Differentially Private Analysis of Graphs

Adam Smith

Pennsylvania State University

Publishing information about graphs

Many types of data can be represented as graphs where

- nodes correspond to individuals
- edges capture relationships
  - **"Friendships" in online social network**
  - **Financial transactions**
  - **Email communication**
  - **Health networks (of doctors and patients)**
  - **Romantic relationships**

image source http://community.expressor-software.com/blogs/mtarallo/36-extracting-data-facebook-social-graph-expressor-tutorial.html

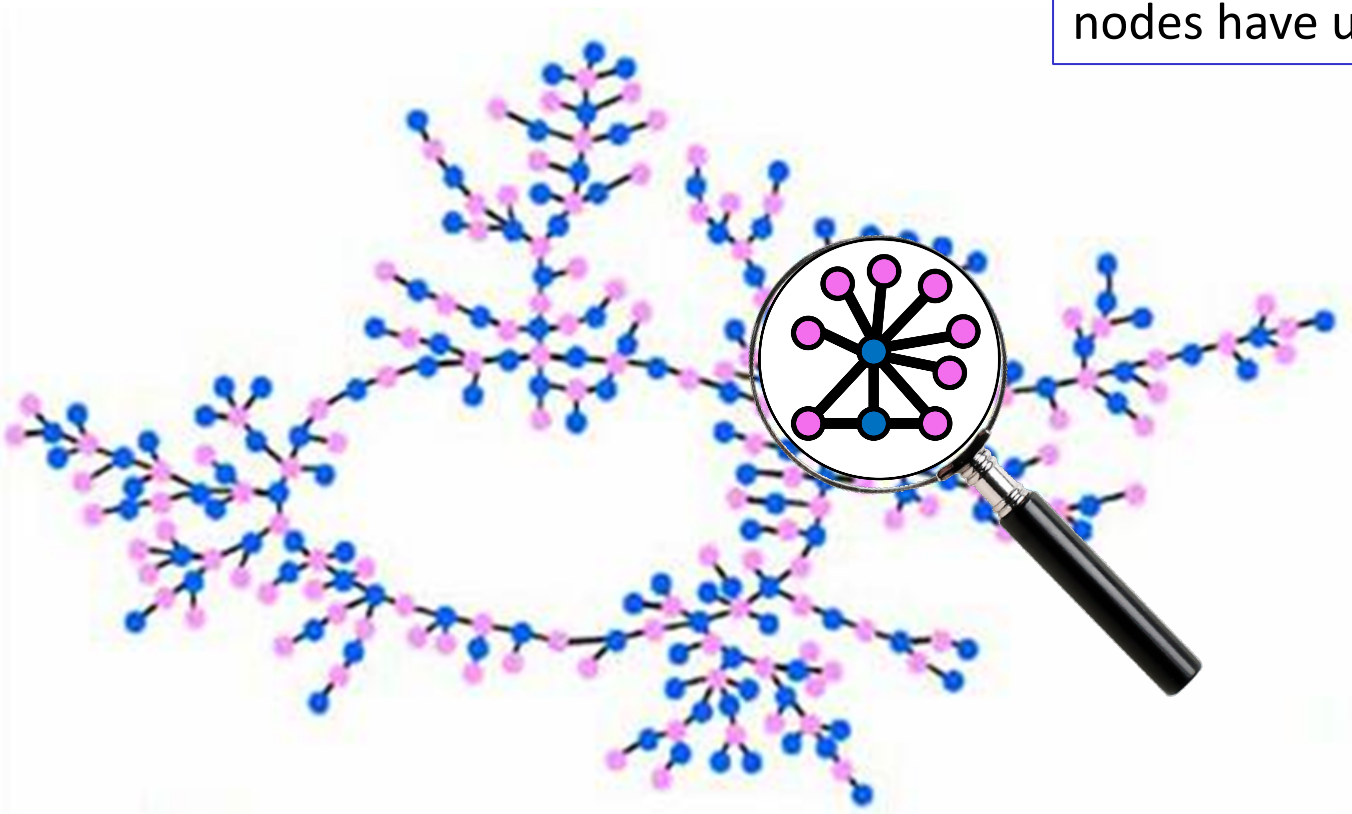
Such graphs contain potentially sensitive information.

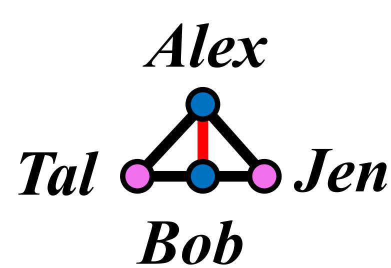
image source http://www.queticointernetmarketing.com/new-amazing-facebook-photo-mapper/

``Anonymized'' graphs still pose privacy risk

- False dichotomy: personally identifying vs. non-personally identifying information.
- Links and any other information about individual can be used for de-anonymization.

In a typical real-life network, many nodes have unique neighborhoods.





Bearman, Moody, Stovel. *Chains of affection: The structure of adolescent romantic and sexual networks, American J. Sociology, 2008*

De-anonymization attacks

– Movie ratings [Narayanan, Shmatikov 08]

– Social networks

[Backstrom Dwork Kleinberg 07,
Narayanan Shmatikov 09, Narayanan Shi Rubinstein 12]

– Computer networks

[Coull Wright Monrose Collins Reiter 07,
Ribeiro Chen Miklau Townsley 08]

Can reidentify individuals based on external sources.

internet

social networks

anonymized datasets

What information can be released without violating privacy?

Two variants of differential privacy for graphs

- **Edge** differential privacy

G:

G':

Two graphs are **neighbors** if they differ in ***one edge***.

- **Node** differential privacy

G:

G':

Two graphs are **neighbors** if one can be obtained from the other by deleting ***a node and its adjacent edges***.

