Differentially Private Frequent Subgraph Mining

Shengzhi Xu #1, Sen Su #2*, Li Xiong †3, Xiang Cheng #4, Ke Xiao #5

State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing, China
{1dear_shengzhi, 2susen, 4chengxiang, 5iejr}@bupt.edu.cn

† *Math and Computer Science Department, Emory University, Atlanta, GA*
3 lxiong@mathcs.emory.edu

***Abstract*—Mining frequent subgraphs from a collection of input graphs is an important topic in data mining research. However, if the input graphs contain sensitive information, releasing frequent subgraphs may pose considerable threats to individual's privacy. In this paper, we study the problem of frequent subgraph mining (FGM) under the rigorous differential privacy model. We introduce a novel differentially private FGM algorithm, which is referred to as *DFG*. In this algorithm, we first privately identify frequent subgraphs from input graphs, and then compute the noisy support of each identified frequent subgraph. In particular, to privately identify frequent subgraphs, we present a frequent subgraph identification approach which can improve the utility of frequent subgraph identifications through candidates pruning. Moreover, to compute the noisy support of each identified frequent subgraph, we devise a lattice-based noisy support derivation approach, where a series of methods has been proposed to improve the accuracy of the noisy supports. Through formal privacy analysis, we prove that our *DFG* algorithm satisfies ϵ-differential privacy. Extensive experimental results on real datasets show that the *DFG* algorithm can privately find frequent subgraphs with high data utility.**

I. Introduction

Frequent subgraph mining (FGM) is a fundamental topic in data mining research. Given a collection of input graphs, FGM aims to find all subgraphs that occur in input graphs more frequently than a given threshold. FGM has practical importance in a number of applications, ranging from bioinformatics to social network analysis. For instance, discovering frequent subgraphs in social networks can be vital to understand the mechanics of social interactions. Despite valuable information the discovery of frequent subgraphs could gain, if the data is sensitive (e.g., mobile phone call graphs, trajectory graphs and web-click graphs), releasing frequent subgraphs unfortunately raises increasing concerns on individual's privacy.

Differential privacy [1] has been proposed as a way to address this problem. Unlike the anonymization-based privacy models (e.g., k-anonymity [2], l-diversity [3]), differential privacy offers strong privacy guarantees and robustness against adversaries with prior knowledge. In general, by adding a carefully chosen amount of perturbation noise, differential privacy assures that the output of a computation is insensitive to the change of any individual's record, and thus restricting privacy breach through the results.

Corresponding author of this paper is Prof. Sen Su

In this paper, we study the problem of frequent subgraph mining under ϵ-differential privacy. Existing work on differentially private FGM [4] employs the Markov Chain Monte Carlo (MCMC) sampling to extend the exponential mechanism [5]. It directly selects frequent subgraphs from all possible graphs which may or may not exist in the input graphs. However, since verifying the convergence of MCMC remains an open problem when the distribution of samples is not observable, the method proposed in [4] only guarantees the weaker (ϵ, δ)-differential privacy. Moreover, as the output space contains all possible graphs, it results in a very large output space from which this method has to select frequent subgraphs, making the selection inaccurate. We also notice that a growing number of studies have recently been proposed for finding frequent itemsets and frequent sequences under differential privacy [6], [7], [8]. However, due to the inherent complex structure of graph data, these studies cannot be applied to discover frequent subgraphs. To our best knowledge, we are not aware of any existing studies which can find frequent subgraphs with high data utility while satisfying ϵ-differential privacy.

To make FGM satisfy ϵ-differential privacy, a potential approach is to utilize the Laplace mechanism [9] to find frequent subgraphs in order of increasing size. In particular, for the mining of frequent i-subgraphs (i.e., the frequent subgraphs containing i edges), we can first generate all possible frequent i-subgraphs (i.e., candidate i-subgraphs) based on the downward closure property [10], and then perturb the support of each candidate i-subgraph. The candidate i-subgraphs with noisy supports larger than the threshold are output as frequent i-subgraphs. Although this approach can identify the frequent subgraphs and obtain their noisy supports simultaneously, it suffers poor performance. The main reason is that the amount of noise added in this approach is proportionate to the number of candidate subgraphs. During the mining process, a large number of candidate subgraphs are generated, which leads to a large amount of perturbation noise.

Another potential approach to make FGM achieve ϵ-differential privacy is to utilize the exponential mechanism [5]. In particular, given the candidate i-subgraphs generated based on the downward closure property, we can first use the exponential mechanism to privately select subgraphs from the candidate i-subgraphs, and then only compute the noisy support of the selected subgraphs. In this way, the amount of added noise is proportionate to the number of selected subgraphs. However, it is hard to know how many subgraphs this approach should select from the candidate i-subgraphs (i.e., the number of frequent i-subgraphs). Moreover, due to the large number of generated candidate i-subgraphs, this approach has to select subgraphs from a large candidate set, which makes the selections inaccurate.

To address the above problems, we introduce a novel ***d***ifferentially private ***f***requent sub***g***raph mining algorithm, called *DFG*. In particular, our *DFG* algorithm consists of two main steps. In the first step, we present a frequent subgraph identification approach to privately identify frequent subgraphs from input graphs. In the second step, we devise a lattice-based noisy support derivation approach to compute the noisy support of each identified frequent subgraph.

Specifically, in the first step of our *DFG* algorithm, we present a frequent subgraph identification approach to privately identify frequent subgraphs in order of increasing size. In this approach, given the candidate i-subgraphs, we first propose a binary estimation method to estimate the number of frequent i-subgraphs, and then utilize a conditional exponential method to privately select frequent i-subgraphs from the candidate i-subgraphs. In particular, in our binary estimation method, we leverage the idea of binary search to estimate the number of frequent i-subgraphs. The amount of noise required in this method is only logarithmic to the number of candidate i-subgraphs. Moreover, in practice, we found the number of real frequent subgraphs is much smaller than the number of generated candidate subgraphs. If we could effectively prune the unpromising candidate subgraphs, the candidate set can be considerably reduced and the probabilities of selecting real frequent subgraphs can be significantly improved. Based on this observation, we propose the conditional exponential method, which utilizes the noisy support of candidate subgraphs to prune the obviously infrequent candidate subgraphs, and privately selects subgraphs from the remaining candidate subgraphs. In doing so, the accuracy of the private selections can be substantially improved. In our conditional exponential method, we prove that, rather than being proportionate to the number of candidate i-subgraphs, the amount of noise added to the support of each candidate i-subgraph is proportionate to the number of selected subgraphs. Notice that, in our frequent subgraph identification approach, only the selected subgraphs are output. The noisy support of subgraphs is not released.

