# Detecting Communities under Differential Privacy

Hiep Nguyen, Abdessamad Imine, Michaël Rusinowitch

# Detecting Communities under Differential Privacy

Hiep H. Nguyen
LORIA/INRIA Nancy
France
huu-hiep.nguyen@inria.fr

Abdessamad Imine
LORIA/INRIA Nancy
France
abdessamad.imine@loria.fr

Michaël Rusinowitch
LORIA/INRIA Nancy
France
michael.rusinowitch@inria.fr

## ABSTRACT

Complex networks usually expose community structure with groups of nodes sharing many links with the other nodes in the same group and relatively few with the nodes of the rest. This feature captures valuable information about the organization and even the evolution of the network. Over the last decade, a great number of algorithms for community detection have been proposed to deal with the increasingly complex networks. However, the problem of doing this in a private manner is rarely considered.

In this paper, we solve this problem under differential privacy, a prominent privacy concept for releasing private data. We analyze the major challenges behind the problem and propose several schemes to tackle them from two perspectives: input perturbation and algorithm perturbation. We choose Louvain method as the back-end community detection for input perturbation schemes and propose the method LouvainDP which runs Louvain algorithm on a noisy supergraph. For algorithm perturbation, we design ModDivisive using exponential mechanism with the modularity as the score. We have thoroughly evaluated our techniques on real graphs of different sizes and verified that ModDivisive steadily gives the best modularity and avg.F1Score on large graphs while LouvainDP outperforms the remaining input perturbation competitors in certain settings.

## Keywords

Differential privacy; Community detection; LouvainDP; ModDivisive

## 1. INTRODUCTION

Graphs represent a rich class of data observed in daily life where entities are described by nodes and their connections are characterized by edges. Apart from microscopic (node level) and macroscopic (graph level) configurations, many complex networks display a *mesoscopic* structure, i.e. they appear as a combination of components fairly independent of each other. These components are called communities, modules or clusters and the problem of how to reveal them plays a significant role in understanding the organization and function of complex networks. Over the last decade, a great number of algorithms for community detection (CD) have been proposed to address the problem in a variety of settings, such as undirected/directed, unweighted/weighted networks and non-overlapping/overlapping communities (for a comprehensive survey, see [10]).

These approaches, however, are adopted in a non-private manner, i.e. a data collector (such as Facebook) knows all the contributing users and their relationships before running CD algorithms. The output of such a CD, in the simplest form, is a clustering of nodes. Even in this case, i.e. where only a node clustering (not the whole graph) is revealed, contributing users privacy may still be put at risk. For example, thirteen nodes in Fig. 1 are clustered in four communities $[\{0,1,2\},\{3,4\},\{5,6,11,12\},\{7,8,9,10\}]$. Assuming that a new edge (0,3) is added, the new clustering of nodes may become $[\{0,1,2,3\},\{4,5,6,11,12\},\{7,8,9,10\}]$. By observing the clusterings before and after the addition of the edge (0,3), an attacker infers that there might have been new edge(s) between 3 and the nodes in the community {0,1,2}.

In this paper, we address the problem of CD from the perspective of differential privacy [8]. This privacy model offers a formal definition of privacy with a lot of interesting properties: no computational/informational assumptions about attackers, data type-agnosticity, composability and so on [16]. By differential privacy, we want to ensure the existence of connections between users to be hidden in the output clustering while keeping the low distortion of clusters compared to the ones generated by the corresponding non-private algorithms.

As far as we know, the problem is quite new and only mentioned in the recent work [18] where Mülle et al. use a randomized response technique [9] to perturb the input graph so that it satisfies differential privacy before running conventional CD algorithms. This scheme (we call it *EdgeFlip* afterwards) is classified as *input perturbation* in differential privacy literature (the other two categories are *algorithm perturbation* and *output perturbation*). Similarly, *TmF* approach [20] can apply to the true graph to get noisy output graphs as in the work of Mülle et al. Earlier, 1k-Series [28], *Density Explore Reconstruct* (DER) [4] and HRG-MCMC [29] are the best known methods for graph structure release under differential privacy. These methods can be followed by any exact CD algorithm to get a noisy clustering satisfying differential privacy. We choose Louvain method [2] as such a CD algorithm. However, as we will see in the exper-

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

iments, the output clusterings by the aforementioned methods have very low modularity scores. This fact necessitates new methods for CD problem under differential privacy.

Our main contributions are the new schemes *LouvainDP* (input perturbation) and *ModDivisive* (algorithm perturbation) which perform much better than the state-of-the-art. LouvainDP is a high-pass filtering method that randomly groups nodes into supernodes of equal size to build a weighted supergraph. LouvainDP is guaranteed to run in linear time. ModDivisive is a top-down approach which privately divides the node set into the k-ary tree guided by the modularity score at each level. The main technique used in ModDivisive is the Markov Chain Monte Carlo (MCMC) to realize the exponential mechanism [15]. We show that ModDivisive's runtime is linear in the number of nodes, the height of the binary tree and the burn-in factor of MCMC. The linear complexity enables us to examine million-scale graphs in a few minutes. The experiments show the high modularity and low distortion of the output clusters by LouvainDP and ModDivisive.

Our contributions are summarized as follows:

- We analyze the major challenges of community detection under differential privacy ( Section 3). We explain why techniques borrowed from k-Means fail and how the difficulty of $\epsilon$-DP recommender systems justifies a relaxation of $\epsilon$.
- We design an input perturbation scheme LouvainDP (Section 4) that runs in linear time using the high-pass filtering technique from [7] and Louvain method [2].
- We propose an algorithm perturbation scheme ModDivisive (Section 5) as a divisive approach by using the modularity-based score function in the exponential mechanism. We prove that modularity has small global sensitivity and ModDivisive also runs in linear time.
- We conduct a thorough evaluation on real graphs of different sizes and show the outperformance of LouvainDP and ModDivisive over the state-of-the-art (Section 6).

Table 1 summarizes the key notations used in this paper.

Table 1: List of notations

| Symbol | Definition |
|---|---|
| $G = (V, E_G)$ | true graph with $n = \|V\|$ and $m = \|E_G\|$ |
| $G' = (V, E_{G'})$ | neighboring graph of $G$ |
| $\tilde{G} = (V, E_{\tilde{G}})$ | sample noisy output graph |
| $G_1 = (V_1, E_1)$ | supergraph generated by LouvainDP |
| $k$ | fan-out of the tree in ModDivisive |
| $K$ | burn-in factor in MCMC-based algorithms |
| $\lambda$ | common ratio to distribute the privacy budget |
| $C$ | a clustering of nodes in $G$ |
| $Q(G, C)$ | modularity of the clustering $C$ on graph $G$ |

# 2. RELATED WORK

## 2.1 Community Detection in Graphs

There is a vast literature on community detection in graphs. For a recent comprehensive survey, we refer to [10]. In this section, we discuss several classes of techniques.

Newman and Girvan [19] propose *modularity* as a quality of network clustering. It is based on the idea that a random graph is not expected to have a modular structure, so the possible existence of clusters is revealed by the comparison between the actual density of edges in a subgraph and the density one would expect to have in the subgraph if the nodes of the graph were connected randomly (the null model). The modularity $Q$ is defined as

$$Q = \sum_{c=1}^{n_c} \left[ \frac{l_c}{m} - \left( \frac{d_c}{2m} \right)^2 \right] \quad (1)$$

where $n_c$ is the number of clusters, $l_c$ is the total number of edges joining nodes in community $c$ and $d_c$ is the sum of the degrees of the nodes of $c$.

Many methods for optimizing the modularity have been proposed over the last ten years, such as agglomerative greedy [6], simulated annealing [17], random walks [22], statistical mechanics [25], label propagation [24] or InfoMap [26], just to name a few. The recent multilevel approach, also called *Louvain method*, by Blondel et al. [2] is a top performance scheme. It scales very well to graphs with hundreds of millions of nodes. This is the chosen method for the input perturbation schemes considered in this paper (see Sections 3.2 and 4 for more detail). By maximizing the modularity, Louvain method is based on *edge counting* metrics, so it fits well with the concept of *edge differential privacy* (Section 3.1). One of the most recent methods *SCD* [23] is not chosen because it is about maximizing Weighted Community Clustering (WCC) instead of the modularity. WCC is based on *triangle counting* which has high global sensitivity (up to $O(n)$) [31]. Moreover, SCD pre-processes the graph by removing all edges that do not close any triangle. This means all 1-degree nodes are excluded and form singleton clusters. The number of output clusters is empirically up to $O(n)$.

