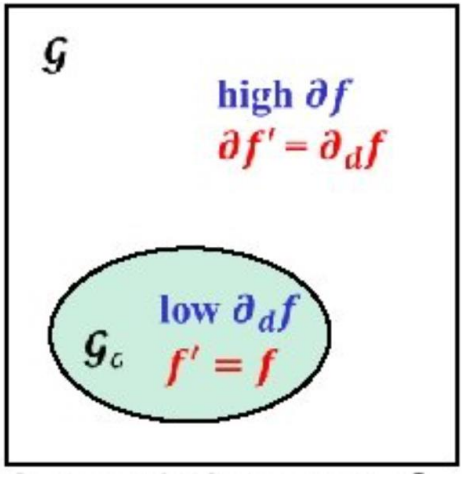
Method-1: Lipschitz Extension

- **Idea:** Given a function *f* with low Lipschitz constant (bounded sensitivity) on "nice" graphs, if we can compute Lipschitz extension *f'* defined on all G
  - Then, use Laplace mechanism to release *f'(G)* with relatively small additive noise
  - Lower the stretch of extension, the lower the overall noise
  - Accurate result if input falls near or "close" to class of "nice" graphs



Method-1: Lipschitz Extension

A function f' is a Lipschitz extension of f from G_d to G if
- f' agrees with f on G_d (same answers), and
- $\delta f' = \delta_d f'$

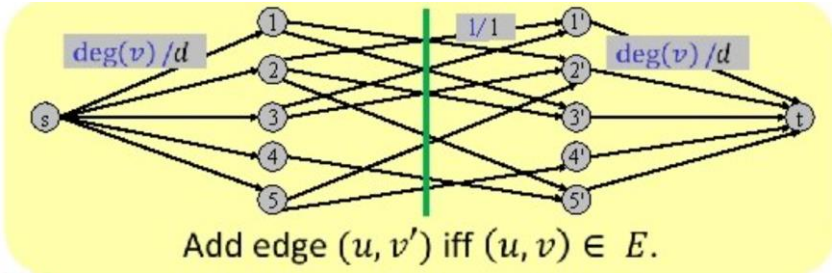


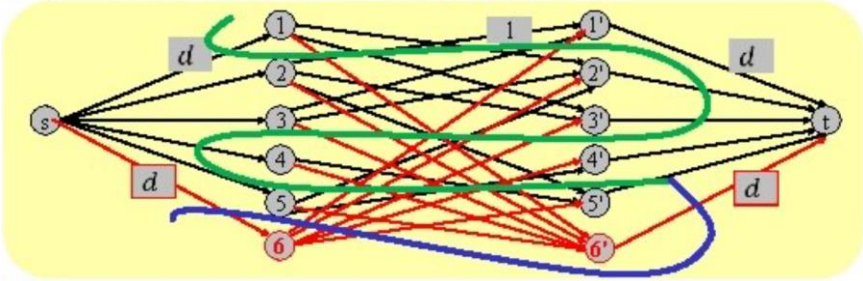
- Requires designing Lipschitz extension for each *f*
  - Done using maximum flow and linear and complex programs

- There exists Lipschitz extension for most real valued functions
- Lipschitz extensions can be computed efficiently for functions:
  - » Subgraph counts
  - » Degree distribution

Lipschitz Extension of *f_: Flow Graph*

- For a graph G= (V,E), define flow graph of G:





- $V_{flow}(G)$ is the value of maximum flow in the graph

Lemma: $V_{flow}(G)/2$ is the Lipschitz extension for *f_*

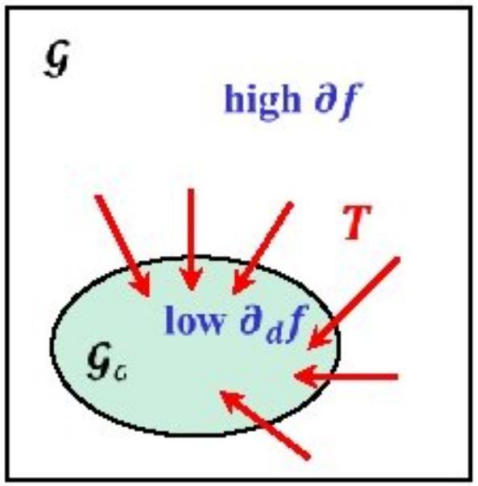
Proof: (1) $V_{flow}(G) = 2f\_(G)$ for all $G \in G_d$ (flow function should have low global sensitivity on degree bounded graphs)

(2) δ Vflow $= 2.\ \delta_d f\_ = 2d$

Method-2: Reduction to Privacy over G_d

- **Input:** Algorithm B that is node-DP over G_d
- **Output:** Algorithm A that is node-DP over G has accuracy similar to B on nice graphs

- Project on G_d by doing
  - Truncation T(G) for separation
  - Truncation outputs G with nodes of degree > d removed
  - Answer queries on truncated graph not G



Contributions

Techniques used to obtain results:

- Node differentially private algorithms for releasing-

1. Number of edges
2. Counts of small subgraphs

(e.g. triangles, k-triangles, k-stars)

1. Degree Distribution

via Lipschitz extensions

via generic reduction

Key Takeaways

- Differential privacy requires that a change to one individual's input data not does not affect the algorithm's output distribution too much

- Accurate subgraph counts or number of edges for realistic graphs can be computed by node private algorithms
  - Use Lipschitz extensions

Pros and Cons

Pros	Cons
Focuses on computing graph statistics using node privacy	Cannot be used for releasing subset of nodes in input graph with differential privacy (e.g. vertex cover)
Comprehensive theoretical analysis and proofs	Limited empirical evaluation without validating on social network graph data
The accuracy guarantees are high for their proposed algorithms (truncation/ Lipschitz mechanism)	Assumes centralized model with a trusted curator that holds entire graph and sanitized versions of statistics (**not practically feasible**)
Stronger privacy guarantee	-

Discussion Questions

Hint

- What can't be computed differentially privately?
- What are the two metrics of differential privacy in graphs?
- Why is node differential privacy a stronger guarantee?
- Why does node privacy exhibit high sensitivity in contrast to edge privacy?