After privately identifying the frequent subgraphs, in the second step of our *DFG* algorithm, we devise a lattice-based noisy support derivation approach to compute the noisy support of each identified frequent subgraph. In particular, we first build a directed lattice, where each node represents a frequent subgraph. Then, for the graphs on a given path of the lattice, by leveraging the inclusion relation between subgraphs [10] and the parallel composition property of differential privacy [9], we propose a count accumulation method which accumulates the number of graphs in disjoint databases to obtain the noisy support of the graphs. We show that, compared with directly perturbing the support of the graphs on a path, our count accumulation method can significantly improve the accuracy of the noisy supports. Next, we can construct multiple paths based on the lattice to cover all the frequent subgraphs, and use the count accumulation method on the selected paths to obtain the noisy support of each frequent subgraph. We found the amount of added noise is mainly affected by the number of constructed paths. To reduce the perturbation noise, we propose a path construction method which constructs as few paths as possible to cover all the frequent subgraphs. At last, to further improve the accuracy of the noisy supports, we propose a path extension method to optimize the constructed paths.

Through formal privacy analysis, we prove that our *DFG* algorithm is ϵ-differentially private. Extensive performance study illustrates that our *DFG* algorithm achieves high data utility. In addition, to demonstrate the generality of our frequent subgraph identification approach (including binary estimation and conditional exponential methods) and further enrich the application spectrum, we apply it to frequent itemset mining. The experimental results show that we can achieve better performance than the state-of-the-art on differentially private frequent itemset mining [11]. The key contributions of this paper are summarized as follows.

1). We introduce a novel differentially private frequent subgraph mining algorithm, called *DFG*. To our best knowledge, it is the first algorithm which can find frequent subgraphs from a collection of input graphs with high data utility while satisfying ϵ-differential privacy.

2). We present a frequent subgraph identification approach (including binary estimation and conditional exponential methods) to privately identify frequent subgraphs from the input graphs. This approach is not only suitable for mining frequent subgraphs, but also can be utilized for mining other kinds of frequent patterns under differential privacy.

3). We devise a lattice-based noisy support derivation approach to compute the noisy support of each identified frequent subgraph, in which a series of methods is proposed to improve the accuracy of the noisy supports.

4). Through formal privacy analysis, we prove that our *DFG* algorithm guarantees ϵ-differential privacy. The extensive experiments illustrate that our *DFG* algorithm can privately find frequent subgraphs with high data utility.

II. Related Work

We broadly categorize existing differentially private frequent pattern mining studies into three groups based on the type of pattern being mined.

Graph Mining. The work most related to ours is by Shen et al. [4]. They propose a differentially private algorithm to discover frequent subgraphs from a set of input graphs. They use the Markov Chain Monte Carlo (MCMC) sampling to bypass the unknown output space when applying the exponential mechanism. However, their algorithm only achieves (ϵ, δ)-differential privacy, which is a relaxed version of ϵ-differential privacy. In contrast, we formally prove our *DFG* algorithm satisfies the standard ϵ-differential privacy. In addition, their algorithm directly selects frequent subgraphs from the space which contains the subgraphs of all possible sizes. Different from their algorithm, we identify frequent subgraphs in order of increasing size. During the frequent subgraph identification process, we utilize our conditional exponential method and the downward closure property [10] to reduce the candidate set.

Besides finding frequent subgraphs from a set of input graphs, another variant of the frequent subgraph mining problem is to find frequent subgraphs in different regions of a single graph. In this variant, it needs to count the number of occurrences of subgraphs in a single graph. Recently, several studies have been proposed to address the issue of subgraph counting in a single graph under differential privacy. Karwa et al. [12] propose algorithms to privately count the occurrences of two families of subgraphs (k-stars and k-triangles). Chen et al. [13] present a new recursive mechanism which can privately release the occurrences of multiple kinds of subgraphs. Proserpio et al. [14] develop a private data analysis platform wPINQ over weighted datasets, which can be used to answer subgraph-counting queries. Very recently, Zhang et al. [15] propose a ladder framework to privately count the number of occurrences of subgraphs. Different from the above studies which answer subgraph-counting queries in a single graph, our work aims to find frequent subgraphs from a collection of input graphs.

Itemset Mining. A number of studies have been proposed to address the frequent itemset mining (FIM) problem under differential privacy. Bhaskar et al. [16] utilize the exponential mechanism [5] and Laplace mechanism [9] to develop two differentially private FIM algorithms. In [6], to meet the challenge of high dimensionality in transaction databases, Li et al. introduce an algorithm which projects the high-dimensional database onto lower dimensions. Zeng et al. [7] find the utility and privacy tradeoff in differentially private FIM can be improved by limiting the length of transactions. They propose a transaction truncating method to limit the length of transactions. Different from [7], Cheng et al. [17] propose a transaction splitting method to limit the length of transactions. In [11], based on the FP-growth algorithm [18], Su et al. present an efficient algorithm, PFP-growth, for mining frequent itemsets under differential privacy. All these differentially private FIM algorithms [16], [6], [7], [17], [11], [19] are shown to be effective for some scenarios. However, the differences between the itemset and the graph prevent us from applying these algorithms to differentially private FGM.

Recently, Lee et al. [20] introduce a differentially private top-k FIM algorithm. They propose a generalized sparse vector technique to identify frequent itemsets, which perturbs the threshold and the support of each candidate itemset. However, Machanavajjhala et al. [21] found such technique does not satisfy differential privacy. The main reason is that this technique ignores the consistency of the noisy threshold in each comparison between the noisy support and the noisy threshold.

Sequence Mining. For differentially private frequent sequence mining, Bonomi et al. [22] propose a two-phase differentially private algorithm for mining prefixes and substrings. Xu et al. [8] utilize a sampling-based candidate pruning technique to discover frequent subsequences under differential privacy. In [23], Cheng et al. introduce a differentially private algorithm for finding maximal frequent sequences.

III. Preliminaries

A. Frequent Subgraph Mining

A graph G=(V, E) consists of a set of vertices V and a set of edges E. Each vertex is associated with a label, which is drawn from a set of vertex labels. The label of each vertex is not required to be unique and multiple vertices can have the same label. The size of a graph is defined to be the number of edges in this graph. A graph is called an i-graph if its size is i. In this paper, we consider that each edge in a graph is undirected and not associated with a label. However, our solution can be easily extended to the case of graphs with directed and labeled edges.