## 2.2 Graph Release via Differential Privacy

In principle, after releasing a graph satisfying $\epsilon$-DP, we can do any mining operations on it, including community detection. The research community, therefore, expresses a strong interest in the problem of graph release via differential privacy. Differentially private algorithms relate the amount of noise to the computation sensitivity. Lower sensitivity implies smaller added noise. Because the edges in simple undirected graphs are usually assumed to be independent, the standard Laplace mechanism [8] is applicable (e.g. adding Laplace noise to each cell of the adjacency matrix). However, this approach severely deteriorates the graph structure.

The state-of-the-art [4, 29] try to reduce the sensitivity of the graph in different ways. *Density Explore Reconstruct* (DER)[4] employs a data-dependent quadtree to summarize the adjacency matrix into a counting tree and then reconstructs noisy sample graphs. DER is an instance of input perturbation. Xiao et al. [29] propose to use *Hierarchical Random Graph* (HRG) [5] to encode graph structural information in terms of edge probabilities. Their scheme HRG-MCMC is argued to be able to sample good HRG models which reflect the community structure in the original graph. HRG-MCMC is classified as an algorithm perturbation scheme. A common disadvantage of the state-of-the-

art DER and HRG-MCMC is the scalability issue. Both of them incur quadratic complexity $O(n^2)$, limiting themselves to medium-sized graphs.

Recently, Nguyen et al. [20] proposed *TmF* that utilizes a filtering technique to keep the runtime linear in the number of edges. TmF also proves the upper bound $O(\ln n)$ for the privacy budget $\epsilon$. At the same time, Mülle et al. [18] devised the scheme *EdgeFlip* using a randomized response technique. However, EdgeFlip costs $O(n^2)$ and is runnable only on graphs of tens of thousands of nodes.

# 3. PRELIMINARIES

In this section, we review key concepts and mechanisms of differential privacy.

## 3.1 Differential Privacy

Essentially, $\epsilon$-differential privacy ($\epsilon$-DP) [8] is proposed to quantify the notion of *indistinguishability* of neighboring databases. In the context of graph release, two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are neighbors if $V_1 = V_2$, $E_1 \subset E_2$ and $|E_2| = |E_1| + 1$. Note that this notion of neighborhood is called *edge differential privacy* in contrast with the notion of *node differential privacy* which allows the addition of one node and its adjacent edges. Our work follows the common use of edge differential privacy. The formal definition of $\epsilon$-DP for graph data is as follows.

DEFINITION 3.1. *A mechanism $\mathcal{A}$ is $\epsilon$-differentially private if for any two neighboring graphs $G_1$ and $G_2$, and for any output $O \in Range(\mathcal{A})$,*

$$Pr[\mathcal{A}(G_1) \in O] \leq e^{\epsilon} Pr[\mathcal{A}(G_2) \in O]$$

Laplace mechanism [8] and Exponential mechanism [16] are two standard techniques in differential privacy. The latter is a generalization of the former. Laplace mechanism is based on the concept of *global sensitivity* of a function $f$ which is defined as $\Delta f = \max_{G_1,G_2} ||f(G_1) - f(G_2)||_1$ where the maximum is taken over all pairs of neighboring $G_1, G_2$. Given a function $f$ and a privacy budget $\epsilon$, the noise is drawn from a Laplace distribution $Lap(\lambda) : p(x|\lambda) = \frac{1}{2\lambda} e^{-|x|/\lambda}$ where $\lambda = \Delta f/\epsilon$.

THEOREM 3.1. *(Laplace mechanism [8]) For any function $f : G \to \mathbb{R}^d$, the mechanism $\mathcal{A}$*

$$\mathcal{A}(G) = f(G) + \langle Lap_1(\frac{\Delta f}{\epsilon}), ..., Lap_d(\frac{\Delta f}{\epsilon}) \rangle \qquad (2)$$

*satisfies $\epsilon$-differential privacy, where $Lap_i(\frac{\Delta f}{\epsilon})$ are i.i.d Laplace variables with scale parameter $\frac{\Delta f}{\epsilon}$.* □

*Geometric mechanism* [11] is a discrete variant of Laplace mechanism with integral output range $\mathbb{Z}$ and random noise $\Delta$ generated from a two-sided geometric distribution $Geom(\alpha) : Pr[\Delta = \delta|\alpha] = \frac{1-\alpha}{1+\alpha}\alpha^{|\delta|}$. To satisfy $\epsilon$-DP, we set $\alpha = \exp(-\epsilon)$. We use geometric mechanism in our LouvainDP scheme.

For non-numeric data, the exponential mechanism is a better choice [15]. Its main idea is based on sampling an output $O$ from the output space $\mathcal{O}$ using a score function $u$. This function assigns exponentially higher probabilities to outputs of higher scores. Let the global sensitivity of $u$ be $\Delta u = \max_{O,G_1,G_2} |u(G_1, O) - u(G_2, O)|$.

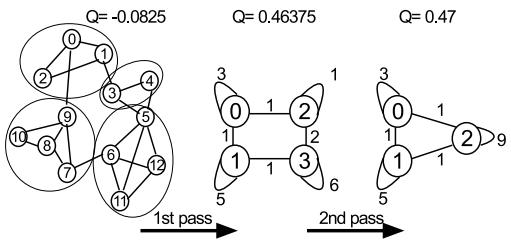



Figure 1: Louvain method

THEOREM 3.2. *(Exponential mechanism [15]) Given a score function $u : (G \times \mathcal{O}) \to \mathbb{R}$ for a graph $G$, the mechanism $\mathcal{A}$ that samples an output $O$ with probability proportional to $\exp(\frac{\epsilon.u(G,O)}{2\Delta u})$ satisfies $\epsilon$-differential privacy.* □

Composability is a nice property of differential privacy which is not satisfied by other privacy models such as k-anonymity.

THEOREM 3.3. *(Sequential and parallel compositions [16]) Let each $A_i$ provide $\epsilon_i$-differential privacy. A sequence of $A_i(D)$ over the dataset $D$ provides $\Sigma_{i=1}^{n} \epsilon_i$-differential privacy.*

*Let each $A_i$ provide $\epsilon_i$-differential privacy. Let $D_i$ be arbitrary disjoint subsets of the dataset $D$. The sequence of $A_i(D_i)$ provides $\max_{i=1}^{n} \epsilon_i$-differential privacy.* □

## 3.2 Louvain Method

Since its introduction in 2008, Louvain method [2] becomes one of the most cited methods for the community detection task. It optimizes the modularity by a bottom-up folding process. The algorithm is divided in passes each of which is composed of two phases that are repeated iteratively. Initially, each node is assigned to a different community. So, there will be as many communities as there are nodes in the first phase. Then, for each node $i$, the method considers the gain of modularity if we move $i$ from its community to the community of a neighbor $j$ (a *local change*). The node $i$ is then placed in the community for which this gain is maximum and positive (if any), otherwise it stays in its original community. This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved and the first pass is then complete.

We demonstrate Louvain method in Fig.1 by a graph of 13 nodes and 20 edges. If each node forms its own singleton community, the modularity Q will be -0.0825. In the first pass of Louvain method, each node moves to the best community selected from its neighbors' communities. We get the partition $[\{0, 1, 2\}, \{3, 4\}, \{5, 6, 11, 12\}, \{7, 8, 9, 10\}]$ with modularity 0.46375. The second phase of first pass builds a weighted graph corresponding to the partition by aggregating communities. The second pass repeats the folding process on this weighted graphs to reach the final partition $[\{0, 1, 2\}, \{3, 4, 5, 6, 11, 12\}, \{7, 8, 9, 10\}]$ with modularity 0.47.

This simple agglomerative algorithm has several advantages as stated in [2]. First, its steps are intuitive and easy to implement, and the outcome is unsupervised. Second, the algorithm is extremely fast, i.e. computer simulations on large modular networks suggest that its complexity is linear on typical and sparse data. This is due to the fact that the possible gains in modularity are easy to compute

and the number of communities decreases drastically after just a few passes so that most of the running time is concentrated on the first iterations. Third, the multi-level nature of the method produces a hierarchical structure of communities which allows multi-resolution analysis, i.e the user can zoom in the graph to observe its structure with the desired resolution. In addition, Louvain method is runnable on weighted graphs. This fact supports naturally our scheme LouvainDP as described in Section 4.1.

## 3.3 Challenges of Community Detection under Differential Privacy

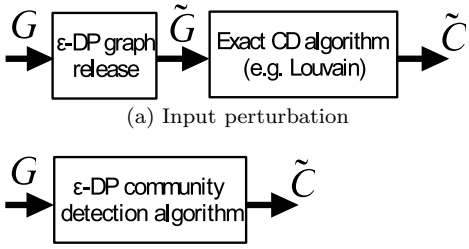
In this section, we explain why community detection under differential privacy is challenging. We show how techniques borrowed from related problems fail. We also advocate the choice of $\epsilon$ as a function of the graph size $n$.