Work on weaker/incomparable privacy definitions

- Edge differential privacy
  - Number of triangles, MST cost [Nissim Raskhodnikova Smith 07]
  - Degree distribution [Hay Rastogi Miklau Suciu 09, Hay Li Miklau Jensen 09, Karwa Slavkovic 12, Kifer Lin 13]
  - Small subgraph counts [Karwa Raskhodnikova Smith Yaroslavtsev 11]
  - Cuts [Hardt Rothblum 10, Gupta Roth Ullman 12, Blocki Blum Datta Sheffet 12]
  - Kronecker graph model parameters [Mir Wright 12]
- Edge-private against Bayesian adversary (weaker privacy)
  - Small subgraph counts [Rastogi Hay Miklau Suciu 09]
- Node zero-knowledge private
  - Privacy only for bounded-degree graphs
  - Average degree, distances to graph families [Gehrke Lui Pass 12]

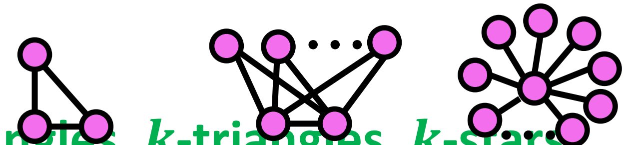
What graph statistics can be computed accurately

with node differential privacy?

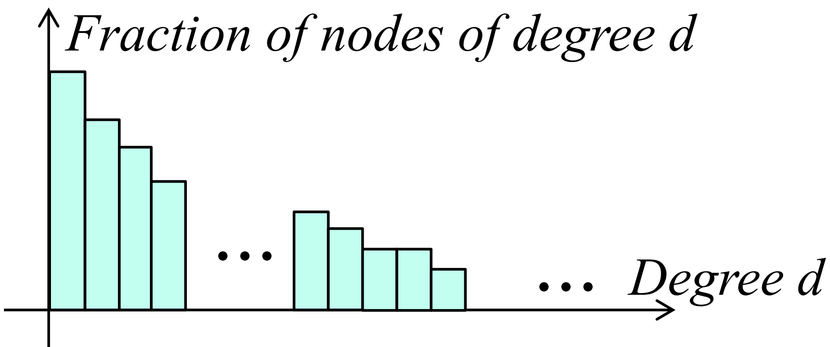
Important: small error on sparse graphs.

Graph statistics

- [Blocki Blum Datta Sheffet 13, Kasiwiswanathan Nissim Raskhodnikova Smith 13, Chen Zhou 13]
  - Number of edges
  - Counts of small subgraphs (e.g., **triangles**, ***k*-triangles**, ***k*-stars**)



- Degree distribution [Kasiwiswanathan Nissim Raskhodnikova Smith 13, Raskhodnikova Smith 16]



Our algorithms for these statistics

- **node differentially private *for all*** graphs
  - **Accurate *for a large subclass*** of graphs (including sparse graphs, scale-free graphs and Erdos-Renyi graphs)
  - **$(1+o_n(1))$-approximation**

Techniques for node-private algorithms

- Previous work
  - **Sensitivity analysis of simple projections** [BBDS'13, KNRS'13]
  - ➢ yielded generic reductions to privacy over bounded-degree graphs
  - **Lipschitz extensions** [BBDS'13, KNRS'13, CZ'13]
  - ➢ yielded much **more accurate** algorithms, but worked only for releasing **1-dimensional statistics** (edge / small subgraph counts).
- This work: new techniques that improve accuracy
  - **Lipschitz extensions** for **high-dimensional statistics**
  - **Generalized exponential mechanism** for both methods

Lipschitz extensions

Basic question: how to compute a statistic f

Graph ***G*** | *Data processing* | *Data release*

Algorithm

Approximation to $f(G)$

How accurately

can an ϵ-differentially private algorithm compute f(G)?

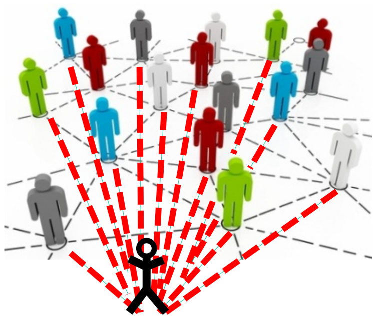
image source http://www.queticointernetmarketing.com/new-amazing-facebook-photo-mapper/

Basic technique: noise proportional to sensitivity

- **Global sensitivity** of a function f is

  $$\partial f = \max_{(\text{node})\text{neighbors } G, G'} |f(G) - f(G')|$$

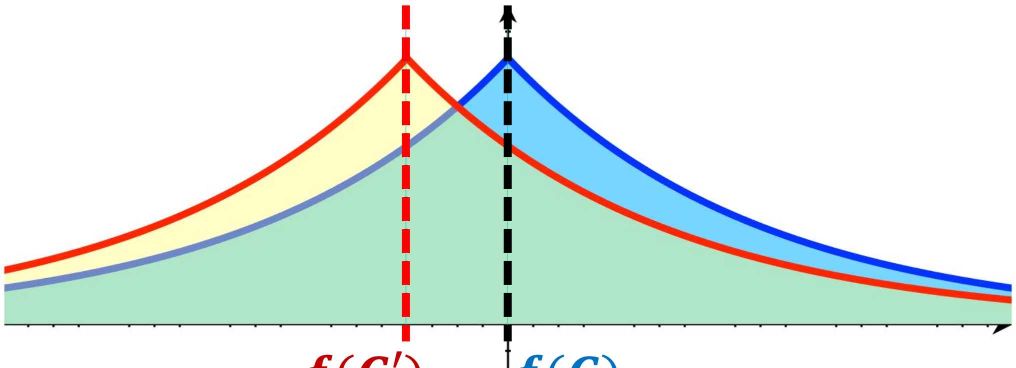
  Also called **Lipschitz constant** of f.



- **GS Framework** [DMNS]: For every real-valued function f, there is an ϵ-differentially private algorithm A such that

  $$\mathbb{E}\left(|A(G) - f(G)|\right) = \frac{\partial f}{\epsilon}.$$

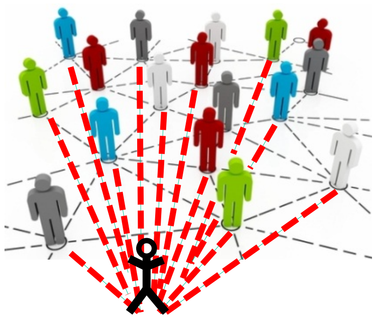
- Intuition: Adding noise $\approx \frac{\partial f}{\epsilon}$ makes G, G' hard to distinguish



Challenge for node privacy: high sensitivity

- **Global sensitivity** of a function f is

$$\partial f = \max_{\text{(node)neighbors } G, G'} |f(G) - f(G')|_1$$



- **Examples:**

for graphs on n nodes:

- f_{edge}(G) is the number of edges in G. $\partial f_{\text{edge}} = n$.
- f_{deg}(G) is the degree list of G. $\partial f_{\text{deg}} = 2n$.