Suppose there are two graphs G_1=(V_1, E_1) and G_2=(V_2, E_2). Let $L(u)$ denote the label of vertex u. We say that G_1 is contained in G_2 if there exists a function f: $V_1 \rightarrow V_2$, such that, $\forall$ (u, v) $\in E_1$, we have ($f(u)$, $f(v)$) $\in E_2$, $L(u)$ = $L(f(u))$ and $L(v)$ = $L(f(v))$. If G_1 is contained in G_2, we say that G_1 is a *subgraph* of G_2, and G_2 is a *super-graph* of G_1, denoted by $G_1 \subseteq G_2$.

A graph database is a multiset of input graphs, where each input graph represents an individual's record. The *support* of a graph is the number of input graphs containing it. Given a threshold, a graph is called *frequent* if its support is no less than this threshold. The problem of frequent subgraph mining (FGM) is formalized as follows.

Definition 1 (*Frequent Subgraph Mining*). *Given a graph database and a threshold, FGM aims to find all frequent subgraphs for the threshold and also compute the support of each frequent subgraph.*

B. Differential Privacy

Differential privacy [1] has emerged as the de-facto standard notion in private data analysis. In general, it requires an algorithm to be insensitive to the changes in any individual's record. In the context of FGM, two graph databases D_1, D_2 are considered as *neighboring databases* iff they differ by at most one input graph (by adding or removing an individual's record). Formally, differential privacy is defined as follows.

Definition 2 (*ϵ-differential privacy*). *A private algorithm $\mathcal{A}$ gives ϵ-differential privacy iff for any neighboring databases D_1 and D_2, and for any possible output $S \in Range(\mathcal{A})$,*

$$\Pr[\mathcal{A}(D_1) = S] \leq e^{\epsilon} \times \Pr[\mathcal{A}(D_2) = S].$$

A fundamental concept for achieving differential privacy is *sensitivity* [9]. It is used to measure the largest difference in the outputs over any two neighboring databases.

Definition 3 (*Sensitivity*). *For any function $f : D \rightarrow \mathbb{R}^n$, and any neighboring databases D_1 and D_2, the sensitivity of f is:*

$$\Delta f = \max_{D_1, D_2} ||f(D_1) - f(D_2)||.$$

Laplace mechanism [9] is a widely-adopted approach for designing algorithms to achieve differential privacy. It adds random noise drawn from the Laplace distribution to the true outputs of a function. The Laplace distribution with magnitude λ, i.e., $Lap(\lambda)$, follows the probability density function as $\Pr[x|\lambda] = \frac{1}{2\lambda} e^{-|x|/\lambda}$, where $\lambda = \frac{\Delta f}{\epsilon}$ is determined by the sensitivity Δf and the privacy budget ϵ.

Theorem 1 *For any function $f : D \rightarrow \mathbb{R}^n$ with sensitivity Δf, the algorithm $\mathcal{A}$*

$$\mathcal{A}(D) = f(D) + Lap(\Delta f/\epsilon)$$

achieves ϵ-differential privacy.

Another widely used mechanism for designing algorithms to achieve differential privacy is exponential mechanism [5]. Given the whole output space, the exponential mechanism assigns each possible output a utility score, and draw a sample from the output space based on the assigned utility scores.

Theorem 2 *For a database D, output space $\mathcal{R}$ and a utility score function $u : D \times \mathcal{R} \rightarrow \mathbb{R}$, the algorithm $\mathcal{A}$*

$$\Pr[\mathcal{A}(D) = r] \propto \exp(\frac{\epsilon \times u(D, r)}{2\Delta u})$$

satisfies ϵ-differential privacy, where Δu is the sensitivity of the utility score function.

For a sequence of differentially private algorithms, the composability properties [9] guarantee the overall privacy.

Theorem 3 (*Sequential Composition*). *Let $\mathcal{A}_1, ..., \mathcal{A}_t$ be t algorithms, and each algorithm satisfies ϵ_i-differential privacy. A sequence of algorithms $\mathcal{A}_i(D)$ over database D provides $(\sum \epsilon_i)$-differential privacy.*

Theorem 4 (*Parallel Composition*). *Let $\mathcal{A}_1, ..., \mathcal{A}_t$ be t algorithms, each satisfies ϵ_i-differential privacy. A sequence of algorithms $\mathcal{A}_i(D_i)$ over disjoint databases D_1, ..., D_t provides $\max(\epsilon_i)$-differential privacy.*

IV. A Straightforward Approach

In this section, we introduce a straightforward approach to make FGM satisfy ϵ-differential privacy. The main idea is to utilize the Laplace mechanism to perturb the support of all possible frequent subgraphs, and compare their noisy supports with the given threshold to determine which graphs are frequent. In particular, we follow the level-wise algorithm Apriori [10] to discover frequent subgraphs. For the mining of frequent i-subgraphs, based on the candidate generation method in [24], we first use the frequent (i-1)-subgraphs to generate all possible frequent i-subgraphs. The generated i-subgraphs are called candidate i-subgraphs. Then, we perturb the support of these candidate i-subgraphs and compare their noisy supports to the threshold. We consider the candidate i-subgraphs whose noisy supports exceed the threshold as frequent i-subgraphs, and output these graphs together with their noisy supports. The above process continues until no candidate subgraphs can be generated for a certain graph size.

Privacy Analysis. We now give the privacy analysis of the above approach. In this approach, given the candidate i-subgraphs, we perturb the support of each candidate i-subgraph. Let $S_i = \{s_{i_1}, s_{i_2}, ..., s_{i_n}\}$ denote the support of all candidate i-subgraphs. The amount of noise added in S_i is determined by the allocated privacy budget and the sensitivity of computing S_i. In particular, suppose the maximal size of frequent subgraphs is M_g. We uniformly assign the support computations of S_i a privacy budget $\frac{\epsilon}{M_g}$. Moreover, since a single input graph can affect the support of each candidate i-subgraph by at most one, the sensitivity Δ_i of computing S_i is the number of candidate i-subgraphs. Thus, adding Laplace noise $Lap(\frac{M_g \times \Delta_i}{\epsilon})$ in S_i achieves $\frac{\epsilon}{M_g}$-differential privacy. Overall, the mining process can be considered as a series of support computations. Based on the sequential composition property [9], this approach satisfies ϵ-differential privacy.

Limitation. The above approach, however, produces poor results. In this approach, the amount of perturbation noise is proportionate to the number of generated candidate subgraphs. During the mining process, a large number of candidate subgraphs are generated, which causes a large amount of noise to be added to the support of each candidate subgraph. As a result, the utility of the results is drastically reduced.