Sensitivity → largest change to the query results caused by adding/deleting any record in the dataset, is the key parameter to determine the magnitude of the added noise.

Paper-2

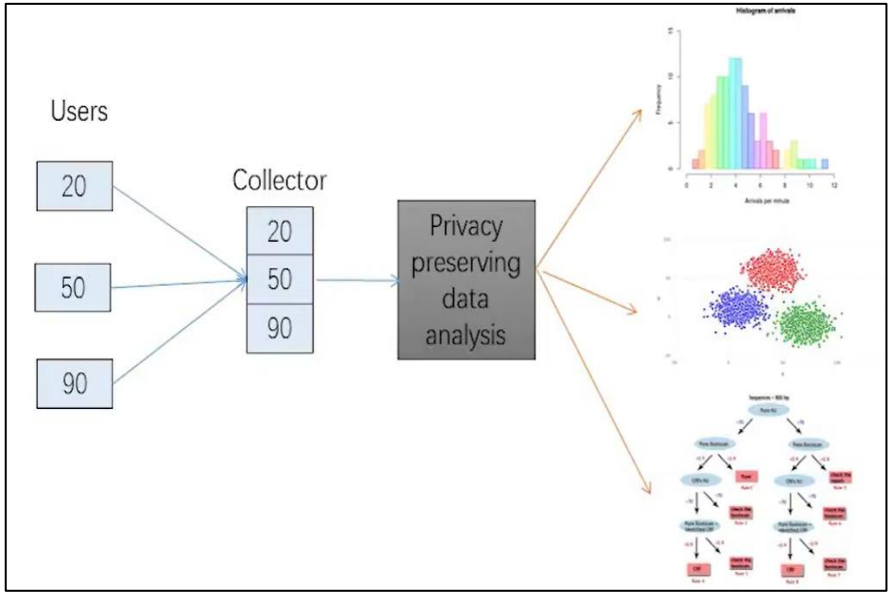
“Generating Synthetic Decentralized Social Graphs with Local Differential Privacy”

Venue: CCS’17

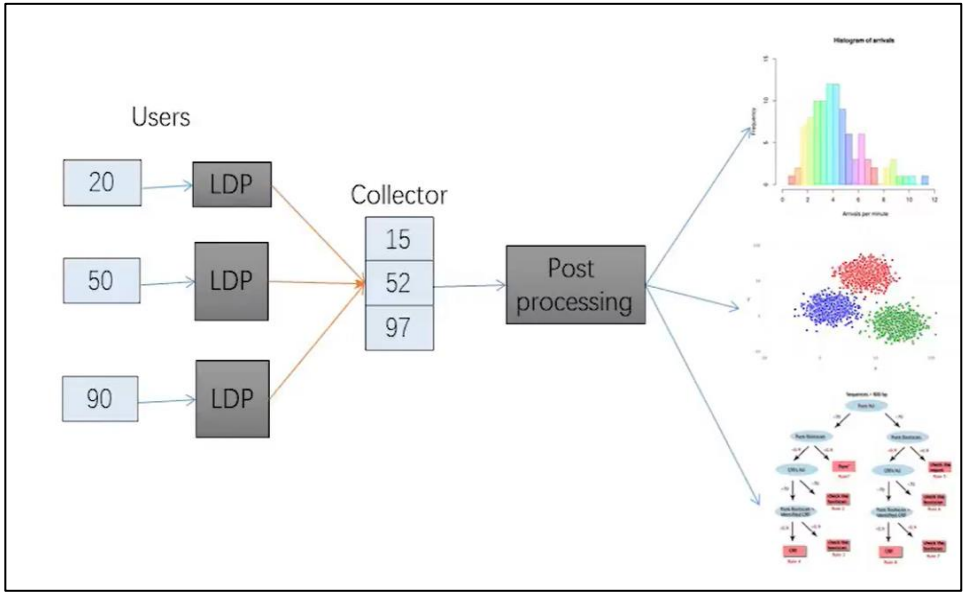
Authors: Zhan Qin1,2 , Ting Yu2 , Yin Yang3 , Issa Khalil2 , Xiaokui Xiao4 , Kui Ren

Motivation

Global Privacy Model



Local Privacy Model



Collecting Statistical information: counts, histograms

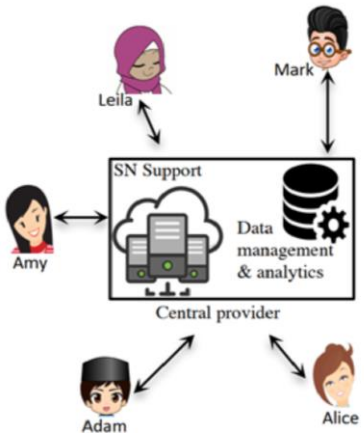
Local Privacy: Graph information stored by users in their limited local view
Global Privacy: Central collectors knows whole input graph

Decentralized Social Networks

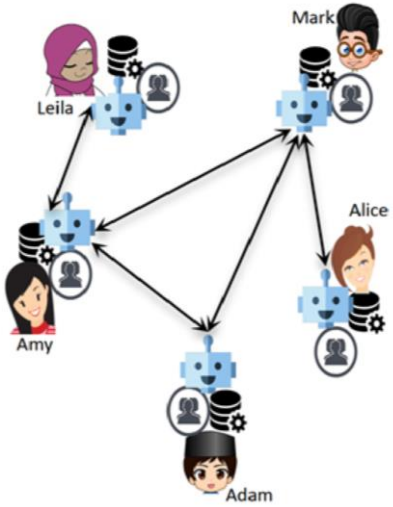
- No single entity holds the whole graph
  - Graph structures distributed among individuals
  - Global privacy model not applicable