Problem: high-degree nodes.

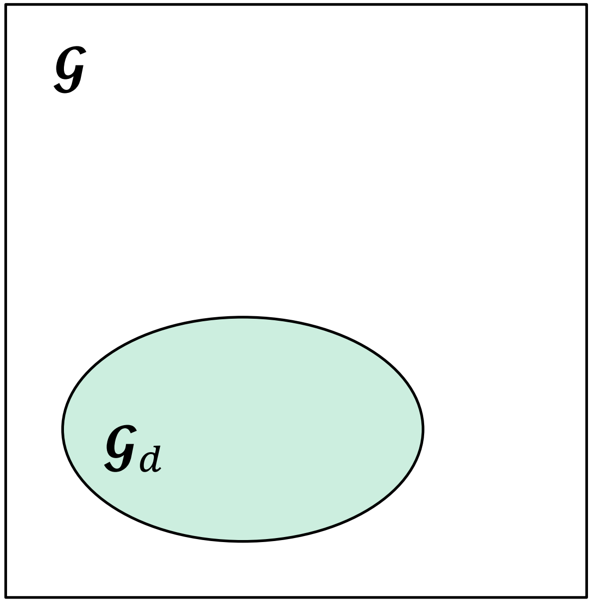
Graphs of small degree

Let $\mathcal{G}$ = family of all graphs,

$\mathcal{G}_d$ = family of graphs of degree $\leq d$.

Notation. $\boldsymbol{\partial f}$ = **global sensitivity of $\boldsymbol{f}$** over $\boldsymbol{\mathcal{G}}$.

$\boldsymbol{\partial_d f}$ = **global sensitivity of $\boldsymbol{f}$** over $\boldsymbol{\mathcal{G}_d}$.



Observation. $\boldsymbol{\partial_d f}$ is low for many useful f.

Examples:

- $\boldsymbol{\partial_d f_{\mathrm{edge}}} = \boldsymbol{d}$ **(compare to** $\boldsymbol{\partial f_{\mathrm{edge}}} = \boldsymbol{n}$**)**
- $\boldsymbol{\partial_d f_{\mathrm{deg}}} = \boldsymbol{2d}$ **(compare to** $\boldsymbol{\partial f_{\mathrm{deg}}} = \boldsymbol{2n}$**)**

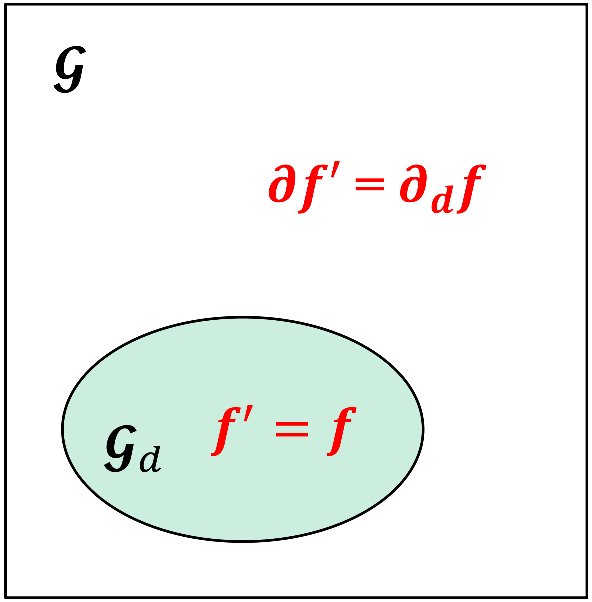
Goal: privacy for all graphs

Idea: ``Extend'' $\boldsymbol{f}$ from $\boldsymbol{\mathcal{G}_d}$ to $\boldsymbol{\mathcal{G}}$ for a carefully chosen $\boldsymbol{d} \in [\boldsymbol{n}]$.

Lipschitz extensions

A function f' is a **Lipschitz extension** of f from $\mathcal{G}_d$ to $\mathcal{G}$ if

- f' agrees with f on $\mathcal{G}_d$ *and*
- $\boldsymbol{\partial f' = \partial_d f}$

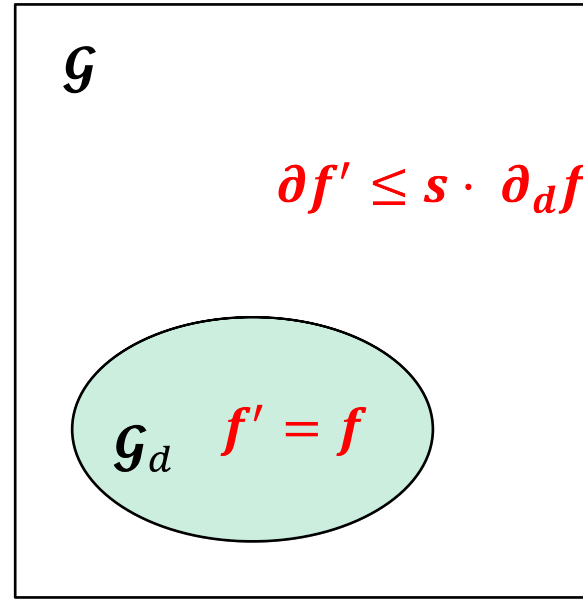


- Release f' via GS framework [DMNS'06]
- All real-valued functions have Lipschitz extensions [McShane 34]
- Lipschitz extensions for **subgraph counts** that can be computed **efficiently** [Kasiviswanathan Nissim **R** Smith 13]

Lipschitz extensions: vector-valued functions

A function f' is a **Lipschitz extension** of f from $\mathcal{G}_d$ to $\mathcal{G}$ **with stretch s** if