V. Overview of Our *DFG* Solution

To discover frequent subgraphs under ϵ-differential privacy, a potential approach is to leverage the exponential mechanism. Specifically, for mining frequent i-subgraphs, we can first utilize the exponential mechanism to privately select subgraphs from the candidate i-subgraphs, and then compute the noisy support of each selected subgraph. In this way, the amount of perturbation noise is just proportionate to the number of selected subgraphs. However, it is hard to know how many subgraphs we need to select from the candidate i-subgraphs (i.e., the number of frequent i-subgraphs). Moreover, in the mining process, a large number of candidate subgraphs are generated. It causes a large candidate set from which this approach has to select, making the selections inaccurate.

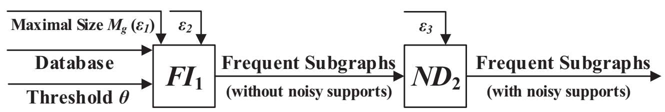


Fig. 1. The Overview of Algorithm *DFG*

Therefore, to privately find frequent subgraphs while providing a high level of data utility, we develop a novel algorithm, namely, *DFG* (i.e., ***d***ifferentially private ***f***requent sub***g***raph mining). An overview of this algorithm is shown in Fig. 1. In particular, we first privately estimate the maximal size of frequent subgraphs. Then, we present a frequent subgraph identification approach (referred to as FI_1) to privately identify frequent subgraphs in order of increasing size. In this approach, a binary estimation method is proposed to estimate the number of frequent subgraphs for a certain size, and a conditional exponential method is proposed to improve the accuracy of frequent subgraph identifications. After identifying frequent subgraphs, we devise a lattice-based noisy support derivation approach (referred to as ND_2) to compute the noisy support of identified frequent subgraphs. In this approach, a series of methods (i.e., count accumulation, path construction and path extension) is proposed to improve the accuracy of the noisy supports. For the privacy budget ϵ, we divide it into three parts: ϵ_1, ϵ_2 and ϵ_3. Specifically, ϵ_1 is used to estimate the maximal size of frequent subgraphs, ϵ_2 is used in FI_1 approach, and ϵ_3 is used in ND_2 approach.

In the following two sections, we introduce our FI_1 and ND_2 approaches, respectively. In Sec. VIII, we describe our *DFG* algorithm and prove that it satisfies ϵ-differential privacy.

VI. Frequent Subgraph Identification

In this section, we introduce our frequent subgraph identification (FI_1) approach, which identifies frequent subgraphs in order of increasing size. In particular, given the candidate i-subgraphs, based on the idea of binary search, we propose a binary estimation method to estimate the number of frequent i-subgraphs (See Sec. VI-A). Then, we privately select frequent i-subgraphs from the candidate i-subgraphs. To improve the accuracy of the private selections, we propose a conditional exponential method, which uses the noisy support of the candidate subgraphs to prune those obviously infrequent candidate subgraphs (See Sec. VI-B).

Algorithm 1 Frequent Subgraph Identification

Input:
Graph Database D; Threshold θ; Privacy Budgets ϵ_2;
Maximal size of frequent subgraph M_g;
Output:
Frequent Subgraphs F;
1: $F \leftarrow \emptyset$; $\epsilon_b = \frac{\alpha\epsilon_2}{M_g}$; $\epsilon_c = \frac{(1-\alpha)\epsilon_2}{M_g}$;
2: **for** i from 1 to M_g **do**
3: $C_i \leftarrow$ generate the set of candidate i-subgraphs;
4: n_i = *binary_estimation* (C_i, ϵ_b, θ); \\ see Sec. VI-A
5: F_i = *conditional_exponential* (C_i, ϵ_c, θ, n_i); \\ see Sec. VI-B
6: F += F_i;
7: **end for**
8: *return* F;

Alg. 1 shows the main steps of our FI_1 approach. Specifically, for the identification of frequent i-subgraphs, we first generate the candidate i-subgraphs (line 3). If i=1, we enumerate all possible single-edge graphs as candidate 1-subgraphs. The number of enumerated candidate 1-subgraphs is O($|V|^2$), where $|V|$ is the number of vertex labels. If i>1, based on the candidate generation method in [24], we use frequent (i-1)-subgraphs to generate candidate i-subgraphs. Then, we use our binary estimation method to estimate n_i, the number of frequent i-subgraphs (line 4). After that, by leveraging our conditional exponential method, we privately select subgraphs from the candidate i-subgraphs, and output them as frequent i-subgraphs (line 5).

A. Binary Estimation Method

Given the candidate i-subgraphs, we need to estimate the number of frequent i-subgraphs (i.e., the number of candidate i-subgraphs with supports larger than the given threshold). A simple method is to perturb the support of each candidate subgraph, and count the number of candidate subgraphs with noisy supports larger than the threshold. However, as shown in Sec. IV, this method will cause a large perturbation noise.

To this end, we put forward a binary estimation method, which leverages the idea of binary search to reduce the amount of added noise. The details are shown in Alg. 2. In particular, let $|C_i|$ be the number of candidate i-subgraphs and m = $\lfloor(|C_i|$-1)/2$\rfloor$. We first obtain the noisy support of the candidate i-subgraph with the m-th largest support (line 5). If this noisy support is larger than the threshold, it means the candidate i-subgraphs in the upper half are all above the threshold, and we only need to consider the candidate i-subgraphs in the lower half in the next iteration. Similarly, if this noisy support is smaller than the threshold, we only need to further search the candidate i-subgraphs in the upper half. This process continues until the number of candidate i-subgraphs with supports larger than the threshold is determined.

Privacy Analysis of Binary Estimation Method.

Theorem 5 *The binary estimation method (i.e., Algorithm 2) satisfies ϵ_b-differential privacy.*

Proof: In our binary estimation method, we first obtain the noisy support of the candidate i-subgraph with the m-th largest support, where m = $\lfloor(|C_i|$-1)/2$\rfloor$ and $|C_i|$ is the number of candidate i-subgraphs. Since adding (removing) an input graph can affect the result of such computation by at most one, the sensitivity of such computation is one. Thus, adding Laplace noise *Lap*($\lceil\log_2|C_i|\rceil/\epsilon_b$) in this computation satisfies ($\epsilon_b/\lceil\log_2|C_i|\rceil$)-differential privacy.