- **Example:**
  - Phone contact list
  - Friends with face-to-face interactions

- Analyzing decentralized social network (structure, properties) requires collecting data from individuals
  - Privacy is a must



Centralized



De- Centralized

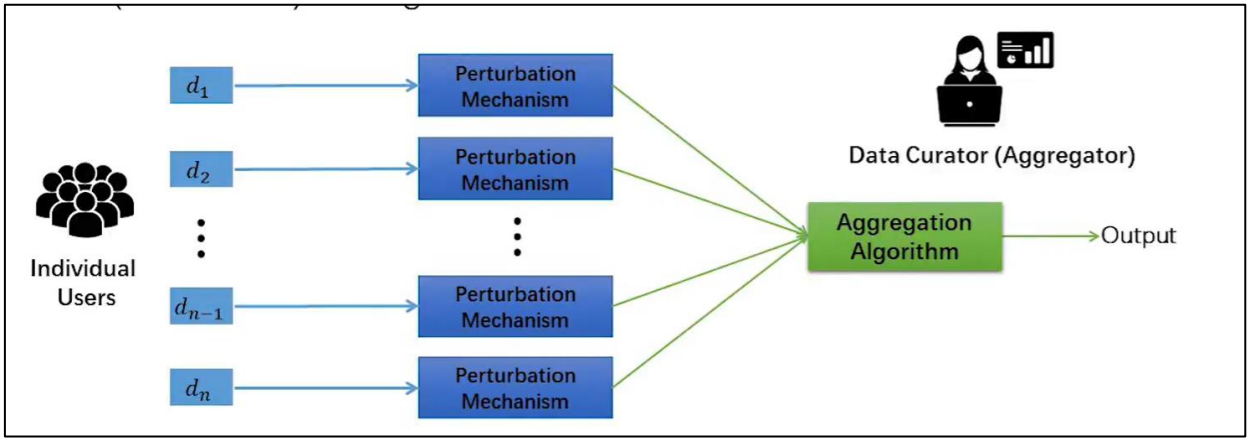
Research Problem

- Could we generate a synthetic graph of a decentralized social network under local privacy definition and local privacy model?

Local Differential Privacy (LDP)

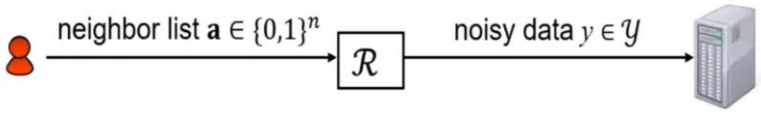
- Privacy guarantee for individual's sensitive data in a local (distributed) setting
- *Mathematically*, mechanism M is ε-LDP if for any two inputs v and v':

$$\frac{\Pr[M(v) \in s]}{\Pr[M(v') \in s]} \leq e^{\varepsilon}$$



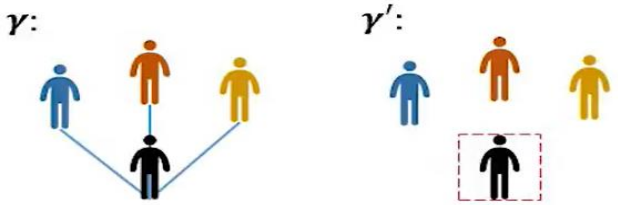
Randomizer $\mathcal{R}$ provides ε-edge LDP if for all $\mathbf{a}, \mathbf{a}' \in \{0,1\}^n$ that differ in one bit and all $y \in \mathcal{Y}$,

$$\Pr[\mathcal{R}(\mathbf{a}) = y] \leq e^{\varepsilon} \Pr[\mathcal{R}(\mathbf{a}') = y]$$

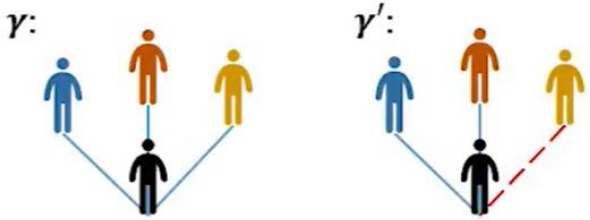


Local Differential Privacy for Graph Data

- Node local differential privacy
  - Hide the inclusion/removal of node
  - Neighbor lists *y* and *y' could* differ in all edges



- Edge local differential privacy
  - Hide the inclusion/removal of an edge
  - Neighbor list y and y' differ in one edge
  - Less perturbation to graph



Local Differential Privacy for Graph Data

- Node local differential privacy is stronger
  - An overkill in some cases, too much noise
  - Heavy price in utility of computed data even in global setting

- Edge local differential privacy is sufficient in many cases
  - Users want to protect who exactly are their contacts rather than whether they have any contacts

Existing Work

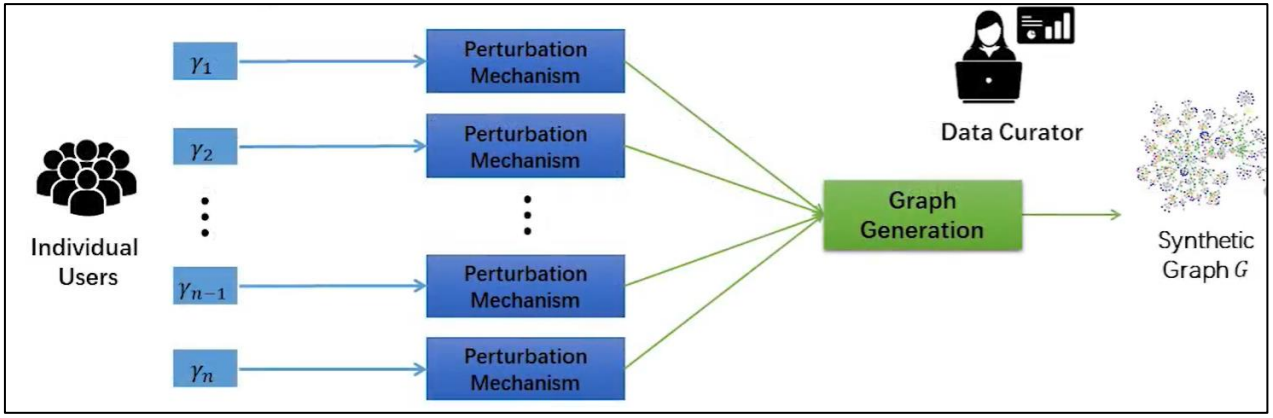
- Binary attribute, unary attribute, single attribute
- Limited to categorical/numerical data statistics
  - Which portion of users give a certain answer?
  - Popular website among population