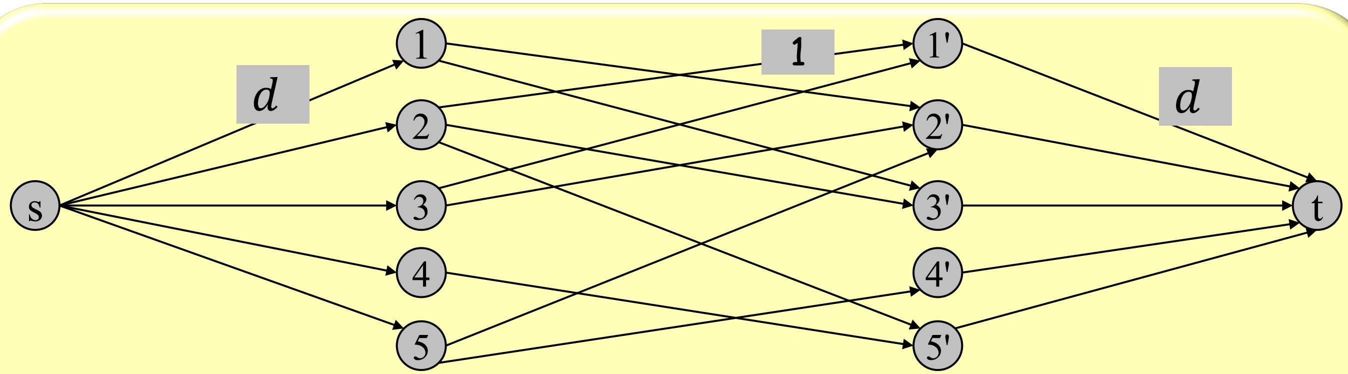
- f' agrees with f on $\mathcal{G}_d$ *and*
- $\boldsymbol{\partial f' \leq s \cdot \partial_d f}$



- We can still release f' via GS framework
- There exist functions $f: \mathcal{G}_d \to \mathbb{R}^3$ ($f: \mathcal{G}_d \to \mathbb{R}^2$ if ℓ_2 is used as output metric instead of ℓ_1) that do not admit *st* pschitz extensions
- Lipschitz extensions of **degree list** and **degree distribution** (with small stretch) that can be computed **efficiently**

Lipschitz extension of f_{edge}: flow graph [KNRS'13]

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



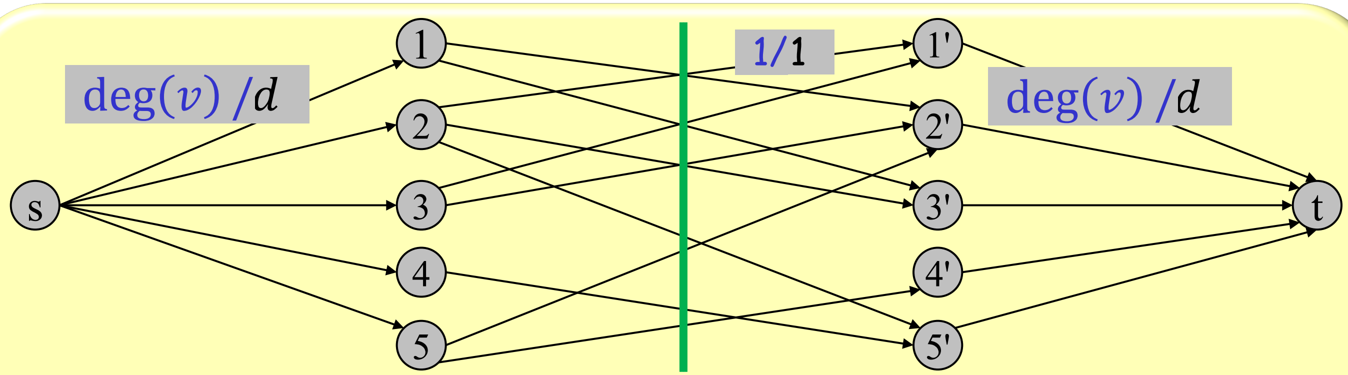
Add edge (u, v') iff $(u, v) \in E$.

v_{flow}**(G)** is the value of the maximum flow in this graph.

Lemma. v_{flow}**(G)/2** is a Lipschitz extension of f_{edge}.

Lipschitz extension of $f_{\mathbf{edge}}$: flow graph [KNRS'13]

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



Add edge (u, v') iff $(u, v) \in E$.

$v_{\mathbf{flow}}$**(G)** is the value of the maximum flow in this graph.

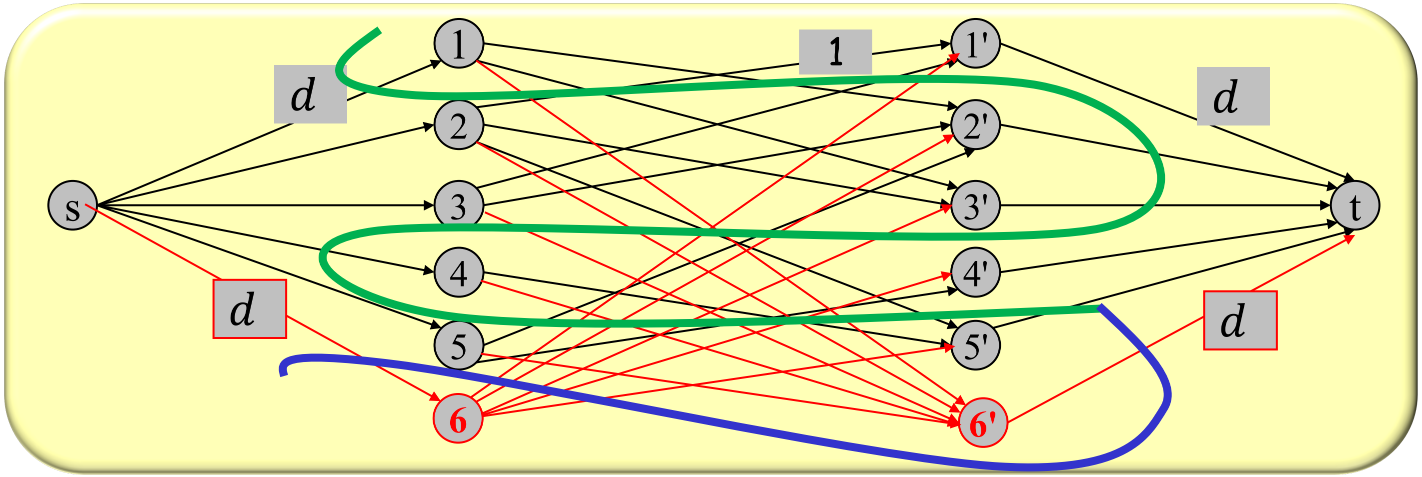
Lemma. $v_{\mathbf{flow}}$**(G)/2** is a Lipschitz extension of $f_{\mathbf{edge}}$.