In this method, like the binary search, we only need to obtain the support of at most $\lceil\log_2|C_i|\rceil$ candidate i-subgraphs. By the sequential composition property [9], we can see our binary estimation method satisfies ϵ_b-differential privacy. □

Algorithm 2 Binary Estimation

Input:
Candidate i-subgraphs C_i; Privacy Budget ϵ_b; Threshold θ;
Output:
The number of frequent i-subgraphs n_i;
1: *low* $\leftarrow$ 0; *high* $\leftarrow$ $|C_i|$ - 1;
2: **while** *low* $\leq$ *high* **do**
3: *m* $\leftarrow$ $\lfloor$(*low* + *high*) / 2$\rfloor$;
4: $s_m \leftarrow$ get support of the candidate i-subgraph with the m-th largest support;
5: $ns_m \leftarrow s_m$ + *Lap*($\lceil\log_2|C_i|\rceil$ / ϵ_b);
6: **if** $ns_m == \theta$ **then**
7: *return* $|C_i|$ - m;
8: **else if** $ns_m > \theta$ **then**
9: *high* = *m* - 1;
10: **else if** $ns_m < \theta$ **then**
11: *low* = *m* + 1;
12: **end if**
13: **end while**
14: *return* $|C_i|$ - 1 - *high*;

B. Conditional Exponential Method

After obtaining the number of frequent i-subgraphs, we can privately select frequent i-subgraphs from candidate i-subgraphs. A simple method is to directly use exponential mechanism to choose graphs from candidate subgraphs. However, a large number of candidate subgraphs are generated in the mining process. It causes a large candidate set from which we have to select graphs, making the selection inaccurate.

To this end, we put forward a conditional exponential method. The main idea of this method is to use the noisy support of candidate subgraphs to prune those obviously infrequent candidate subgraphs, such that the candidate set can be considerably reduced and the probabilities of selecting real frequent subgraphs can be significantly improved. For the privacy budget ϵ_c, we divide it into two parts ϵ_{c1} and ϵ_{c2}, which are used to perturb the support of candidate subgraphs and privately select graphs, respectively.

The conditional exponential method is shown in Alg. 3. In particular, given the candidate i-subgraphs, we first add Laplace noise *Lap*($\frac{2n_i}{\epsilon_{c1}}$) to the true support of each candidate i-subgraph (line 5). Then, we prune those candidate i-subgraphs with noisy supports smaller than the threshold. Next, for each remaining candidate i-subgraph, we use its true support as its utility score. Based on the assigned scores, we privately select a subgraph from the remaining candidate i-subgraphs, and output it as a frequent i-subgraph (line 10). The above steps iterate until we choose n_i subgraphs from the candidate i-

subgraphs. Notice that, in our conditional exponential method, rather than being proportionate to the number of candidate i-subgraphs, the amount of noise added to the support of each candidate i-subgraph is proportionate to the number of selected subgraphs. Moreover, as this method only involves simple operations (e.g., addition and multiplication), it does not incur much overhead. Furthermore, in this method, we only output the selected subgraphs as frequent i-subgraphs, and do not release the noisy support of candidate subgraphs.

Algorithm 3 Conditional Exponential Method

```
Input:
    Candidate i-subgraphs C_i; Privacy Budget ε_c; Threshold θ;
    The number of frequent i-subgraphs n_i;
Output:
    Frequent i-subgraphs F_i;
ε_c1 ← βε_c, ε_c2 ← (1-β)ε_c;
for j from 1 to n_i do
    Set set ← ∅;
    for each subgraph g in C_i do
        ns_g = s_g + Lap(2n_i / ε_c1);
        if ns_g ≥ θ then
            Add g into set;
        end if
    end for
    g_j ← select a subgraph from set without replacement such that Pr[Selecting
    subgraph g] ∝ exp(ε_c2 × s_g / 2n_i), where s_g is the true support of subgraph g;
    F_i += g_j;
end for
return F_i;
```

Privacy Analysis of Conditional Exponential Method.

Theorem 6 *The conditional exponential method (i.e., Algorithm 3) satisfies ϵ_c-differential privacy.*

Proof: In this method, we first use the noisy support of candidate subgraphs to prune the obviously infrequent candidate subgraphs. Then, we select a subgraph from the remaining candidate subgraphs. Overall, the utility score we assigned to each candidate subgraph can be considered as

$$u(g,D)=\begin{cases}0 & ns_g(D)<\theta\\ \exp(\frac{\epsilon_{c2}\times s_g(D)}{2n_i}) & ns_g(D)\geq\theta\end{cases},$$

where $s_g(D)$ and $ns_g(D)$ are the true support and noisy support of subgraph g in database D, respectively.

Suppose D_1 and D_2 are two neighboring databases. For each candidate subgraph g, we have $-1\leq s_g(D_2)-s_g(D_1)\leq 1$. Let $f(g,\ D)$ = $\exp(\frac{\epsilon_{c2}\times s_g(D)}{2n_i})$. Then, we can prove that

$$\exp(-\frac{\epsilon_{c2}}{2n_i})\leq\frac{f(g,D_1)}{f(g,D_2)}\leq\exp(\frac{\epsilon_{c2}}{2n_i}). \quad (1)$$

In this method, we add Laplace noise $Lap(\frac{2n_i}{\epsilon_{c1}})$ to the support of each candidate subgraph. Let *noise* denote the amount of added noise. Then, based on the definition of Laplace mechanism, we have

$$\frac{\Pr[noise=X]}{\Pr[noise=X+1]}=\frac{\exp(-\frac{\epsilon_{c1}|X|}{2n_i})}{\exp(-\frac{\epsilon_{c1}|X+1|}{2n_i})}$$
$$=\exp(-\frac{\epsilon_{c1}}{2n_i}(|X|-|X+1|))\leq\exp(\frac{\epsilon_{c1}}{2n_i}). \quad (2)$$

Similarly, we also have

$$\frac{\Pr[noise=X+1]}{\Pr[noise=X]}\leq\exp(\frac{\epsilon_{c1}}{2n_i}). \quad (3)$$

Based on equations (2) and (3), given any subgraph g, for its noisy support $ns_g(D_1)$ = $s_g(D_1)$+*noise* in D_1 and its noisy support $ns_g(D_2)$ = $s_g(D_2)$+*noise* in D_2, we can prove

$$\Pr[s_g(D_1)+noise\geq\theta]$$
$$=\Pr[noise\geq\theta-s_g(D_1)]=\int_{\theta-s_g(D_1)}^{\infty}\Pr[noise=x]\ dx$$
$$\leq\int_{\theta-s_g(D_2)-1}^{\infty}\Pr[noise=x]\ dx=\int_{\theta-s_g(D_2)}^{\infty}\Pr[noise=x-1]\ dx$$
$$\leq\int_{\theta-s_g(D_2)}^{\infty}e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[noise=x]\ dx=e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[noise\geq\theta-s_g(D_2)]$$
$$=e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[s_g(D_2)+noise\geq\theta].$$