- Challenge: Here, we need graphs with statistics

Problem Statement

Input: each user has a neighbor list *y*

Output: synthetic social Graph G

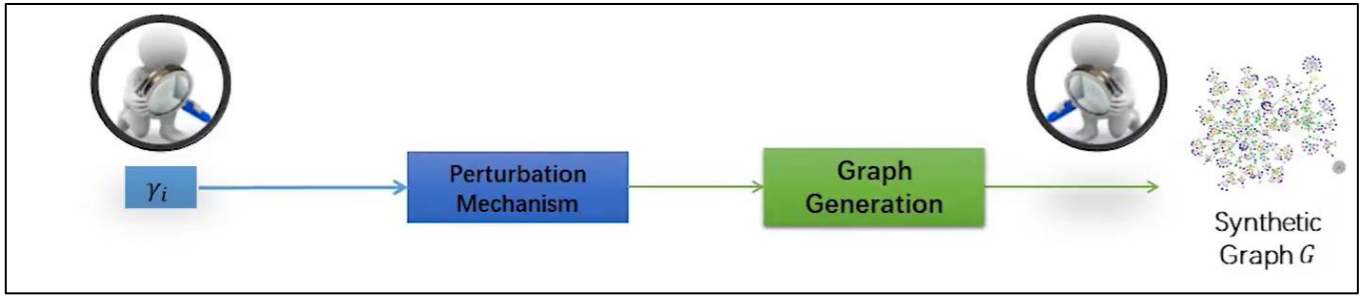


The synthetic graph should capture underlying regional properties of decentralized social graph

Straw-man Approaches

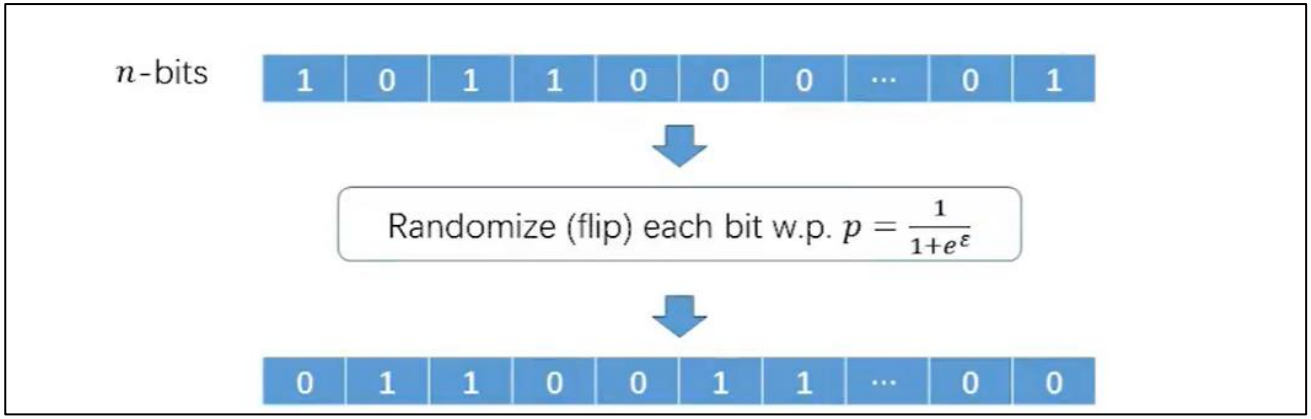
- Approach-1: Focus on data collection
  - Collect everything that user has (neighbor list) with privacy guarantee

- Approach-2: Focus on graph generation
  - Collect parameters graph generating algorithm needs with privacy guarantee
  - Parameters are derived from social network



Approach-1 (Randomized Neighbor List; RNL)

- Collecting everything that user has with privacy guarantee
  - Randomized Neighbor List Approach (RNL)

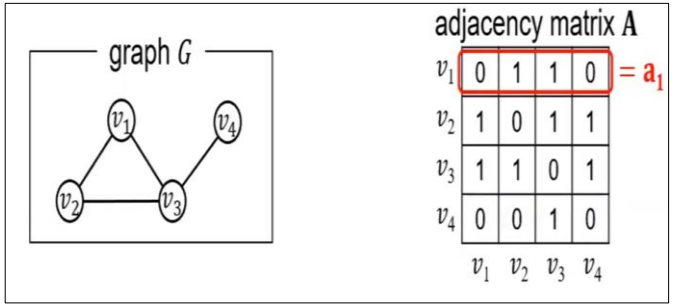


- Shortcoming: Much denser graph, original real graph is sparse (few 1s)
  - 200 times edge density increase when p=0.01, ε=4.6
  - Loosing information of original graph

Approach-2 (Degree based Graph Generation; DGG)

- Collecting everything graph generating algorithm needs (like *local information)* with privacy guarantee

- Existing synthetic social graph generation algorithms
  - Erdos-Reny, BTER, Kroneker
  - Some need local information e.g. node degrees
  - Kroneker needs global information e.g. submatrix of adjacency matrix