Proof: (1) $v_{\mathbf{flow}}(\mathrm{G}) = 2 f_{\mathbf{edge}}(\mathrm{G})$ for all $\mathbf{G} \in \mathcal{G}_d$

(2) $\partial\, v_{\mathbf{flow}} = 2 \cdot \partial_d f_{\mathbf{edge}}$

Lipschitz extension of $f_{\mathbf{edge}}$: flow graph [KNRS'13]

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



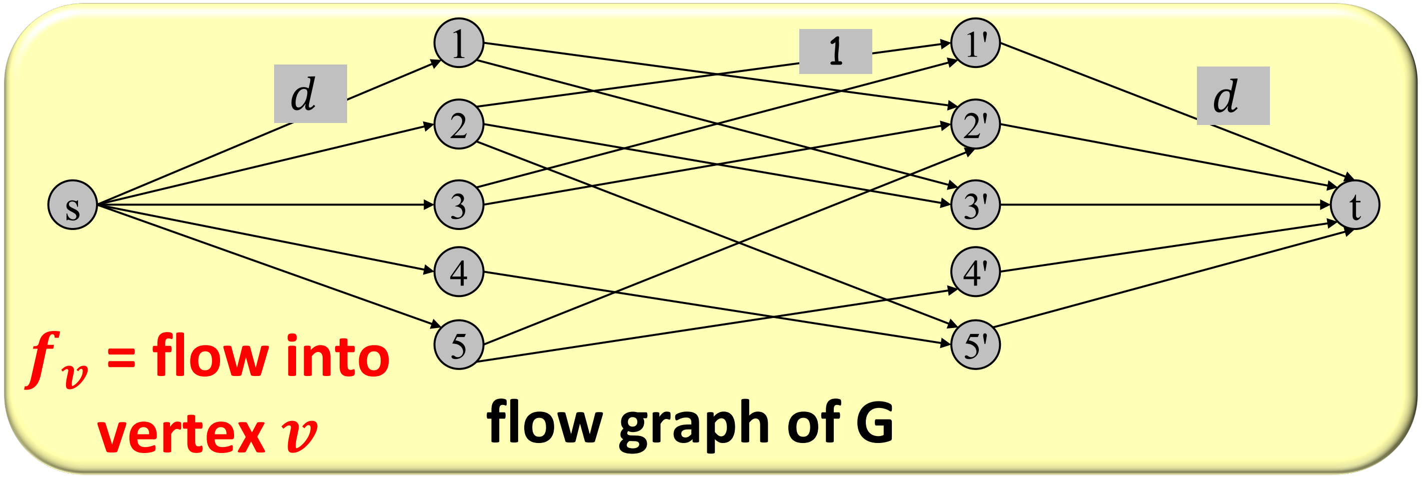
$v_{\mathbf{flow}}$**(G)** is the value of the maximum flow in this graph.

Lemma. $v_{\mathbf{flow}}$**(G)/2** is a Lipschitz extension of $f_{\mathbf{edge}}$.

Proof: (1) $v_{\mathbf{flow}}$**(G) = 2**$f_{\mathbf{edge}}$**(G)** for all **G**$\in \mathcal{G}_d$

(2) $\partial\, v_{\mathbf{flow}}$ **= 2** $\cdot\, \partial_d f_{\mathbf{edge}}$**= 2**$d$

Lipschitz extension of the degree list [RS '16]

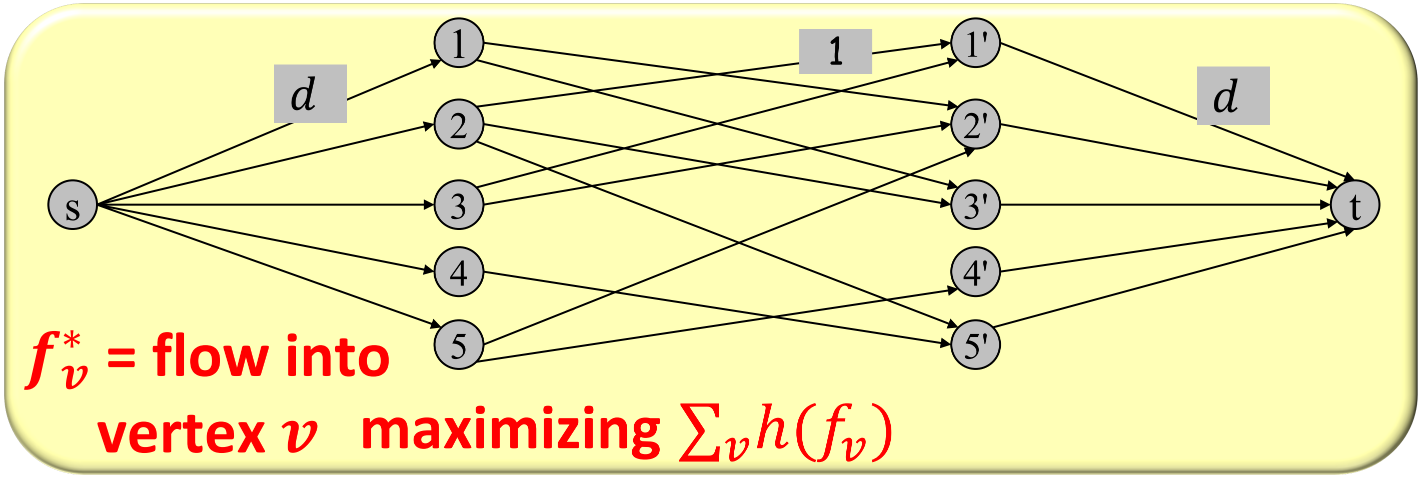


Can we use f_v as a proxy for degree of v?