That is, we have

$$\Pr[ns_g(D_1)\geq\theta]\leq e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_g(D_2)\geq\theta]. \quad (4)$$

Moreover, we can also prove

$$\Pr[noise\geq\theta-s_g(D_1)]$$
$$=\int_{\theta-s_g(D_1)}^{\infty}\Pr[noise=x]\ dx\geq\int_{\theta+1-s_g(D_2)}^{\infty}\Pr[noise=x]\ dx$$
$$=\int_{\theta-s_g(D_2)}^{\infty}\Pr[noise=x+1]\ dx\geq e^{-\frac{\epsilon_{c1}}{2n_i}}\int_{\theta-s_g(D_2)}^{\infty}\Pr[noise=x]\ dx$$
$$=e^{-\frac{\epsilon_{c1}}{2n_i}}\Pr[noise\geq\theta-s_g(D_2)].$$

That is, we also have

$$\Pr[ns_g(D_1)\geq\theta]\geq e^{-\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_g(D_2)\geq\theta]. \quad (5)$$

In a similar way, we can see that

$$\Pr[ns_g(D_1)<\theta]\leq e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_g(D_2)<\theta], \quad (6)$$

and

$$\Pr[ns_g(D_1)<\theta]\geq e^{-\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_g(D_2)<\theta]. \quad (7)$$

At last, based on equations (1), (4), (5), (6) and (7), we can prove that

$$\Pr[\text{Selecting subgraph } G \text{ from } C_i \text{ in } D_1\ ]$$
$$=\frac{0\times\Pr[ns_G(D_1)<\theta]+f(G,D_1)\times\Pr[ns_G(D_1)\geq\theta]}{\sum_{g\in C_i}(0\times\Pr[ns_g(D_1)<\ \theta]+f(g,D_1)\times\Pr[ns_g(D_1)\geq\theta])}$$
$$=\frac{f(G,D_1)\times\Pr[ns_G(D_1)\geq\theta]}{\sum_{g\in C_i}f(g,D_1)\times\Pr[ns_g(D_1)\geq\theta]}$$
$$\leq\frac{f(G,D_1)\times e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_G(D_2)\geq\theta]}{\sum_{g\in C_i}f(g,D_1)\times e^{-\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_g(D_2)\geq\theta]}$$
$$\leq\frac{e^{\frac{\epsilon_{c2}}{2n_i}}f(G,D_2)\times e^{\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_G(D_2)\geq\theta]}{\sum_{g\in C_i}e^{-\frac{\epsilon_{c2}}{2n_i}}\times f(g,D_2)\times e^{-\frac{\epsilon_{c1}}{2n_i}}\Pr[ns_g(D_2)\geq\theta]}$$
$$=e^{2\times\frac{\epsilon_{c1}}{2n_i}+2\times\frac{\epsilon_{c2}}{2n_i}}\frac{f(G,D_2)\times\Pr[ns_G(D_2)\geq\theta]}{\sum_{g\in C_i}f(g,D_2)\times\Pr[ns_g(D_2)\geq\theta]}$$
$$=e^{\frac{\epsilon_c}{n_i}}\frac{0\times\Pr[ns_G(D_2)<\ \theta]+f(G,D_2)\times\Pr[ns_G(D_2)\geq\theta]}{\sum_{g\in C_i}(0\times\Pr[ns_g(D_2)<\ \theta]+f(g,D_2)\times\Pr[ns_g(D_2)\geq\theta])}$$
$$=e^{\frac{\epsilon_c}{n_i}}\Pr[\text{Selecting subgraph } G \text{ from } C_i \text{ in } D_2].$$

Based on the above analysis, we can see that, in our conditional exponential method, each subgraph selection guarantees $\frac{\epsilon_c}{n_i}$-differential privacy. We iteratively select n_i subgraphs from the candidate i-subgraphs without replacement. By the sequential composition property [9], our conditional exponential method overall satisfies ϵ_c-differential privacy. □

VII. Lattice-based Noisy Support Derivation

After privately identifying frequent subgraphs, we now discuss how to compute their noisy supports. A simple method

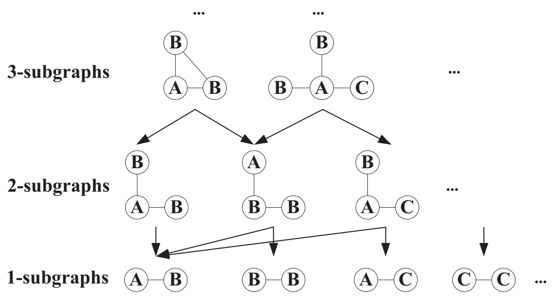


Fig. 2. A Lattice Formed by Graphs

is to uniformly assign the privacy budget to the support computation of each frequent subgraph and directly perturb their supports. However, it causes the amount of added noise to be proportionate to the number of frequent subgraphs. If the number of frequent subgraphs is large, the support of each frequent subgraph has to be perturbed by a large amount of noise, which reduces the accuracy of the noisy supports.

To this end, we devise a lattice-based noisy support derivation (*ND_2*) approach. The core of our *ND_2* approach is to leverage the inclusion relation between subgraphs [10] to reduce the amount of added noise. In our *ND_2* approach, we build a directed lattice based on the identified frequent subgraphs. In the lattice, each node n_g is associated with a frequent subgraph g, and the height of n_g is equal to the size of g. An edge from node n_{g_1} to node n_{g_2} is introduced if graph g_2 can be expanded to graph g_1 by adding one edge. An example of a lattice formed by graphs is shown in Fig. 2. In the lattice, there are several directed paths. For ease of presentation, we say a graph g is on a path p if g's corresponding node n_g is contained in p, and the depth of g on p is the number of nodes from the first node in p to n_g.

In our *ND_2* approach, we first propose a count accumulation method to compute the noisy support of the graphs on a given path of the lattice. Then, we present a path construction method which constructs multiple paths to cover all the frequent subgraphs, such that we can use the count accumulation method on these paths to obtain the noisy support of all the frequent subgraphs. At last, we also propose a path extension method, which further optimizes the constructed paths to improve the accuracy of the noisy supports.