The problem of differentially private community detection is closely related to $\epsilon$-DP k-Means clustering and recommender systems. The $\epsilon$-DP k-Means is thoroughly discussed in [27]. However, techniques from $\epsilon$-DP k-Means are not suitable to $\epsilon$-DP community detection. First, items in k-Means are in low-dimensional spaces and the number of clusters $k$ is usually small. This contrast to the case of community detection where nodes lie in a $n$-dimensional space and the number of communities varies from tens to tens of thousands, not to say the communities may overlap or be nested (multi-scale). Second, items in $\epsilon$-DP k-Means are normalized to $[-1, 1]^d$ while the same preprocessing seems invalid in $\epsilon$-DP community detection. Moreover, the output of k-Means usually consists of equal-sized balls while this is not true for communities in graphs. Considering the graph as a high-dimensional dataset, we tried the private projection technique in [14] which is followed by spectral clustering, but the modularity scores of the output are not better than random clustering.

Recent papers on $\epsilon$-DP recommender systems [12, 1] show that privately learning the clustering of items from user ratings is hard unless we relax the value of $\epsilon$ up to $\log n$. Banerjee et al. [1] model differentially private mechanisms as noisy channels and bound the mutual information between the generative sources and the privatized sketches. They show that in the information-rich regime (each user rates $O(n)$ items), their *Pairwise-Preference* succeeds if the number of users is $\Omega(n \log n/\epsilon)$. Compared to $\epsilon$-DP community detection where the number of users is $n$, we should have $\epsilon = \Omega(\log n)$. Similarly, in D2P scheme, Guerraoui et al. [12] draw a formula for $\epsilon$ as

$$\epsilon_{D2P}^{(p,0,\lambda)} = \ln(1 + \frac{(1-p).\mathcal{N}_E}{p.|\mathcal{G}_\lambda|}) \quad (3)$$

where $\lambda$ is the distance used to conceal the user profiles ($\lambda = 0$ reduces to the classic notion of differential privacy). $\mathcal{N}_E$ is the number of items which is exactly $n$ in community detection. $|\mathcal{G}_\lambda|$ is the minimum size of user profiles at distance $\lambda$ over all users ($|\mathcal{G}_\lambda| = o(\mathcal{N}_E)$ except at unreasonably large $\lambda$). Clearly, at $p = 0.5$ (as used in [12]), we have $\epsilon_{D2P}^{(0.5,0,\lambda)} \approx \ln n$. The randomized technique in D2P is very similar to EdgeFlip [18] which is shown ineffective in $\epsilon$-DP community detection for $\epsilon \in (0, 0.5 \ln n)$ (Section 6). Note that $\epsilon$-DP community detection is unique in the sense that the set of items and the set of users are the same. Graphs for community detection are more general than bipartite graphs in recommender systems. In addition, modularity $Q$ (c.f. Formula 1) is *non-monotone*, i.e. for two disjoint sets of nodes $A$ and $B$, $Q(A \cup B)$ may be larger, smaller than or equal to $Q(A) + Q(B)$.



(a) Input perturbation

(b) Algorithm perturbation

Figure 2: Two categories of $\epsilon$-DP community detection

To further emphasize the difficulty of $\epsilon$-DP community detection, we found that IDC scheme [13] using *Sparse Vector Technique* [9, Section 3.6] is hardly feasible. As shown in Algorithm 1 of [13], to publish a noisy graph that can approximately answer all cut queries with bounded error $m^{0.25}n/\epsilon^{0.5}$, IDC must have $B(\alpha)$ "yes" queries among all $k$ queries. $B(\alpha)$ may be as low as $\sqrt{m}$ but $k = 2^{2n}$. In the average case, IDC incurs exponential time to complete.

The typical epsilon in the literature is 1.0 or less. However, this value is only applicable to graph metrics of low sensitivity $O(1)$ such as the number of edges, the degree sequence. The global sensitivity of other metrics like the diameter, the number of triangles, 2K-series etc. is $O(n)$, calling for smooth sensitivity analysis (e.g. [21]). For counting queries, Laplace/Geometric mechanisms are straightforward on real/integral (*metric*) spaces. However, *direct* noise adding mechanisms on the space $\mathcal{P}$ of all ways to partition the nodeset $V$ are non-trivial because $|\mathcal{P}| \approx n^n$ and $\mathcal{P}$ is non-metric. Compared to $\epsilon$-DP recommender systems discussed above which use $\epsilon = \ln n$ for the super-exponential spaces of size $O(2^{n(n-1)/2})$, $\epsilon$-DP community detection clearly needs lower privacy budget.

To conclude, $\epsilon$-DP community detection is challenging and requires new techniques. In this paper, we evaluate the schemes for $\epsilon$ up to $0.5 \ln n$. At $\epsilon = 0.5 \ln n$, the multiplicative ratio (c.f. Definition 3.1) is $e^\epsilon = e^{0.5 \ln n} = \sqrt{n}$. We believe it is a reasonable threshold for privacy protection compared to $\epsilon = \ln n$ (i.e. $e^\epsilon = n$) in $\epsilon$-DP graph release and $\epsilon$-DP recommender systems.

# 4. INPUT PERTURBATION

In this section, we propose the linear scheme LouvainDP that uses a filtering technique to build a noisy weighted supergraph and calls the exact Louvain method subsequently. Then we discuss several recent $\epsilon$-DP schemes that can be classified as input perturbation. Fig.2a sketches the basic steps of the input perturbation paradigm.

## 4.1 LouvainDP: Louvain Method on Noisy Supergraphs

The basic idea of LouvainDP is to create a noisy weighted supergraph $G_1$ from $G$ by grouping nodes into supernodes of equal size $k$. We then apply the filtering technique of Cormode et al. [7] to ensure that there are only $O(m)$ noisy weighted edges in $G_1$. Finally, we run the exact community detection on $G_1$.

In [7], Cormode et al. propose several summarization techniques for sparse data under differential privacy. Let $M$ be a contingency table having the domain size $m_0$ and $m_1$ non-zero entries ($m_1 \ll m_0$ for sparse data), the conventional publication of a noisy table $M'$ from $M$ that satisfies $\epsilon$-DP requires the addition of Laplace/geometric noise to $m_0$ entries. The entries in $M'$ could be filtered (e.g. removing negative ones) and/or sampled to get a noisy summary $M''$. This direct approach would be infeasible for huge domain sizes $m_0$. Techniques in [7] avoid materializing the vast noisy data by computing the summary $M''$ directly from $M$ using filtering and sampling techniques.

In our LouvainDP, the supergraph $G_1$ is an instance of sparse data with the domain size $m_0 = \frac{n_1(n_1+1)}{2}$ where $n_1$ is the number of supernodes and $m_1$ is the number of non-zero entries corresponding to non-zero superedges. We use the one-sided filtering [7] to efficiently compute $G_1$ with $O(m)$ edges in linear time.

#### 4.1.1 Algorithm

LouvainDP can run with either geometric or Laplace noise. We describe the version with geometric noise in Algorithm 1.

**Algorithm 1** LouvainDP($G, s$)

**Input:** undirected graph $G$, group size $k$, privacy budget $\epsilon$
**Output:** noisy partition $\tilde{C}$

1: $G_1 \leftarrow \varnothing$, $n_1 = \lfloor \frac{|V|}{k} \rfloor - 1$, $V_1 \leftarrow \{0, 1, .., n_1\}$
2: $\epsilon_2 = 0.01$, $\epsilon_1 = \epsilon - \epsilon_2$, $\alpha = \exp(-\epsilon_1)$
3: get a random permutation $V_p$ of $V$
4: compute the mapping $M : V_p \rightarrow V_1$
5: compute superedges of $G_1$: $E_1 = \{e_1(i,j)\}$ where $i, j \in V_1$
6: $m_1 = |E_1| + Lap(1/\epsilon_2)$, $m_0 = \frac{n_1(n_1+1)}{2}$
7: $\theta = \lceil \log_\alpha \frac{(1+\alpha)m_1}{m_0 - m_1} \rceil$
8: $s = (m_0 - m_1)\frac{\alpha^\theta}{1+\alpha}$
9: **for** $e_1(i,j)$ in $E_1$ **do**
10: $\quad e_1(i,j) = e_1(i,j) + Geom(\alpha)$
11: $\quad$ **if** $e_1(i,j) \geq \theta$ **then**
12: $\quad\quad$ add $e_1(i,j)$ to $G_1$
13: **for** $s$ edges $e_0(i,j) \notin E_1$ sampled uniformly at random **do**
14: $\quad$ draw $w$ from the distribution $Pr[X \leq x] = 1 - \alpha^{x-\theta+1}$
15: $\quad$ **if** $w > 0$ **then**
16: $\quad\quad$ add edge $e_0(i,j)$ with weight $w$ to $G_1$
17: run Louvain method on $G_1$ to get $\tilde{C}_1$
18: compute $\tilde{C}$ from $\tilde{C}_1$ using the mapping $M$
19: **return** $\tilde{C}$