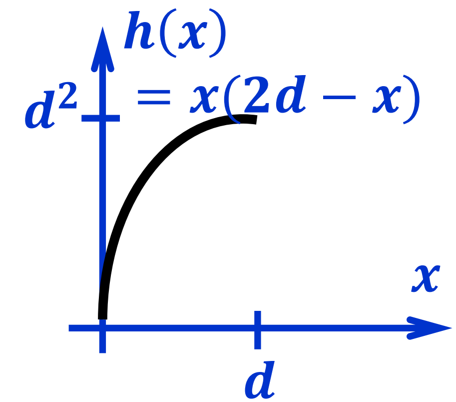
Issue: max flow is not unique.

Want: unique flow that has low global sensitivity.

Lipschitz extension of $f_{\mathbf{deg}}$: convex programming



Idea: maximize $\sum_v h(f_v)$ instead of $\sum_v f_v$,
where $h(x)$ is strictly concave.



- Let f^* be the vector of s-out-flows.
  - ➢ f^* is unique, since h is strictly concave.
  - ➢ Poly-time computable [Lee Rao Srivastava 13].
- **Lemma.** $\boldsymbol{f^*}$ is a Lipschitz extension of degree-list with stretch 3/2.

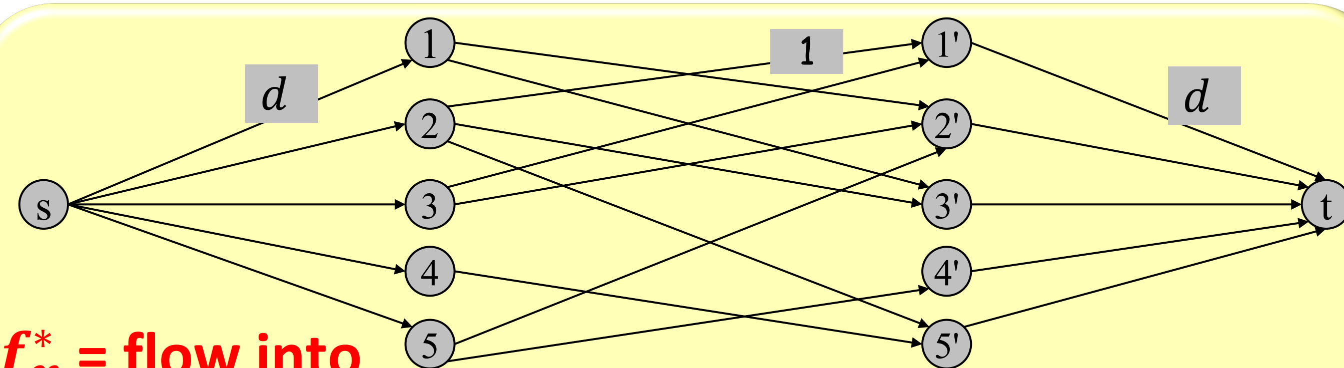
Lipschitz extension of $f_{\mathbf{deg}}$: combinatorial

- Subsequently simplified [W.-Y. Day, N. Li, M. Lyu, SIGNMOD 2016]
- **DLL Algorithm**: On input $G = (V, E)$ and degree bound d
  - Order the edges lexicographically: $E = e\_1, e_2, \ldots, e_m$
  - $G' = (V, \emptyset)$ //empty graph
  - **For** $i = 1$ to m:
    - **If** (adding e_i to G' would not push maximum degree over d)
      - Add e to V
    - **Else**
      - Ignore e_i
  - **Return** sort(degree-list(G'))

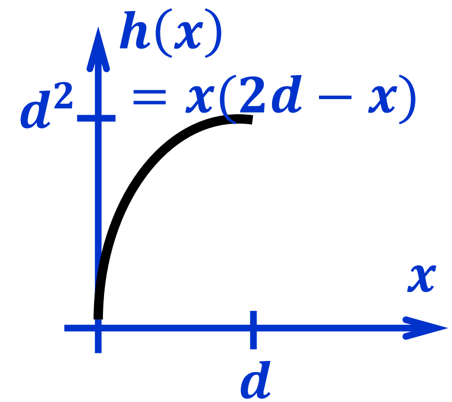
- **Lemma:** For any two graphs G_1, G_2 that differ in one node,

$$\|DLL(G_1) - DLL(G_2)\|_1 \leq 2d + 1$$

Releasing degree list: summary

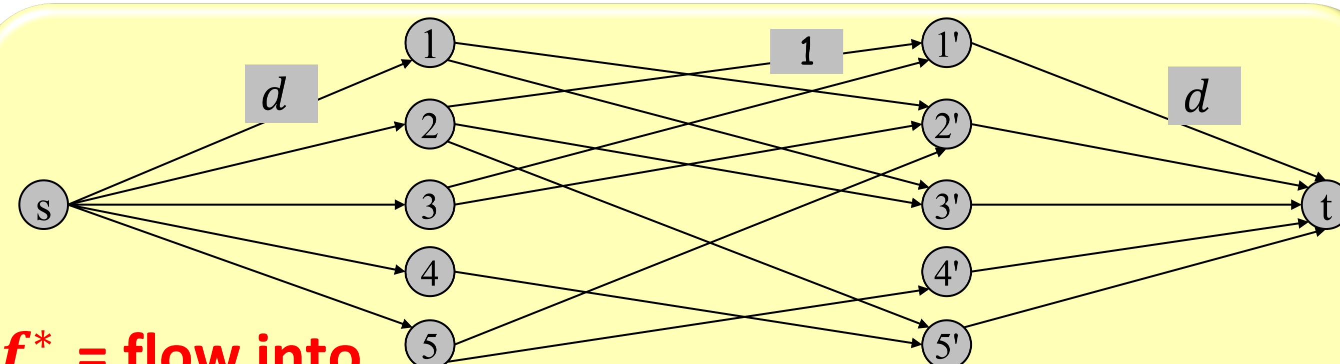


f_v^* = flow into vertex v maximizing $\sum_v h(f_v)$



1. Construct flow graph of G.
2. Compute s-out-flows f^*.
3. Release vector f^*, with $\mathrm{Lap}\left(\frac{3d}{\epsilon}\right)$ per coordinate.
4. Use post-processing techniques by [Hay Rastogi Miklau Suciu 09, Hay Li Miklau Jensen 09, Karwa Slavkovic 12, Kifer Lin 13] to remove some noise.