Algorithm 4 Lattice-based Noisy Support Derivation

Input:
The Set of Frequent Subgraphs F; Privacy Budget ϵ_3;
Output:
A Map M of Frequent Subgraphs to Noisy Supports;

```
1: L ← build a directed lattice based on FS;
2: PS ← construct a path set based on L;          \\ see Sec. VII-B
3: PS' ← extend paths in PS;                      \\ see Sec. VII-C
4: for i from 1 to |PS'| do
5:     M_i ← obtain noisy supports of graphs on the i-th path in PS' by using privacy
       budget ε_3/|PS'|;                          \\ see Sec. VII-A
6: end for
7: M ← combine noisy supports of frequent subgraphs based on M_1 ... M_|PS'|;
8: return M;
```

Alg. 4 shows the main steps of our *ND_2* approach. In particular, given the identified frequent subgraphs, we first build a lattice based on these frequent subgraphs (line 1). Then, by using the path construction method, we construct multiple paths to cover all the nodes in the lattice (line 2). Next, we further optimize these paths by leveraging our path extension method (line 3). At last, for each resulting path, we utilize our count accumulation method to obtain the noisy support of the graphs on this path (line 5). If a graph is contained in more than one path, we will get multiple noisy supports of this graph. In this case, we combine these noisy supports to get a more accurate result (line 7).

In the rest of this section, we present the details of these methods. We do this in a reverse order, first presenting *count accumulation* in Sec. VII-A, then *path construction* in Sec. VII-B, and finally *path extension* in Sec. VII-C. In Sec. VII-D, we give the privacy guarantee of our *ND_2* approach.

A. Count Accumulation Method

Given a path of the lattice, we introduce a *count accumulation method* to compute the noisy support of the graphs on this path. The main idea is to leverage the inclusion relation between subgraphs and the parallel composition property of differential privacy to improve the accuracy of the results. Suppose a path p contains $|p|$ nodes, and these nodes represent $|p|$ graphs g_1, g_2, ..., $g_{|p|}$, where $g_1 \subseteq g_2 ... \subseteq g_{|p|}$. For the input database D, we can utilize these $|p|$ graphs to divide D into mutually disjoint sub-databases. In particular, as shown in Fig. 3, we first divide D into two sub-databases based on graph $g_{|p|}$: the sub-database $D_{|p|}$ that includes all the input graphs containing $g_{|p|}$ and the sub-database $D_{|\overline{p}|}$ that includes the remaining input graphs. Then, we further divide sub-database $D_{|\overline{p}|}$ based on graph $g_{|p|-1}$. This process continues until D is partitioned into $|p|$+1 disjoint sub-databases $D_{\overline{1}}$, D_1, ..., $D_{|p|}$.

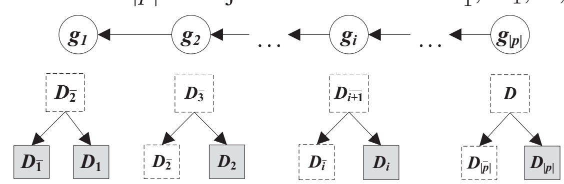


Fig. 3. The Sub-database Generation Process

For graphs g_j, g_i on path p, where $j{>}i$, we have $g_j{\supseteq}g_i$. By the inclusion relation between subgraphs, if an input graph contains g_j, it also contains g_i. Thus, the input graphs containing g_i are distributed across sub-databases $D_{|p|}$, $D_{|p|-1}$, ..., D_i. Let α_y be the number of input graphs in D_y. Then, the support of g_i is equal to $\sum_{m=i}^{|p|} \alpha_m$. Thus, based on $\alpha_{|p|}$, $\alpha_{|p|-1}$, ..., α_1, we can derive the support of each graph on path p. Suppose the privacy budget allocated to p is ϵ_p. We add Laplace noise $Lap(1/\epsilon_p)$ to each α_y, where $1{\leq}y{\leq}|p|$.

Theorem 7 *Our count accumulation method guarantees ϵ_p-differential privacy.*

Proof: In this method, we compute the noisy values of $\alpha_{|p|}$, $\alpha_{|p|-1}$, ..., α_1. For i from 1 to $|p|$, as an input graph can affect α_i by at most one, the sensitivity of computing α_i is one. Thus, adding Laplace noise $Lap(1/\epsilon_p)$ to α_i satisfies ϵ_p-differential privacy. Moreover, as sub-databases D_1, ..., $D_{|p|}$ are mutually disjoint, by the parallel composition property [9], the privacy budgets used in computing α_1, α_2, ..., $\alpha_{|p|}$ do not need to accumulate. For the graphs on path p, as their noisy supports are obtained based on the noisy values of α_1, ..., $\alpha_{|p|}$, we can safely use them without privacy implications. Therefore, our count accumulation method satisfies ϵ_p-differential privacy. □

In what follows, we will show that, compared with directly perturbing the support of each graph on a path, our count accumulation method can significantly improve the accuracy of the noisy supports. In particular, we use the variance to measure the errors of the noisy supports. Continue from above example, if we directly perturb the support of the $|p|$ graphs on path p, as an input graph can affect the support of a graph by at most one, the sensitivity of computing the support of a graph is one. For each graph on p, we uniformly assign it a privacy budget $\epsilon_p/|p|$, and add noise $Lap(|p|/\epsilon_p)$ to its support, which has variance $2|p|^2/\epsilon_p^2$. We can see the error of the noisy support of each graph is quadratic to the depth of path p. In contrast, in our count accumulation method, we add Laplace noise $Lap(1/\epsilon_p)$ to α_i, which has variance $2/\epsilon_p^2$. For the i-th graph g_i, its noisy support sup_i is equal to the sum of the noisy values of α_i, α_{i+1}, ..., $\alpha_{|p|}$. Since the noise added to α_i, α_{i+1}, ..., $\alpha_{|p|}$ is generated independently, the variance of sup_i is the sum of the variances of α_i, α_{i+1}, ..., $\alpha_{|p|}$, which is $2(|p|$ - i + $1)/\epsilon_p^2$. We can see, in our method, the error of the noisy support of a graph is proportionate to its depth on path p. Thus, our count accumulation method can significantly improve the accuracy of the noisy supports.

B. Path Construction Method

In the constructed lattice, we can select multiple paths to cover all frequent subgraphs, and use the count accumulation method on these paths to obtain the noisy support of each frequent subgraph. Assume the set of selected paths is *PS* = $\{p_1, p_2, ..., p_t\}$ and the privacy budget is ϵ_3. We allocate each path in *PS* a privacy budget ϵ_3/t. For the i-th graph g_i on a path p_m, based on the count accumulation method, the variance of its noisy support is $2t^2(|p_m|$-i+$1)/\epsilon_3^2$. We can see the number of paths t is a quadratic factor on the error of the noisy supports. Thus, to improve the accuracy of the noisy supports, we should select as few paths as possible to cover all frequent subgraphs. We formulate this problem as follows.