Given the group size $k$, LouvainDP starts with a supergraph $G_1$ having $\lfloor \frac{|V|}{k} \rfloor$ nodes by randomly permuting the nodeset $V$ and grouping every $k$ consecutive nodes into a supernode (lines 1-4). The permutation prevents the possible bias of node ordering in $G$. The set of superedges $E_1$ is easily computed from G. Note that $m_1 = |E_1| \leq m$ due to the fact that each edge of $G$ appears in one and only one superedge. The domain size is $m_0 = \frac{n_1(n_1+1)}{2}$ (i.e. we consider all selfloops in $G_1$). The noisy number of non-zero superedges is $m_1 = |E_1| + Lap(1/\epsilon_2)$. Then by *one-sided* filtering [7], we estimate the threshold $\theta$ (line 7) and the number of passing zero superedges $s$ (line 8). For each non-zero superedge, we add a geometric noise and add the superedge to $G_1$ if the noisy value is not smaller than $\theta$. For $s$ zero superedges $e_1(i,j) \notin E_1$, we draw an integral weight $w$ from the distribution $Pr[X \leq x] = 1 - \alpha^{x-\theta+1}$ and add $e_1(i,j)$ with weight $w$ to $G_1$ if $w > 0$.

#### 4.1.2 Complexity

LouvainDP runs in $O(m)$ because the loops to compute superedges (Line 5) and to add geometric noises (lines 9-12) cost $O(m)$. We have $s = (m_0 - m_1)\frac{\alpha^\theta}{1+\alpha} \leq \frac{m_0 - m_1}{1+\alpha}\frac{(1+\alpha)m_1}{m_0 - m_1} = m_1$ (see Line 7). So the processing of $s$ zero-superedges costs $O(m)$. Moreover, Louvain method (line 17) is empirically linear in $m_1$ [2]. We come up with the following theorem.

THEOREM 4.1. *The number of edges in $G_1$ is not larger than $2m$. LouvainDP's runtime is $O(m)$.*

#### 4.1.3 Privacy Analysis

In LouvainDP, we use a small privacy budget $\epsilon_2 = 0.01$ to compute the noisy number of non-zero superedges $m_1$. The remaining privacy budget $\epsilon_1$ is used for the geometric mechanism $Geom(\alpha)$. Note that getting a random permutation $V_p$ (line 3) costs no privacy budget. The number of nodes $n$ is public and given the group size $k$, the number of supernodes $n_1$ is also public. The high-pass filtering technique (Lines 6-16) inherits the privacy guarantee by [7]. By setting $\epsilon_1 = \epsilon - \epsilon_2$, LouvainDP satisfies $\epsilon$-differential privacy (see the sequential composition (Theorem 3.3)).

### 4.2 Alternative Input Perturbation Schemes

1K-series [28], DER [4], TmF [20] and EdgeFlip [18] are the most recent differentially private schemes for graph release that can be classified as input perturbation. While 1K-series and TmF run in linear time, DER and EdgeFlip incur a quadratic complexity. DER and EdgeFlip are therefore tested only on two medium-sized graphs in Section 6.

The expected number of edges by EdgeFlip is $|E_{\tilde{G}}| = (1-s)m + \frac{n(n-1)}{4}s$ (see [18]) where $s = \frac{2}{e^\epsilon + 1}$ is the flipping probability. Substitute $s$ into $|E_{\tilde{G}}|$, we get $|E_{\tilde{G}}| = m + (\frac{n(n-1)}{4} - m)\frac{2}{e^\epsilon+1}$. The number of edges in the noisy graph $\tilde{G}$ generated by EdgeFlip increases exponentially as $\epsilon$ decreases. To ensure the linear complexity for million-scale graphs, we propose a simple extension of EdgeFlip, called *EdgeFlipShrink* (see Appendix A.1).

## 5. ALGORITHM PERTURBATION

The schemes in the algorithm perturbation category privately sample a node clustering from the input graph without generating noisy sample graphs as in the input perturbation. This can be done via the exponential mechanism. We introduce our main scheme *ModDivisive* in Section 5.1 followed by *HRG-Fixed*, a variant of HRG-MCMC [29] runnable on large graphs (Section 5.2). Fig.2b sketches the basic steps of the algorithm perturbation paradigm.

### 5.1 ModDivisive: Top-down Exploration of Cohesive Groups

#### 5.1.1 Overview

In contrast with the agglomerative approaches (e.g. Louvain method) in which small communities are iteratively merged if doing so increases the modularity, our ModDivisive scheme is a divisive algorithm in which communities at each level are iteratively split into smaller ones. Our goal is to heuristically detect cohesive groups of nodes in a

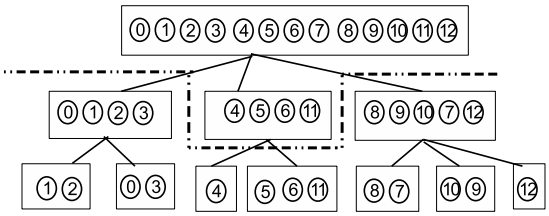



Figure 3: Example of ModDivisive with $k = 3$. A cut $C$ is shown by the dot-dashed line

private manner. There are several technical challenges in this process. The first one is how to efficiently find a good split of nodes that induces a high modularity and satisfies $\epsilon$-DP at the same time. The second one is how to merge the small groups to larger ones. We cope with the first challenge by realizing an exponential mechanism via MCMC (Markov Chain Monte-Carlo) sampling with the modularity as the score function (see Theorem 3.2). The second challenge is solved by dynamic programming. We design ModDivisive as a *k-ary* tree (Fig.3), i.e. each internal node has no more than $k$ child nodes. The root node (level 0) contains all nodes in $V$ and assigns arbitrarily each node to one of the $k$ groups. Then we run the MCMC over the space of all partitions of $V$ into no more than $k$ nonempty subsets. The resulting subsets are initialized as the child nodes (level 1) of the root. The process is repeated for each child node at level 1 and stops at level $maxL$. Fig.3 illustrates the idea with $k = 3$ for the graph in Fig.1.

### 5.1.2 Algorithm

Algorithm 2 sketches the main steps in our scheme ModDivisive. It comprises two phases: differentially private sampling a k-ary tree of depth $maxL$ which uses the privacy budget $\epsilon_1$ and finding the best cut across the tree to get a good clustering of nodes which consumes a budget $maxL \cdot \epsilon_m$.

The first phase (lines 1-14) begins with the creation of $eA$, the array of privacy budgets allocated to levels of the tree. We use the parameter $\lambda \geq 1$ as the common ratio to form a geometric sequence. The rationale behind the common ratio is to give higher priority to the levels near the root which have larger node sets. By sequential composition (Theorem 3.3), we must have $\sum_i eA[i] = \epsilon_1$. All internal nodes at level $i$ do the MCMC sampling on disjoint subsets of nodes, so the parallel composition holds (Theorem 3.3). Subsection 5.1.4 analyzes the privacy of ModDivisive in more detail. We use a queue to do a level-by-level exploration. Each dequeued node $r$'s level will be checked. If its level is not larger than $maxL$, we will run *ModMCMC* (Algorithm 3) on it (line 9) to get a partition $r.part$ of its nodeset $r.S$ (Fig.3). Each subset in $r.part$ forms a child node and is pushed to the queue. The second phase (line 15) calls *BestCut* to find a highly modular partition across the tree.