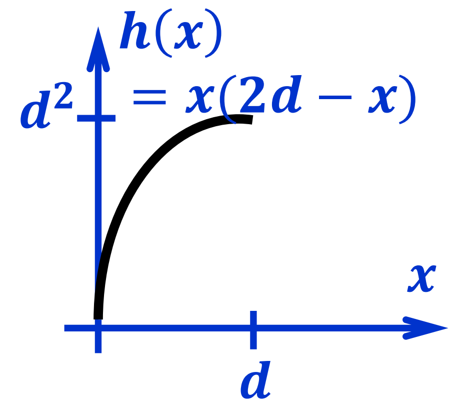
Lipschitz extension of degree list via convex programming



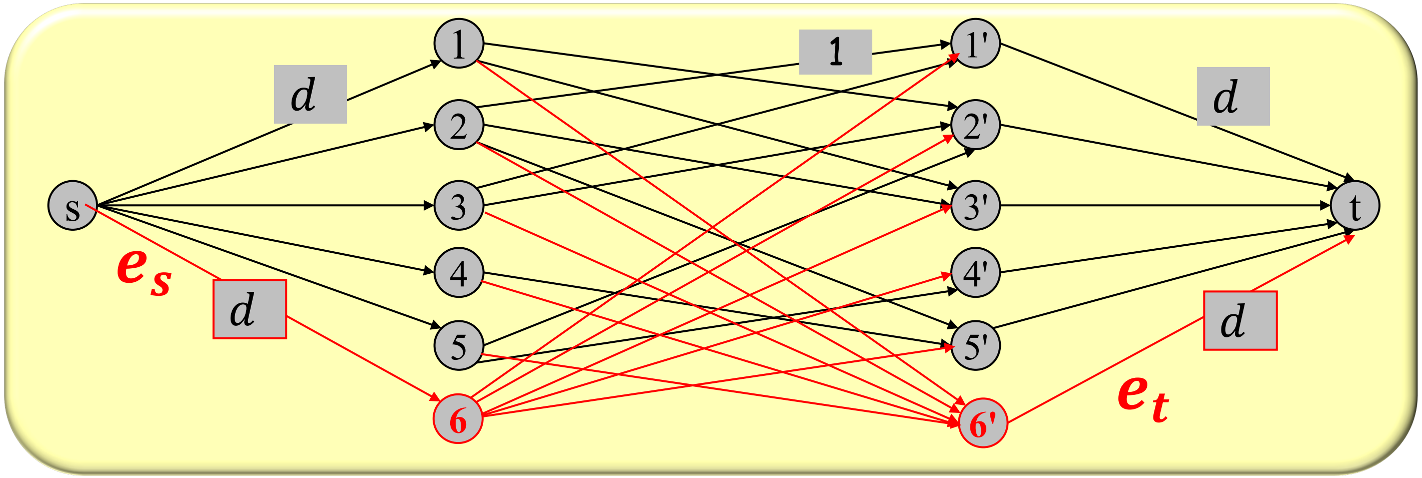
f_v^* = flow into vertex v

$$\phi = \text{argmax} \sum_v h(f_v).$$

- If **G**$\in \mathcal{G}_d$, then f_v^*= **deg(v) for all v**,
  since h is strictly increasing on [0,d].
- $\boldsymbol{\partial_d(degree\ list) = 2d}$:
  can add a node of degree $\leq d$.
- **Lemma.** ℓ_1 **global sensitivity** $\boldsymbol{\partial f^* \leq 3d}$**.**



Lipschitz extension of degree list via convex programming



Lemma. **ℓ_1 global sensitivity $\partial f^* \leq 3d$.**

Proof sketch: Consider $g = \phi_{new} - \phi_{old}$.

g is a union of simple s-t-paths and cycles of several types:

1. s-t-paths and cycles using e_s. Contribute $\leq 2d$ to $|f^*_{new} - f^*_{old}|_1$
2. s-t-paths using e_t. $\leq d$
3. Cycles using e_t. 0
4. Remaining paths and cycles. Do not exist.

Use strict concavity of h

Generalized Exponential Mechanism

Choosing the cutoff degree d

Evaluating Cutoff Degrees

Given: candidate real-valued Lipschitz extensions f_d and their sensitivities ∂f_d

(Specifically, we will try $d =$ powers of 2 in $[n]$)

- If we approximate f by releasing f_d in GS framework, the expected error is roughly $\boldsymbol{f(G) - f_d(G) + \frac{\partial f_d}{\epsilon}}$

Want: f_d with (approximately) smallest

$$\text{score } q_d(G) \coloneqq -f_d(G) + \frac{\partial f_d}{\epsilon}$$

Differentially private algorithms for choosing the item with the smallest score

- Exponential Mechanism [McSherry Talwar 07]
  - Initial motivation: auction design.
  - Subsequent applications:
    - Learning discrete classifiers [KLN**R**S'08]
    - Synthetic data generation [BLR'08,...,HLM'10]
    - Convex Optimization [CM'08,CMS'10]
    - Frequent Pattern Mining [BLST'10]
    - Genome-wide association studies [FUS'11]
    - High-dimensional sparse regression [KST'12]
    - ...

Differentially private algorithms for choosing the item with the smallest score

- Exponential Mechanism [McSherry Talwar 07]
  - Additive error in the score is proportional to **maximum** (over items) **sensitivity of a score** function
- Our Generalized Exponential Mechanism
  - Additive error in the score is proportional to the **sensitivity of the optimal score** function

Exponential Mechanism [McSherry Talwar 07]

Given: database x from universe U, parameter $\epsilon > 0$,

score functions $q_i\colon U \to \mathbb{R}$ with $\delta_i = \boldsymbol{\partial} q_i$ for $i \in [k]$

Want: index $i^* = \arg\min_i q_i(x)$

Algorithm EM

1. Set $\boldsymbol{\delta} = \mathbf{max}_i \boldsymbol{\delta}_i$
2. Output $\hat{\imath}$, set to i with probability $\propto \exp(\epsilon \cdot \boldsymbol{q_i(x)/\delta})$, normalized so that probabilities sum to 1, for $i \in [k]$.

Guarantees: (1) EM is 2ϵ-differentially private

(2) $\forall \beta \in (0,1)$, w.p. $\geq \beta$,

$$q_{\hat{\imath}}(x) \leq q_{i^*}(x) + \delta \cdot \frac{2\ln(k/\beta)}{\epsilon}$$

Goal: Guarantee with $\boldsymbol{\delta_{i^*}}$ instead of $\boldsymbol{\delta}$.