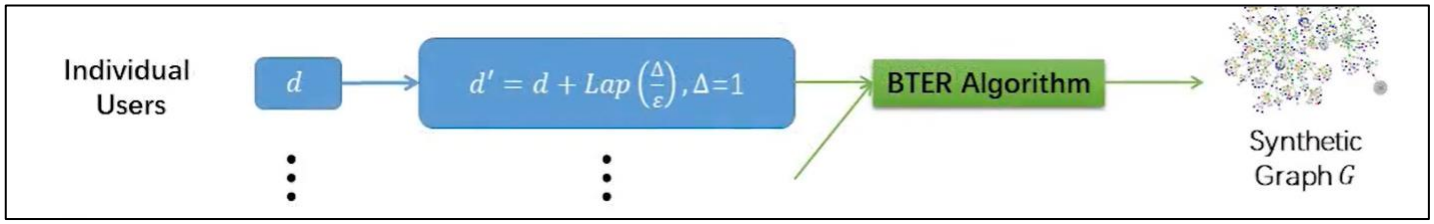


- Perturbed node degree by ε-edge DP is sent to curator
  - Computation of node degree gives accurate degree dist. in synthetic graphs

Limitation: Each user only has a limited local graph view

Straw-man Approach 2

- Degree based Graph Generation Approach (DGG)
  - Adapted BTER
  - Perturbing node degree under ε-edge LDP
  - Cluster nodes and generates edges based on node degrees



Downside: Capture node degrees and clustering coefficient but lose all other structure information

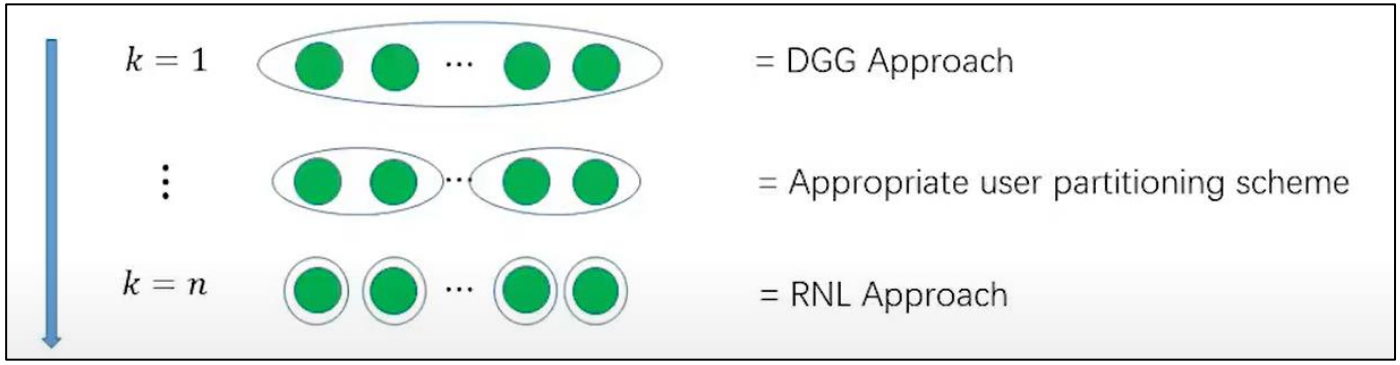
- Two nodes with same degree → different cluster
- Cannot capture node connections of graph, only degree

Key Observations

- *Balance between noise introduction for DP and information lost* for collecting information at coarser granularity
- RNL collects fine-grained information (neighbor list), but has the price of heavy perturbations to satisfy local DP
- DGG introduces small amount of noise, but only collects coarse-grained statistics (node degree) to satisfy local DP

Methodology

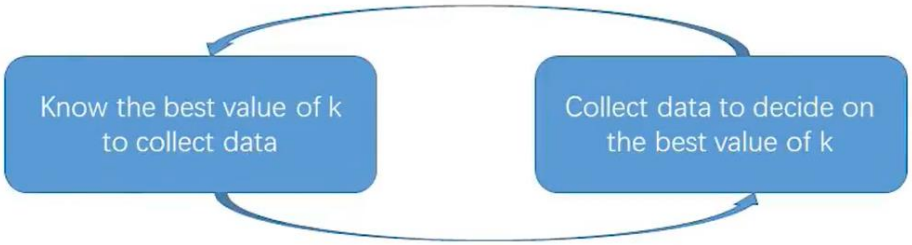
- Partition all users into k groups , each user reports k degrees for each group respectively



- Choosing an appropriate user partitioning scheme is the key

Methodology

- Ideally, clustering of nodes → depends on data
- **Circular Dependency:** To collect data, need to know best value of k and partitioning schedule AND to know k, need to collect data