***Differentially Private Nodeset Partitioning*** Let $\mathcal{P}$ be the space of all ways $P$ to partition a nodeset $A$ to no more than $k$ disjoint subsets, the direct application of exponential mechanism needs the enumeration of $\mathcal{P}$. The probability of a partition $P$ being sampled is

$$\frac{\exp(\frac{\epsilon_p}{2\Delta Q}Q(P,G))}{\sum_{P'\in\mathcal{P}}\exp(\frac{\epsilon_p}{2\Delta Q}Q(P',G))} \quad (4)$$

**Algorithm 2** ModDivisive

**Input:** graph $G$, group size $k$, privacy budget $\epsilon$, max level $maxL$, ratio $\lambda$, BestCut privacy at each level $\epsilon_m$
**Output:** noisy partition $\tilde{C}$

```
1: compute the array eA[0..maxL − 1] s.t. ∑_i eA[i] = ε_1, eA[i] = eA[i + 1] ∗ λ where ε_1 = ε − maxL.ε_m
2: initialize the root node with nodeset V
3: root = NodeSet(G, V, k)
4: root.level = 0
5: queue Q ← root
6: while Q is not empty do
7:     r ← Q.dequeue()
8:     if r.level < maxL then
9:         r.part = ModMCMC(G, r.S, k, eA[r.level])
10:        for subset S_i in r.part do
11:            P_i = NodeSet(G, S_i, k)
12:            P_i.level = r.level + 1
13:            r.child_i ← P_i
14:            Q.enqueue(P_i)
15: C̃ ← BestCut(root, ε_m)
16: return C̃
```

However, $|\mathcal{P}| = \sum_{i=1}^{k} S(|A|, i)$ where $S(|A|, i)$ is the Stirling number of the second kind, $S(n,k) \approx \frac{k^n}{k!}$. This sum is exponential in $|A|$, so enumerating $\mathcal{P}$ is computationally infeasible. Fortunately, MCMC can help us simulate the exponential mechanism by a sequence of local transitions in $\mathcal{P}$.

The space $\mathcal{P}$ is connected. It is straightforward to verify that the transitions performed in line 3 of ModMCMC are *reversible* and *ergodic* (i.e. any pair of nodeset partitions can be connected by a sequence of such transitions). Hence, ModMCMC has a unique stationary distribution in equilibrium. By empirical evaluation, we observe that ModMCMC converges after $K|r.S|$ steps for $K = 50$ (see Section 6.3).

Each node $r$ in the tree is of type *NodeSet*. This structure consists of an array $r.part$ where $r.part[u] \in \{0..k-1\}$ is the group id of $u$. To make sure that ModMCMC runs in linear time, we must have a constant time computation of modularity $Q(P)$ (line 4 of Algorithm 3). This can be done with two helper arrays: the number of intra-edges $r.lc[0..k-1]$ and the total degree of nodes $r.dc[0..k-1]$ in each group. The modularity $Q$ is computed in $O(k)$ (Formula 1) using $r.lc, r.dc$. When moving node $u$ from group $i$ to group $j$, $r.lc$ and $r.dc$ are updated accordingly by checking the neighbors of $u$ in $G$. The average degree $\bar{d}$ is a constant, so the complexity of ModMCMC is linear in the number of MCMC steps.

**Algorithm 3** ModMCMC

**Input:** graph $G$, nodeset $r.S$, group size $k$, privacy budget $\epsilon_p$
**Output:** sampled partition $r.part$

```
1: initialize r.part with a random partition P_0 of k groups
2: for each step i in the Markov chain do
3:     pick a neighboring partition P' of P_{i−1} by randomly selecting node u ∈ r.S and moving u to another group.
4:     accept the transition and set P_i = P' with probability min(1, exp(ε_p/(2ΔQ) Q(P',G)) / exp(ε_p/(2ΔQ) Q(P_{i−1},G)))
5: // until equilibrium is reached
6: return a sampled partition r.part = P_i
```

***Finding the Best Cut*** Given the output k-ary tree $R$ with the root node $root$, our next step is to find the best cut

across the tree. A cut $C$ is a set of nodes in $R$ that cover all nodes in $V$. As an example, a cut $C$ in Fig.3 returns the clustering [{0,1,2,3}, {4}, {5,6,11}, {8,9,10,7,12}]. Any cut has a modularity score. Our goal is to find the best cut, i.e. the cut with highest modularity, in a private manner.

We solve this problem by a dynamic programming technique. Remember that modularity is an *additive* quantity (c.f. Formula 1). By denoting $opt(r)$ as the optimal modularity for the subtree rooted at node $r$, the optimal value is $opt(root)$. The recurrence relation is straightforward

$$opt(r) = \max\{Q(r), \sum_{t \in r.children} opt(t)\}$$

This idea is realized in three steps. The pseudo-code is provided in Appendix A.2. The first step uses a queue to fill a stack $S$. The stack ensures any internal node to be considered after its child nodes. The second step solves the recurrence relation. Because all modularity values are sensitive, we add Laplace noise Laplace($\Delta Q/\epsilon_m$). The global sensitivity $\Delta Q = 3/m$ (see Theorem 5.2), so we need only a small privacy budget for each level ($\epsilon_m = 0.01$ is enough in our experiments). The noisy modularity $mod_n$ is used to decide whether the optimal modularity at node $r$ is by itself or by the sum over its children. The final step backtracks the best cut from the root node.

### 5.1.3 Complexity

ModDivisive creates a k-ary tree of height $maxL$. At each node $r$ of the tree other than the leaf nodes, ModMCMC is run once. The run time of ModMCMC is $O(K \cdot |r.S|)$ thanks to the constant time for updating the modularity (line 4 of ModMCMC). Because the union of nodesets at one level is $V$, the total runtime is $O(K \cdot |V| \cdot maxL)$. BestCut only incurs a sublinear runtime because the size of tree is always much smaller than $|V|$. The following theorem states this result

THEOREM 5.1. *The time complexity of ModDivisive is linear in the number of nodes $n$, the maximum level $maxL$ and the burn-in factor $K$.*

### 5.1.4 Privacy Analysis

We show that ModMCMC satisfies differential privacy. The goal of MCMC is to draw a random sample from the desired distribution. Similarly, exponential mechanism is also a method to sample an output $x \in X$ from the target distribution with probability proportional to $\exp(\frac{\epsilon u(x)}{2\Delta u})$ where $u(x)$ is the score function ($x$ with higher score has bigger chance to be sampled) and $\Delta u$ is its sensitivity. The idea of using MCMC to realize exponential mechanism is first proposed in [3] and applied to $\epsilon$-DP graph release in [29].

In our ModMCMC, the modularity $Q(P, G)$ is used directly as the score function. We need to quantify the global sensitivity of $Q$. From Section 3.1, we have the following definition

DEFINITION 5.1. *(Global Sensitivity $\Delta Q$)*

$$\Delta Q = \max_{P,G,G'} |Q(P, G) - Q(P, G')| \quad (5)$$

We prove that $\Delta Q = O(1/m)$ in the following theorem whose proof is provided in Appendix A.3.

THEOREM 5.2. *The global sensitivity of modularity, $\Delta Q$, is smaller than $\frac{3}{m}$*

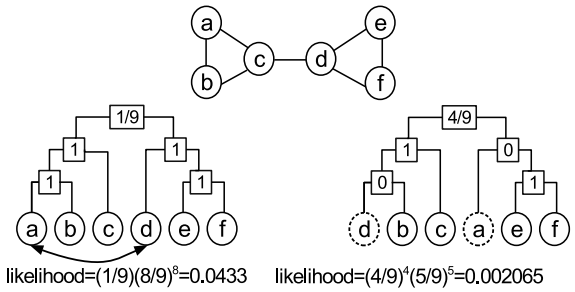



Figure 4: HRG-Fixed

## 5.2 A Variant of HRG-MCMC

Similar to DER and EdgeFlip, HRG-MCMC [29] runs in quadratic time due to its costly MCMC steps. Each MCMC step in HRG-MCMC takes $O(n)$ to update the tree. To make it runnable on million-scale graphs, we describe briefly a variant of HRG-MCMC called *HRG-Fixed* (see Fig.4). Instead of exploring the whole space of HRG trees as in HRG-MCMC, HRG-Fixed selects a fixed binary tree beforehand. We choose a balanced tree in our HRG-Fixed. Then HRG-Fixed realizes the exponential mechanism by sampling a permutation of $n$ leaf nodes (note that each leaf node represents a graph node). The next permutation is constructed from the current one by randomly choosing a pair of nodes and swap them. The bottom of Fig.4 illustrates the swap of two nodes $a$ and $d$. The log-likelihood $L$ still has the sensitivity $\Delta L = 2 \ln n$ as in HRG-MCMC [29]. Each sampling (MCMC) operation is designed to run in $O(\bar{d} \cdot \log n)$. By running $K \cdot n$ MCMC steps, the total runtime of HRG-Fixed is $O(K \cdot m \cdot \log n)$, feasible on large graphs. The burn-in factor $K$ of HRG-Fixed is set to 1,000 in our evaluation.

# 6. EXPERIMENTS AND RESULTS

In this section, our evaluation aims to compare the performance of the competitors by clustering quality and efficiency. The clustering quality is measured by the modularity $Q$ and the average $F_1$-score in which the modularity is the most important metric as we aim at highly modular clusterings. The efficiency is measured by the running time. All algorithms are implemented in Java and run on a desktop PC with *Intel*® Core i7-4770@ 3.4Ghz, 16GB memory.