Generalized *Exponential Mechanism* [RS16]

Given: database x from universe U, $\epsilon > 0$, $\beta \in (0,1)$,
score functions q_i: $U \to \mathbb{R}$ with $\delta_i = \boldsymbol{\partial} q_i$ for $i \in [k]$

Want: index $i^* = \arg\min_i q_i(x)$

Algorithm GEM

1. $\boldsymbol{t \leftarrow 2\ln(k/\beta)/\epsilon}$
2. $\boldsymbol{q_i'(x) \leftarrow q_i(x) + t\delta_i}$ **for all** $\boldsymbol{i \in [k]}$
3. $\boldsymbol{s_i(x) \leftarrow \max_j \frac{q_i'(x) - q_j'(x)}{\delta_i + \delta_j}}$ **for all** $\boldsymbol{i \in [k]}$
4. Output $\hat{\imath}$, set to i with probability $\propto \exp(\epsilon \cdot \boldsymbol{s_i(x)})$, normalized so that probabilities sum to 1, for $i \in [k]$.

Note: each q_i' has sensitivity δ_i
each s_i has sensitivity 1

Generalized Exponential Mechanism [RS16]

Given: database x from universe U, $\epsilon > 0$, $\beta \in (0,1)$,
score functions $q_i\colon U \to \mathbb{R}$ with $\delta_i = \boldsymbol{\partial} q_i$ for $i \in [k]$

Want: index $i^* = \arg\min_i q_i(x)$

Guarantees: (1) GEM is 2ϵ-differentially private

(2) $\forall \beta \in (0,1)$, w.p. $\geq \beta$,

$$q_{\hat{\imath}}(x) \leq q_{i^*}(x) + \delta_{i^*} \cdot \frac{4\ln(k/\beta)}{\epsilon}$$

Summary of techniques we saw

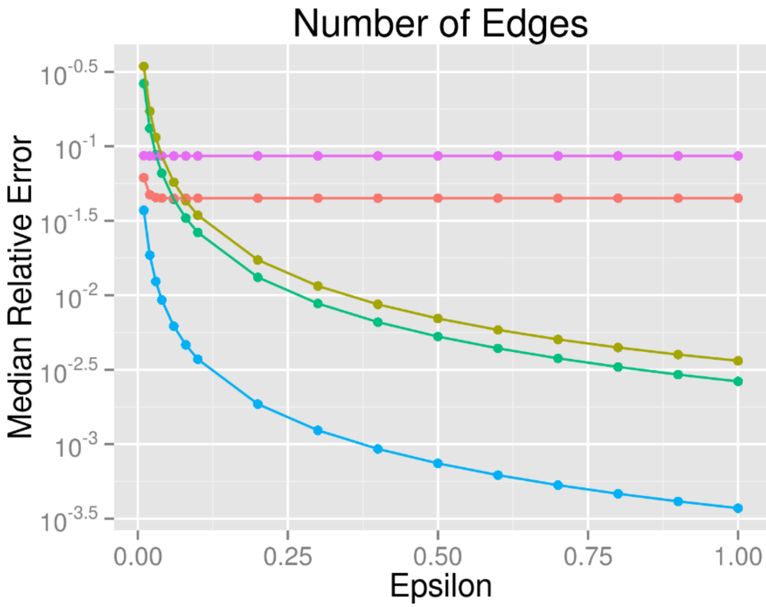
1. **Lipschitz extensions** [BBDS13, KNRS13]
   - Releasing number of edges **via max flow** [KNRS13]
   - Releasing degree list: **via convex programming** [RS16]
   - The most accurate known method
2. **Generalized exponential mechanism** [SR16]
   - For choosing among objects with score functions of different sensitivities
   - For choosing the cutoff degree

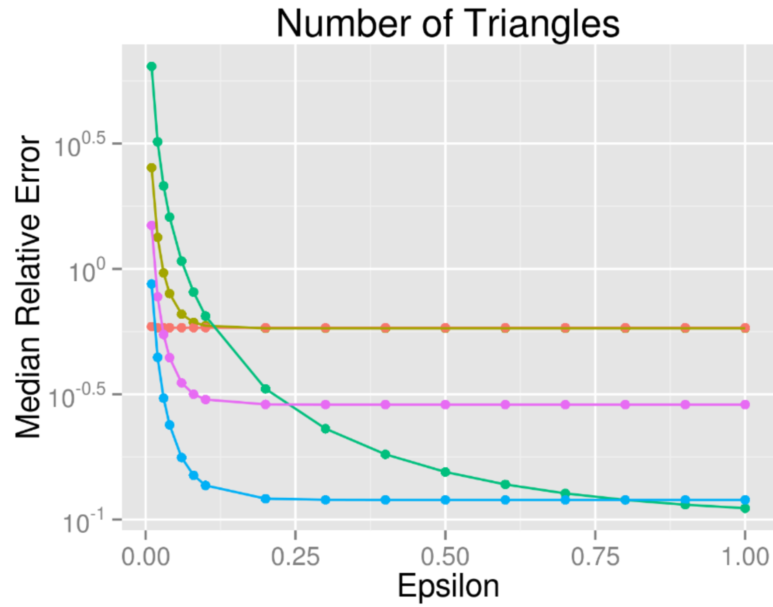
Conclusions

- It is possible to design node differentially private algorithms with good utility for a large class of graphs
  - One can choose a "good" value of d privately
- Directions for future work
  - Understanding which functions have efficiently computable Lipschitz extensions with small stretch
  - Node-private algorithms for releasing other graph statistics
  - Node-private synthetic graphs

- **Open Question:** Is there a node-differentially private algorithm for releasing the cost of all graph cuts with worst-case error $o(n \cdot \text{max-degree}(G))$?

Experiments for the flow and LP method [Lu]





Graph	# nodes	# edges	Max degree	Time, secs # edges	Time, secs # Δs
CA-GrQc	5,242	28,992	81	0.02	7
CA-HepTh	9,877	51,996	65	0.68	0.5
CA-AstroPh	18,772	396,220	504	0.34	10,222
com-dblp-ungraph	317,080	2,099,732	343	2	2128
com-youtube-ungraph	1,134,890	5,975,248	28,754	9	94