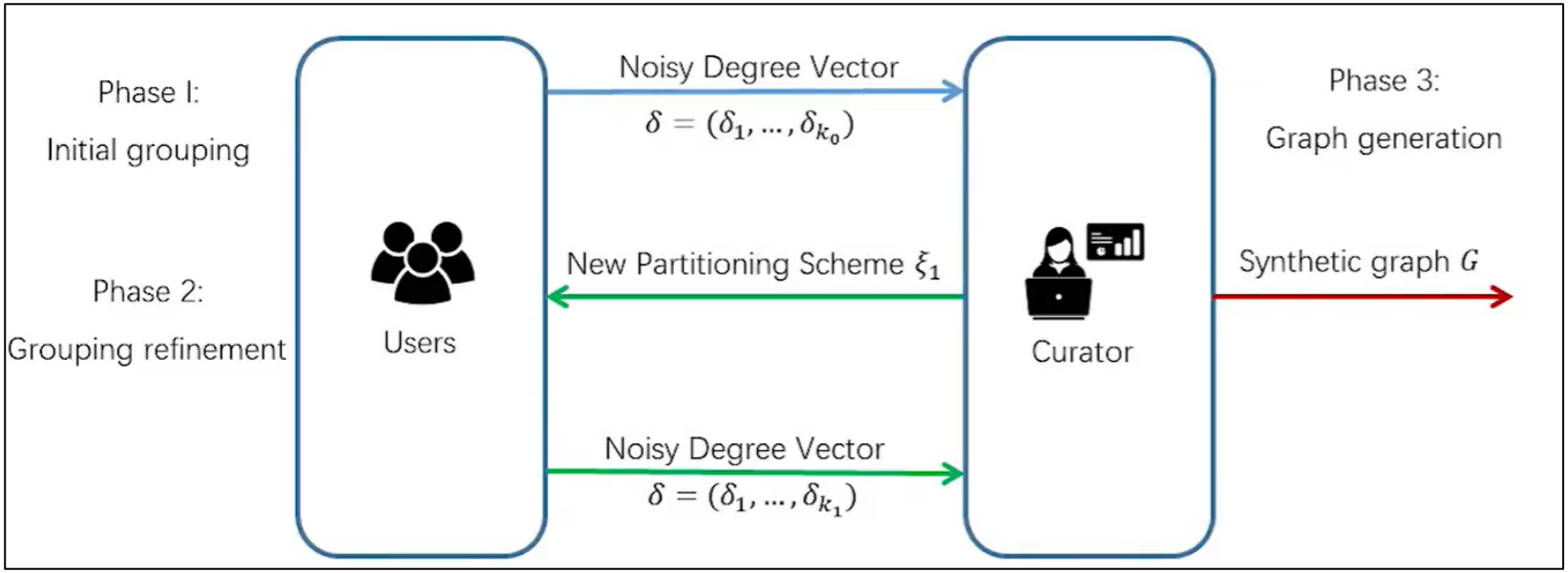


- Initial stage: ask nodes to report degree to random partitions → *iterative!*

Multi-phase design addresses:

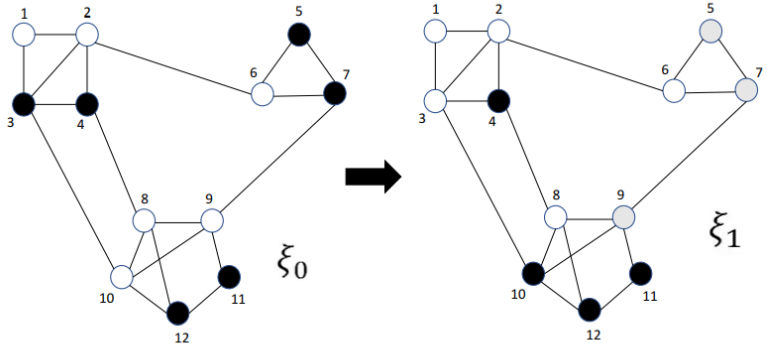
1. Collects data from users in multiple rounds
2. Each round refines user partitioning scheme, collects data again with higher accuracy used in next round
3. Structurally close users will gradually group together

LDPGen: General Framework



LDPGen: Design Phase I

- Initial Grouping
- Users report degree vectors using initial partition groups ξ_0 provided by curator
- Curator computes a new grouping scheme ξ_1 based on collected degree vectors to each partition
- Approx. optimal number of partitions



Phase I: initial random partition (k_0=2)

Phase I: optimize group number (k_1=3) and refine partition

$$k_1 \approx \left\lceil \sum_{\eta=1}^{\eta_{max}} p_\eta \left(\frac{1}{2}\eta + \frac{\frac{1}{4}\eta^2 - (1+\sqrt{5})\eta + 1}{\varepsilon_2}\right) \right\rceil$$

Where p_η is the percentage of nodes with degree η

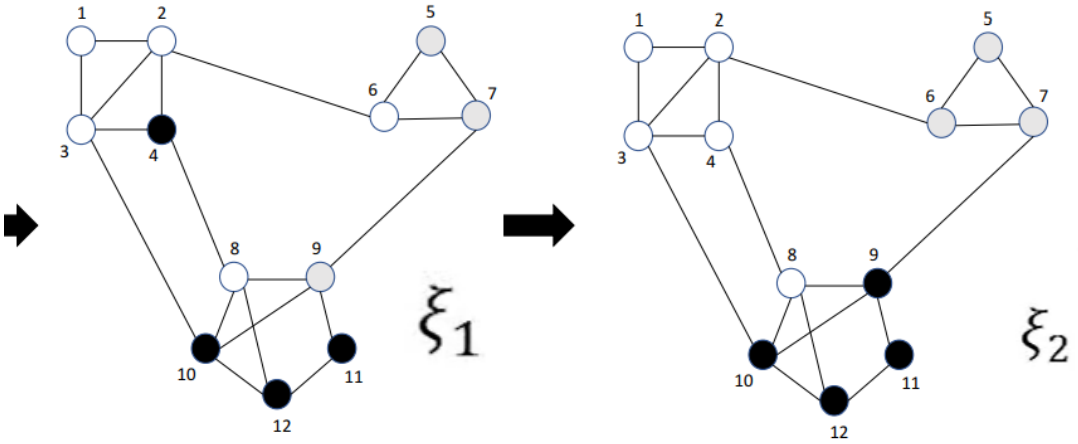
Performing k-means algorithm to partition users into k1 groups

LDPGen: Design Phase II

- Refining grouping results
- Users report again new degree vectors using refined partition groups provided by curator
- Curator computes a new grouping scheme based on collected degree vectors

$$k_1 \approx \left\lceil \sum_{\eta=1}^{\eta_{max}} p_\eta \left(\frac{1}{2}\eta + \frac{\frac{1}{4}\eta^2 - (1+\sqrt{5})\eta + 1}{\varepsilon_2} \right) \right\rceil$$