Table 2: Characteristics of the test graphs

| | as20graph | ca-AstroPh | amazon | youtube |
|---|---|---|---|---|
| Nodes | 6,474 | 17,903 | 334,863 | 1,134,890 |
| Edges | 12,572 | 196,972 | 925,872 | 2,987,624 |
| Com | 30 | 37 | 257 | 13,485 |
| Mod | 0.623 | 0.624 | 0.926 | 0.710 |
| $0.5 \ln n$ | 4.4 | 4.9 | 6.4 | 7.0 |

Two medium-sized and two large real graphs are used in our experiments [1]. `as20graph` is the graph of routers comprising the internet. `ca-AstroPh` is a co-authorship network where two authors are connected if they publish at least one paper together. `amazon` is a product co-purchasing network where the graph contains an undirected edge from $i$ to $j$ if a product $i$ is frequently co-purchased with product

[1] http://snap.stanford.edu/data/index.html

$j$. `youtube` is a video-sharing web site that includes a social network. Table 2 shows the characteristics of the graphs. The rows Com(munities) and Mod(ularity) are the output of Louvain method. The row $0.5\ln n$ is the largest privacy budget $\epsilon$ considered in the experiments. The number of samples in each test case is 20.

The schemes are abbreviated as 1K-series (1K), EdgeFlip (EF), Top-m-Filter (TmF), DER, LouvainDP (LDP), ModDivisive (MD), HRG-MCMC and HRG-Fixed.

## 6.1 Quality Metrics

Apart from modularity $Q$, we use $\bar{F}_1$, the *average* $F_1$*-score*, following the benchmarks in [30]. The $F_1$ score of a set $A$ with respect to a set $B$ is defined as the harmonic mean $H$ of the precision and the recall of $A$ against $B$. We define $prec(A,B) = \frac{|A\cap B|}{|A|}$, $recall(A,B) = \frac{|A\cap B|}{|B|}$

$$F_1(A,B) = \frac{2.prec(A,B).recall(A,B)}{prec(A,B) + recall(A,B)}$$

Then the average $F_1$ score of two sets of communities $C$ and $C'$ is defined as

$$F_1(A,C) = \max_i F_1(A,c_i), \quad c_i \in C = \{c_1,..,c_n\}$$

$$\bar{F}_1(C,C') = \frac{1}{2|C|}\sum_{c_i\in C} F_1(c_i,C') + \frac{1}{2|C'|}\sum_{c_i\in C'} F_1(c_i,C)$$

We choose the output clustering of Louvain method as the ground truth for two reasons. First, the evaluation on the real ground truth is already done in [23] and Louvain method is proven to provide high quality communities. Second, the real ground truth is a set of *overlap* communities whereas the schemes in this paper output only *non-overlap* communities. The chosen values of $\epsilon$ are $\{0.1\ln n,\ 0.2\ln n,\ 0.3\ln n,\ 0.4\ln n,\ 0.5\ln n\}$.

Note that there is a well-known resolution limit of modularity [10, Section VI.C]. This phenomenon may occur in top-down community detection approaches in which the modularity of a clustering of large communities is not much different from that of a clustering of many smaller communities. Similarly, the F1-score may not be consistent with modularity. For example, on `amazon` graph, non-private Louvain method returns a clustering of 52,655 small communities with modularity 0.678 after first round. In the final round, it returns a clustering of 257 large communities with modularity 0.926. The F1-score between 52,655 small communities and 257 large communities (as ground truth) is 0.01. On `youtube` graph, the first round and the last round of non-private Louvain return 147,361 and 13,485 communities with modularity 0.644 and 0.710 respectively. The F1-score between the two clusterings is 0.14. These results confirm the inherent limitations of modularity and F1-score for community detection.

## 6.2 Performance of LouvainDP

We test LouvainDP for the group size $k \in \{4,8,16,32,64\}$. The results on `youtube` are displayed in Fig. 5. We observe a clear separation of two groups $k = 4,8$ and $k = 16,32,64$. As $k$ increases, the modularity increases faster but also saturates sooner. Similar separations appear in avg.F1Score and the number of communities. Non-trivial modularity scores in several settings of $(k,\epsilon)$ indicate that randomly grouping of nodes and high-pass filtering superedges do not destroy all community information of the original graph.

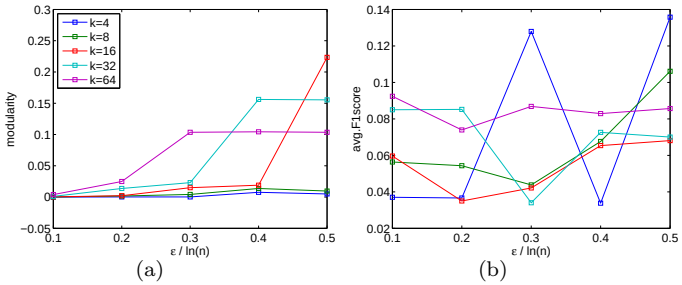



Figure 5: LouvainDP on `youtube`

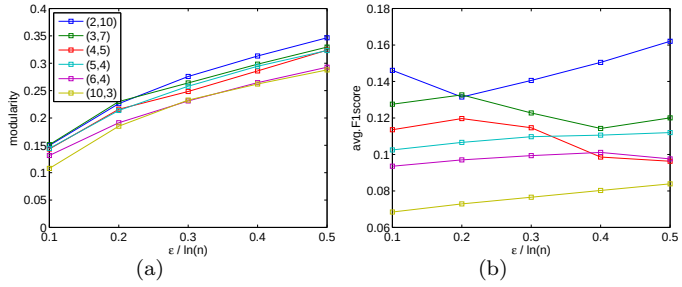



Figure 6: ModDivisive on `youtube` with $\lambda = 2.0$, $K = 50$

At $\epsilon = 0.5\ln n$ and $k = 4,8$, the total edge weight in $G_1$ is very low ($< 0.05m$), so many supernodes of $G_1$ are disconnected and Louvain method outputs a large number of communities. The reason is that the threshold $\theta$ in LouvainDP is an integral value, so causing abnormal leaps in the total edge weight of $G_1$. We pick $k = 8,64$ for the comparative evaluation (Section 6.4).

## 6.3 Performance of ModDivisive

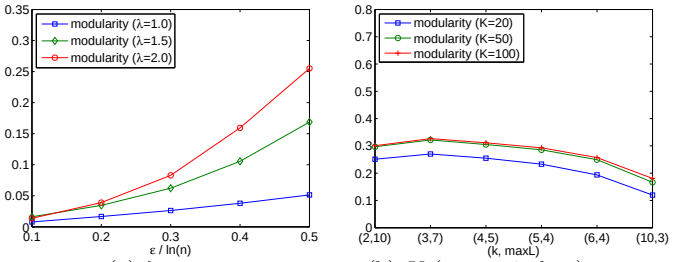
The effectiveness of ModDivisive is illustrated in Fig. 6 for graph `youtube` and $\lambda = 2.0$, $K = 50$. We select six pairs of $(k, maxL)$ by the set {(2,10),(3,7),(4,5), (5,4),(6,4),(10,3)}. Modularity increases steadily with $\epsilon$ while it is not always the case for avg.F1Score.

We choose $\lambda = 2.0$ to obtain a good allocation of $\epsilon$ among the levels. Fig. 7a shows the modularity for different values of $\lambda$. Note that $\lambda = 1.0$ means $\epsilon$ is equally allocated to the $maxL$ levels. By building a k-ary tree, we reduce considerably the size of the state space $\mathcal{P}$ for MCMC. As a result, we need only a small burn-in factor $K$. Looking at Fig. 7b, we see that larger $K = 100$ results in only tiny increase of modularity in comparison with that of $K = 50$.

## 6.4 Comparative Evaluation

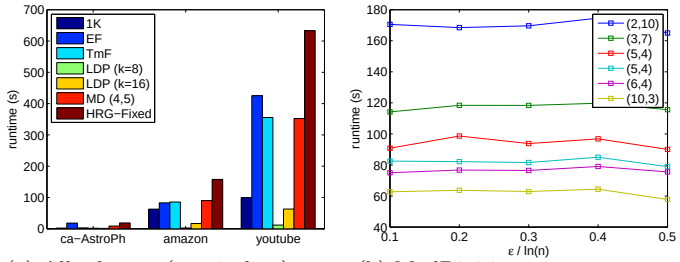
In this section, we report a comparative evaluation of LouvainDP and ModDivisive against the competitors in Fig. 9. The dashed lines in subfigures 9i, 9j, 9k and 9l represent the ground-truth number of communities by Louvain method. ModDivisive performs best in most of the cases.

On `as20graph` and `ca-AstroPh`, HRG-MCMC outputs the whole nodeset $V$ with the zero modularity while 1K-series, TmF, DER also give useless clusterings. EdgeFlip produces good quality metrics exclusively on `ca-AstroPh` while LouvainDP returns the highest modularity scores on `as20graph`. However, the inherent quadratic complexity of EdgeFlip




(a) $\lambda$ (b) $K$ (at $\epsilon = 0.5\ln n$)

Figure 7: ModDivisive: modularity vs. $\lambda$ and $K$ on `amazon`




(a) All schemes ($\epsilon = 0.5\ln n$) (b) ModDivisive on `amazon`

Figure 8: Runtime (in second)

makes Louvain method fail at $\epsilon = 0.1\ln n$ and $0.2\ln n$ for `ca-AstroPh` graph.

On the two large graphs, ModDivisive dominates the other schemes by a large margin in modularity and avg.F1Score. LouvainDP is the second best in modularity at $k = 64$. Our proposed HRG-Fixed is consistent with $\epsilon$ and has good performance on `youtube`. Note that HRG-MCMC is infeasible on the two large graphs due to its quadratic complexity. Again, 1K-series, TmF and EdgeFlipShrink provide the worst quality scores with the exception of 1K-series's avg.F1Score on `youtube`.

The runtime of the linear schemes is reported in Fig. 8. EdgeFlipShrink, 1K-series, TmF and LouvainDP benefit greatly by running Louvain method on the noisy output graph $\tilde{G}$. ModDivisive and HRG-Fixed also finish their work quickly in $O(K \cdot n \cdot maxL)$ and $O(K \cdot m \cdot \log n)$ respectively.

## 7. CONCLUSION

We have given a big picture of the problem $\epsilon$-DP community detection within the two categories: input and algorithm perturbation. We analyzed the major challenges of community detection under differential privacy. We explained why techniques borrowed from k-Means fail and how the difficulty of $\epsilon$-DP recommender systems enables a relaxation of privacy budget. We proposed LouvainDP and ModDivisive as the representatives of input and algorithm perturbations respectively. By conducting a comprehensive evaluation, we revealed the advantages of our methods. ModDivisive steadily gives the best modularity and avg.F1Score on large graphs while LouvainDP outperforms the remaining input perturbation competitors in certain settings. HRG-MCMC/HRG-Fixed give low modularity clusterings, indicating the limitation of the HRG model in divisive CD. The input perturbation schemes DER, EF, 1K-series and TmF hardly deliver any good node clustering except EF on the two medium-sized graphs.

For future work, we plan to develop an $\epsilon$-DP agglomerative scheme based on Louvain method and extend our work for directed graphs and overlapping community detection under differential privacy.

## 8. REFERENCES


[1] S. Banerjee, N. Hegde, and L. Massoulie. The price of privacy in untrusted recommender systems. *Selected Topics in Signal Processing, IEEE Journal of*, 9(7):1319–1331, 2015.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[3] K. Chaudhuri, A. D. Sarwate, and K. Sinha. A near-optimal algorithm for differentially-private principal components. *The Journal of Machine Learning Research*, 14(1):2905–2943, 2013.

[4] R. Chen, B. C. Fung, P. S. Yu, and B. C. Desai. Correlated network data publication via differential privacy. *VLDB Journal*, 23(4):653–676, 2014.

[5] A. Clauset, C. Moore, and M. E. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[6] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[7] G. Cormode, C. Procopiuc, D. Srivastava, and T. T. Tran. Differentially private summaries for sparse data. In *ICDT*, pages 299–311. ACM, 2012.

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *TCC*, pages 265–284, 2006.

[9] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[11] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

[12] R. Guerraoui, A.-M. Kermarrec, R. Patra, and M. Taziki. D2p: distance-based differential privacy in recommenders. *PVLDB*, 8(8):862–873, 2015.

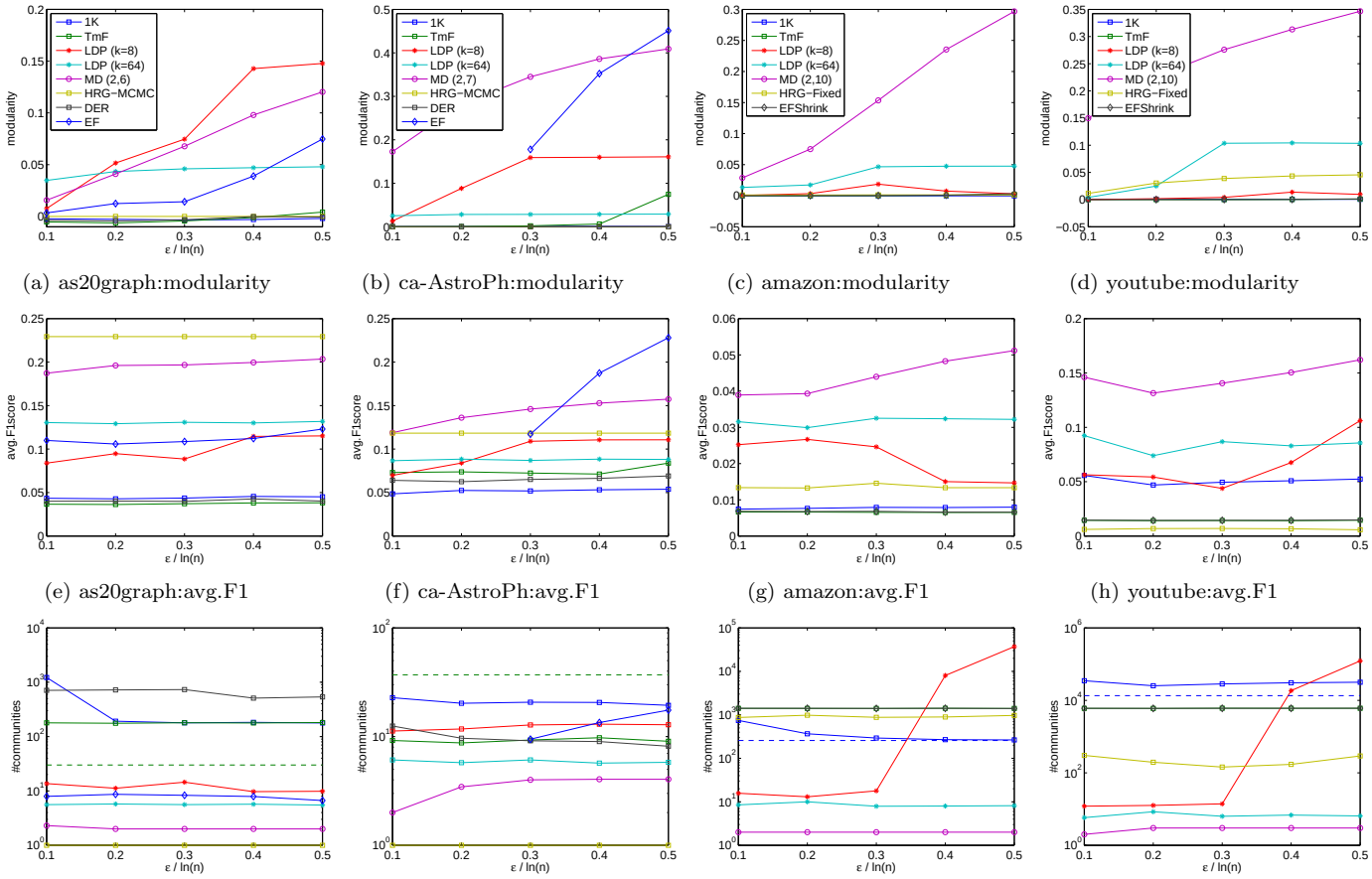
[13] A. Gupta, A. Roth, and J. Ullman. Iterative constructions and private data release. In *Theory of Cryptography*, pages 339–356. Springer, 2012.

[14] K. Kenthapadi, A. Korolova, I. Mironov, and N. Mishra. Privacy via the johnson-lindenstrauss transform. *arXiv preprint arXiv:1204.2606*, 2012.

[15] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103. IEEE, 2007.

[16] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 19–30. ACM, 2009.

[17] A. Medus, G. Acuna, and C. Dorso. Detection of community structures in networks via global optimization. *Physica A: Statistical Mechanics and its Applications*, 358(2):593–604, 2005.




(a) as20graph:modularity (b) ca-AstroPh:modularity (c) amazon:modularity (d) youtube:modularity

(e) as20graph:avg.F1 (f) ca-AstroPh:avg.F1 (g) amazon:avg.F1 (h) youtube:avg.F1

(i) as20graph:#communities (j) ca-AstroPh:#communities (k) amazon:#communities (l) youtube:#communities

Figure 9: Quality metrics and the number of communities on four graphs


[18] Y. Mülle, C. Clifton, and K. Böhm. Privacy-integrated graph clustering through differential privacy. In *PAIS 2015*, 2015.