Where p_η is the percentage of nodes with degree η



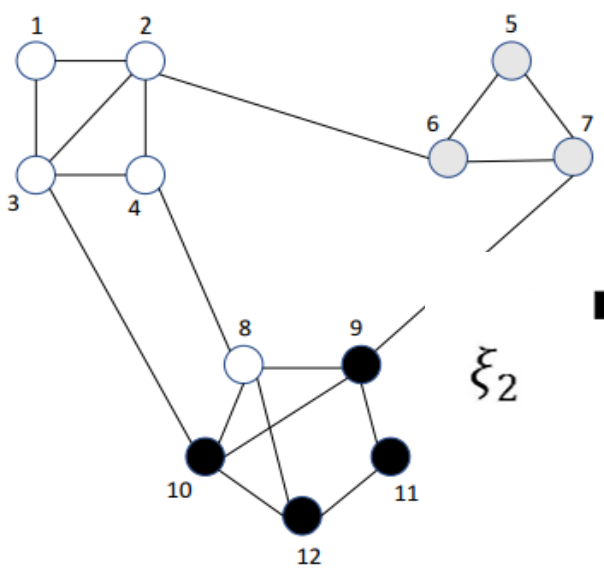
Phase I: optimize group number (k_1=3) and refine partition

Phase II: further refine partition with optimized group number

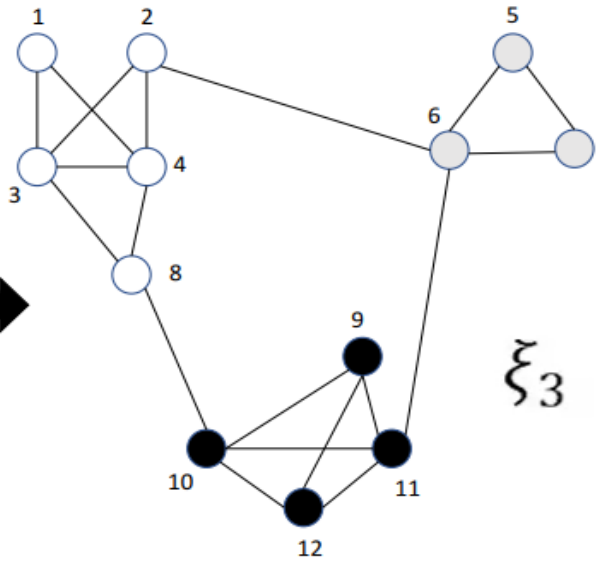
LDPGen: Design Phase III

- Graph Generation using refined grouping result
- Curator adapt graph generation algorithm (*Inter-cluster and intra-cluster edges are generated*) on the user clusters
  - Probability of generating an edge between two nodes is proportional to their node clusters' aggregated degrees to each other

$$p = \frac{\delta_j^u \cdot \sum_{v \in G} \frac{\delta_j^v}{|U_j^2|}}{\sum_{u \in U_i^2} \delta_j^u + \sum_{v \in G} \delta_j^v}$$



Phase II: further refine partition with optimized group number



Phase III: generate synthetic graph based on refined user partition

Experimental Analysis

- Benchmark Datasets:

Facebook, Enron, Last.fm, Flixster

Utility Measurements:

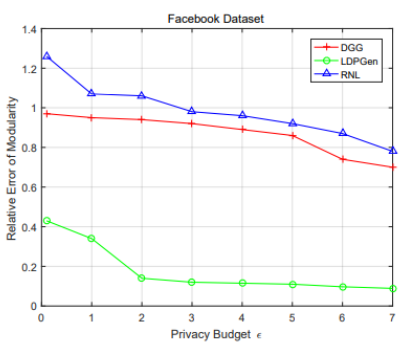
1. **Global statistics:** graph structure statistics
   - Modularity and Clustering Coefficient
2. **Structural information:** Community structures preservation
   - Similarity of the communities obtained from synthetic graph and original graph
   - Adjusted Mutual Information

3. **Application:** Social Recommendation use case
   - » A list of top-k items based on the two graphs using a same preference dataset
   - » Normalized discounted cumulative gain