[19] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[20] H. H. Nguyen, A. Imine, and M. Rusinowitch. Differentially private publication of social graphs at linear cost. In *ASONAM 2015*, 2015.

[21] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, pages 75–84. ACM, 2007.

[22] P. Pons and M. Latapy. Computing communities in large networks using random walks. In *Computer and Information Sciences-ISCIS 2005*, pages 284–293. Springer, 2005.

[23] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *WWW*, pages 225–236. ACM, 2014.

[24] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[25] J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1):016110, 2006.

[26] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[27] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin. Differentially private $k$-means clustering. *arXiv preprint arXiv:1504.05998*, 2015.

[28] Y. Wang and X. Wu. Preserving differential privacy in degree-correlation based graph generation. *TDP*, 6(2):127, 2013.

[29] Q. Xiao, R. Chen, and K.-L. Tan. Differentially private network data release via structural inference. In *KDD*, pages 911–920. ACM, 2014.

[30] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pages 745–754. IEEE, 2012.

[31] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *SIGMOD*, pages 731–745. ACM, 2015.

# APPENDIX

# A. ALGORITHMS AND PROOFS

## A.1 EdgeFlipShrink

Instead of outputting $\tilde{G}$, EdgeFlipShrink computes $\hat{G}$ that has the expected number of edges $m$ by shrinking $E_{\tilde{G}}$. First, the algorithm compute the private number of edges $\tilde{m}$ using a small budget $\epsilon_2$ (Lines 2-3 ). The new flipping probability $\tilde{s}$ is updated (Line 5). The noisy expected number of edges in the original EdgeFlip is shown in Line 6. We obtain the shrinking factor $p = \frac{\tilde{m}}{m_0}$ (Line 7). Using $p$, every 1-edge is sampled with probability $\frac{1-\tilde{s}}{2}p$ instead of $\frac{1-s}{2}$ as in [18]. The remaining 0-edges are randomly picked from $E_G$ as long as they do not exist in $\hat{G}$ (Lines 14-19).

The expected edges of $\hat{G}$ is $E[\hat{G}] = E[\tilde{m}] = m$ and the running time of EdgeFlipShrink is $O(m)$.

**Algorithm 4** EdgeFlipShrink($G, s$)

**Input:** undirected graph $G$, flipping probability $s$
**Output:** anonymized graph $\hat{G}$

1: $\hat{G} \leftarrow \varnothing$
2: $\epsilon_2 = 0.01$
3: $\tilde{m} = m + Lap(1/\epsilon_2)$
4: $\epsilon = \ln(\frac{2}{s} - 1) - \epsilon_2$
5: $\tilde{s} = \frac{2}{e^{\epsilon}+1}$
6: $m_0 = (1-\tilde{s})\tilde{m} + \frac{n(n-1)}{4}\tilde{s}$
7: $p = \frac{\tilde{m}}{m_0}$
8: // process 1-edges
9: $n_1 = 0$
10: **for** edge $(i,j) \in E_G$ **do**
11: add edge $(i,j)$ to $\hat{G}$ with prob. $\frac{1-\tilde{s}}{2}p$
12: $n_1 + +$
13: // process 0-edges
14: $n_0 = \tilde{m} - n_1$
15: **while** $n_0 > 0$ **do**
16: random pick an edge $(i,j) \notin E_G$
17: **if** $\hat{G}$ does not contain $(i,j)$ **then**
18: add edge $(i,j)$ to $\hat{G}$
19: $n_0$- -
20: **return** $\hat{G}'$

## A.2 BestCut Algorithm

Algorithm 5 outlines the steps of BestCut (Section 5.1.2).

## A.3 Proof of Theorem 5.2

PROOF. Given the graph $G$ and a partition $P$ of a nodeset $V_p \subseteq V$ (for any node of the k-ary tree other than the root node, its nodeset $V_p$ is a strict subset of $V$), the neighboring graph $G'$ has $E_{G'} = E_G \cup e$. We have two cases

*Case 1.* The new edge $e$ is an intra-edge within the community $s$. The modularity $Q(P,G)$ is $\sum_c^k(\frac{l_c}{m} - \frac{d_c^2}{4m^2})$. The modularity $Q(P,G')$ is $\sum_{c\neq s}^k(\frac{l_c}{m+1} - \frac{d_c^2}{4(m+1)^2}) + (\frac{l_s+1}{m+1} - \frac{(d_s+2)^2}{4(m+1)^2})$.

The difference $d_1 = Q(P,G') - Q(P,G) = \frac{1}{m+1} - \frac{1}{m(m+1)}\sum_c^k l_c + \frac{2m+1}{4m^2(m+1)^2}\sum_c^k d_c^2 - \frac{d_s+1}{(m+1)^2}$

Because $\Delta Q$ is the absolute value of $d_1$, we consider the most positive and the most negative values of $d_1$. Remember that $\sum_c^k d_c \leq 2m$, so the positive bound $d_1 < \frac{1}{m+1} + \frac{(2m+1)4m^2}{4m^2(m+1)^2} < \frac{3}{m+1}$. For the negative bound, we use the constraints $\sum_c^k l_c \leq m$ and $d_s \leq 2m$, so $d_1 > \frac{1}{m+1} - \frac{m}{m(m+1)} - \frac{2m+1}{(m+1)^2} > \frac{-2}{m+1}$. As a result, $\Delta Q = |d_1| < \frac{3}{m+1} < \frac{3}{m}$.

**Algorithm 5** BestCut

**Input:** undirected graph $G$, root node *root*, privacy budget at each level $\epsilon_m$
**Output:** best cut $C$

1: stack $S \leftarrow \varnothing$, queue $Q \leftarrow$ root
2: **while** $Q$ is not empty **do**
3: $r \leftarrow Q.dequeue()$
4: $S.push(r)$
5: **for** child node $r_i$ in $r.children$ **do**
6: $Q.enqueue(r_i)$
7: dictionary $sol \leftarrow \varnothing$
8: **while** $S$ is not empty **do**
9: $r \leftarrow S.pop()$, $r.mod_n = r.mod + \text{Laplace}(\Delta Q/\epsilon_m)$
10: **if** $r$ is a leaf node **then**
11: $sol.put(r.id, (\text{val} = r.mod_n, \text{self} = True))$
12: **else**
13: $s_m = \sum_{r_i \in r.children} sol[r_i.id].mod_n$
14: **if** $r.mod_n < s_m$ **then**
15: $sol.put(r.id, (\text{val} = s_m, \text{self} = False))$
16: **else**
17: $sol.put(r.id, (\text{val} = r.mod_n, \text{self} = True))$
18: list $C \leftarrow \varnothing$, queue $Q \leftarrow$ root
19: **while** $Q$ is not empty **do**
20: $r \leftarrow Q.dequeue()$
21: **if** $sol[r.id].\text{self} == True$ **then**
22: $C = C \cup \{r\}$
23: **else**
24: **for** child node $r_i$ in $r.children$ **do**
25: $Q.enqueue(r_i)$
**return** $C$

*Case 2.* The new edge $e$ is an inter-edge between the communities $s$ and $t$. Similarly, we have $Q(P,G) = \sum_c^k(\frac{l_c}{m} - \frac{d_c^2}{4m^2})$ while $Q(P,G') = \sum_{c\neq s,t}^k(\frac{l_c}{m+1} - \frac{d_c^2}{4(m+1)^2}) + (\frac{l_s}{m+1} - \frac{(d_s+1)^2}{4(m+1)^2}) + (\frac{l_t}{m+1} - \frac{(d_t+1)^2}{4(m+1)^2})$.

The difference $d_2 = Q(P,G') - Q(P,G) = -\frac{1}{m(m+1)}\sum_c^k l_c + \frac{2m+1}{4m^2(m+1)^2}\sum_c^k d_c^2 - \frac{2d_s+2d_t+2}{4(m+1)^2}$. Again, we consider the most positive and the most negative values of $d_2$, using the constraint $\sum_c^k d_c \leq 2m$, the positive bound $d_2 < \frac{(2m+1)4m^2}{4m^2(m+1)^2} < \frac{2}{m+1}$. For the negative bound, we use the constraints $\sum_c^k l_c \leq m$ and $d_s + d_t \leq 2m$, so $d_2 > -\frac{m}{m(m+1)} - \frac{4m+2}{4(m+1)^2} > \frac{-2}{m+1}$. As a result, $\Delta Q = |d_2| < \frac{2}{m+1} < \frac{2}{m}$.

To recap, in both cases $\Delta Q < \frac{3}{m}$. □