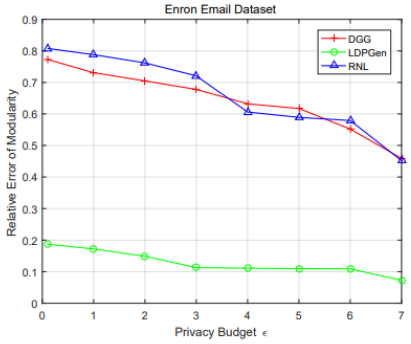
Graph Structure Statistics

Relative error (lower →better) of modularity and clustering coefficient

- DGG is based on BTER which is optimized for capturing clustering coefficient

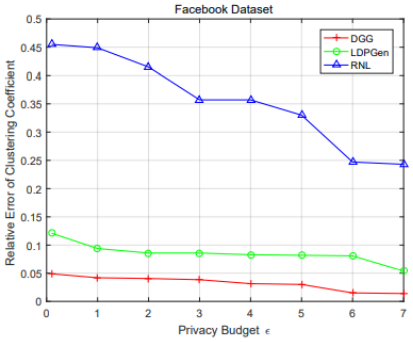


(a) Facebook

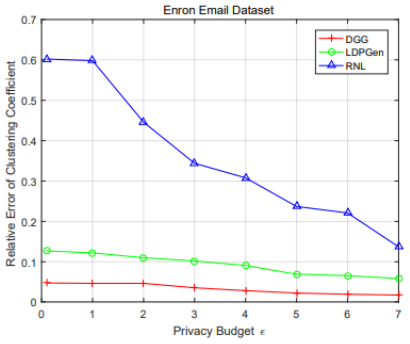


(b) Enron

Effect of ε on Modularity



(a) Facebook



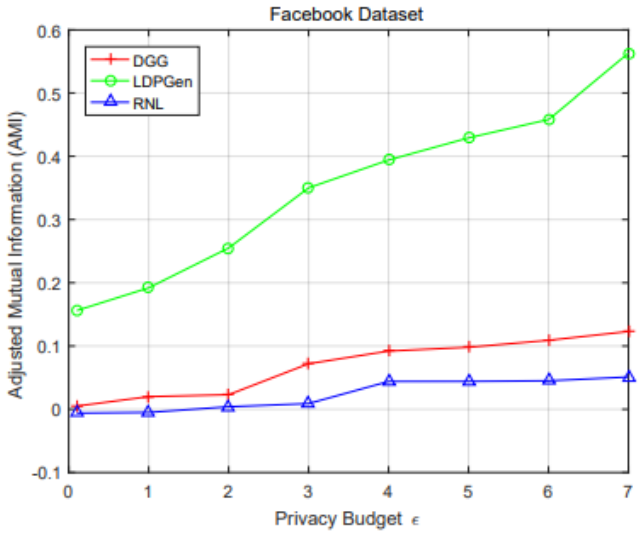
(b) Enron

Effect of ε on Clustering Coefficient

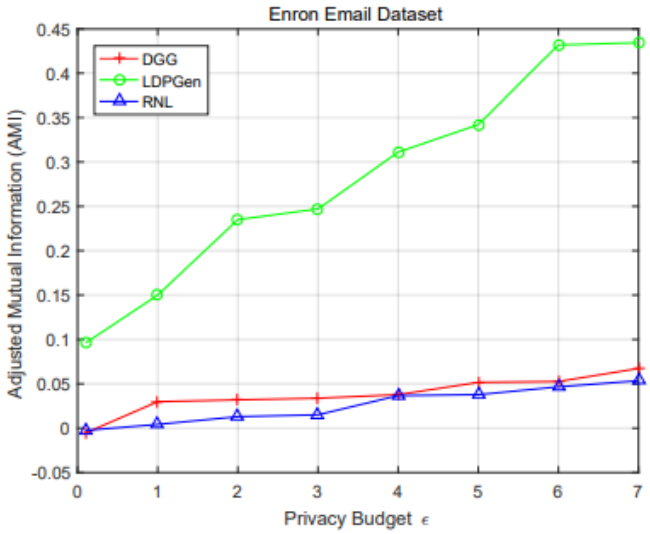
Why LDPGen underperforms?
DGG → degree distribution optimized for clustering coefficient

Community Preservation

- Adjusted Mutual Information (Number of similar communities in synthetic vs. original graph)



(a) Facebook

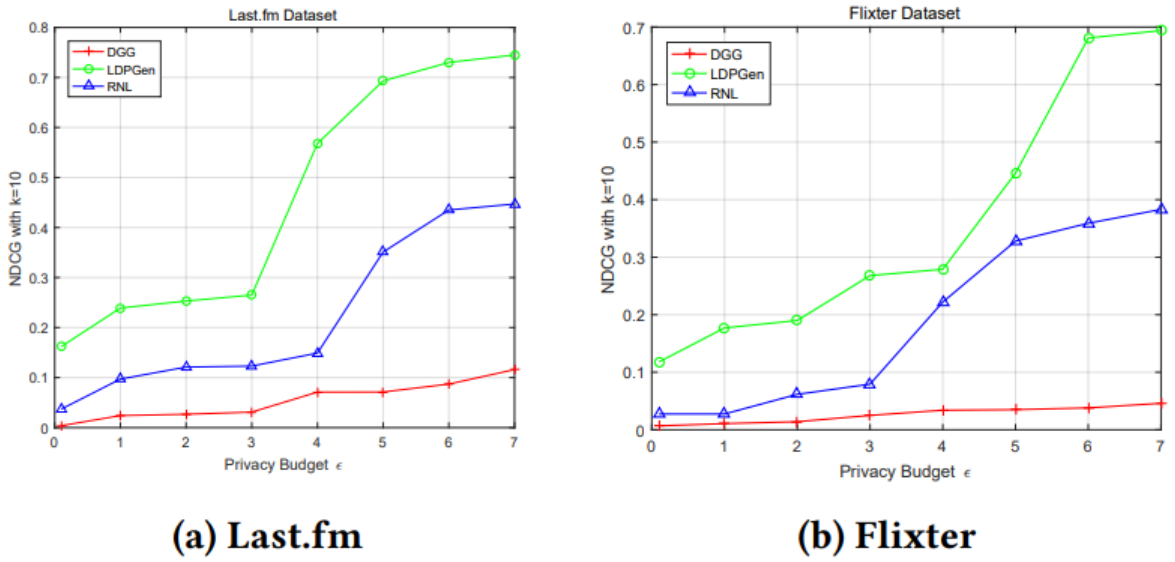


(b) Enron

Effect of ε on AMI

Effectiveness of Recommendation System

- Normalized Discounted Cumulative Gain (NDCG)



(a) Last.fm (b) Flixter

Effect of ε on NDCG

Pros and Cons

Pros	Cons
Similar structure and distribution of nodes and edges	Neighbor information is lost in synthetic graphs
Comprehensive empirical evaluation	No theoretical analysis on cost of LDP in graph generation
Early effort in privacy preservation graph analysis	Task specific approach; centrality measure, community detection
No trust issues, no central entity to share data with	Cannot capture edge weights, node attributes
Leverages users limited local view of graph information with LDP guarantee	May not work well for sparse communities social graph

Conclusion

- **LDPGen:** A multi-phase technique to incrementally cluster structurally similar users via refining parameters into different partitions.
  - Add Laplacian noise when user reports information → guarantee local differential privacy
  - Achieve good clustering → Synthetic graph generation

Future Scope: Stronger privacy guarantees (Node LDP) and more complicated mining tasks (frequent subgraph mining)

Discussion questions

1. What are the disadvantages of local differential privacy over global differential privacy?

2. What is the communication overhead for LDPGen?
   - O(1), O (n), O(n^2)
   - Overhead for RNL, DGG?
3. What are some ways to reduce the communication cost?

4. Why did the authors only include two rounds of iteration for grouping refinement during multiphase process?