Problem 1 *(Path Selection): Given a lattice L, where each node represents a frequent subgraph, the set of nodes in L is NS and the set of nodes on a path p is $ns(p)$. Find a path set PS, such that $\bigcup_{p \in PS} ns(p) = NS$, and the number of paths in PS is minimized.*

In the constructed lattice, each node can be considered as an element with weight 1. Each path can be considered as an item with profit 1, which is a subset of elements. The total weight of a set of items is given by the total weight of the elements in the union of the items in this set. Suppose there are $|NS|$ nodes in the lattice. Then, the path selection problem is equivalent to finding a set of items such that the profit is minimized and the total weight is equal to $|NS|$. It can be reduced from a variant of the Set-Union Knapsack Problem [25], which is known to be NP-hard.

To this end, we propose a heuristic method, called *path construction*, to construct a set of paths. The main idea is to iteratively construct multiple paths, where each path is appended with as many nodes as possible. The details of this method are shown in Alg. 5. It has time complexity $O(|NS|^2)$, where $|NS|$ is the number of nodes in the lattice.

Algorithm 5 Path Construction

Input:
Lattice L;
Output:
A Path Set PS;
1: $PS \leftarrow \emptyset$;
2: $TN \leftarrow$ find all the nodes with in-degree 0 in L;
3: Sort nodes in TN in decreasing order based on their heights;
4: **while** $TN \neq \emptyset$ **do**
5: $p \leftarrow \emptyset$; $n \leftarrow$ get a node from TN;
6: Add node n into path p;
7: **while** $true$ **do**
8: *childSet* $\leftarrow$ find the set of the child nodes of n;
9: **if** *childSet* == $\emptyset$ **then**
10: break;
11: **end if**
12: $n \leftarrow$ find the node with highest height from *childSet*;
13: Add node n into path p;
14: **end while**
15: Remove the nodes in p from L;
16: Add new nodes with in-degree 0 into TN; Add p into PS;
17: **end while**
18: *return* PS;

In particular, given the lattice, we first find the top nodes whose in-degrees are zero (line 2). Let TN denote the set of these nodes. For example, in the lattice shown in Fig. 4, nodes n_1 and n_2 are the top nodes. For each node with in-degree zero, it can only be the first node in a path. Thus, to cover such a node, we need to construct a new path. Since the path starting with a node at higher height is able to contain more nodes, we sort the nodes in TN in decreasing order based on their heights (line 3). If some nodes are at the same height, as the nodes with larger out-degrees have more candidate nodes to append, we arrange these nodes in ascending order based on their out-degrees. In Fig. 4, since the out degree of n_1 is smaller than that of n_2, the position of n_1 in TN is less than that of n_2. Next, we gradually construct paths according to the resulting order in TN.

For each node in TN, we insert it as the first node in a new path (line 6). In line 7-15, we iteratively append nodes to a path and remove them from the lattice. Specifically, after appending a node n to a path p, we decide whether to continue to extend p based on the set of the child nodes of n. If this set is empty, the construction process of p is ended and p is output as a resulting path. Otherwise, we append a child node whose height is highest to p. If multiple child nodes are at the same height, since the nodes with larger in-degrees have a higher chance to be appended to other paths, we append the node with minimal in-degree first.

After we construct a path p, to differentiate the nodes still not contained in any constructed paths, we remove the nodes in p from the lattice (line 15). Specifically, for each node n in p, suppose n_p is one of its parent nodes and n_c is one of its child nodes. If n_p and n_c are not connected after removing node n, a new edge from n_p to n_c is introduced. For example, in Fig. 4, after constructing path p_1, we remove node n_4, which causes nodes n_2 and n_5 not connected. In this case, a new edge from n_2 to n_5 is introduced. Moreover, after removing the nodes in p, if the in-degrees of some nodes in the lattice decrease to zero, we insert them into set TN (line 16). The positions of these nodes in TN are determined by the heights and out-degrees of these nodes.

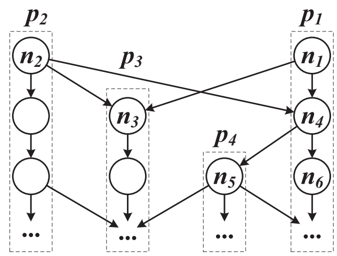


Fig. 4. A Set of Paths on A Lattice

C. Path Extension Method

After constructing a set of paths, we can use the count accumulation method on these paths to obtain the noisy support of the frequent subgraphs. Suppose the constructed path set is *PS*. As shown in Sec. VII-B, for each frequent subgraph, we can efficiently compute the variance of its noisy support based on the allocated privacy budget and *PS*.

We observe, given a path in *PS*, its first (last) node may have parent (child) nodes in the lattice. For example, for path p_3 in Fig. 4, its first node n_3 has two parent nodes n_1 and n_2. For node n_1, it is already contained in path p_1. Suppose g_1 is the associated graph of n_1. Based on path p_1, we can obtain a noisy support sup_1 of g_1 with variance v_1. Moreover, if we add n_1 into path p_3, we can utilize p_3 to get another noisy support sup'_1 for g_1 with variance v'_1. Then, we can aggregate these two noisy supports and get a new noisy support $\frac{v'_1 \times sup_1 + v_1 \times sup'_1}{v_1 + v'_1}$ with variance $\frac{v_1 \times v'_1}{v_1 + v'_1}$. Clearly, $\frac{v_1 \times v'_1}{v_1 + v'_1}$ is smaller than v_1 and v'_1. Thus, by extending paths, we can improve the accuracy of the noisy supports.

We also notice that, in the lattice, the first (last) node of a path may have more than one parent (child) node. When we extend a path, due to the inclusion constraint in the count accumulation method, we can only add one of these parent (child) nodes into the path. For example, in Fig. 4, only one node between n_1 and n_2 can be added into path p_3. Let g_1 and g_2 be the associated graph of n_1 and n_2, and v_1 and v_2 be the variance of the noisy support of g_1 and g_2, respectively. After adding n_1 or n_2 into p_3, we can utilize p_3 to obtain another noisy support of g_1 or g_2. Suppose the variance of the new noisy support is v. In particular, if n_1 is added into p_3, we can update the variance of the noisy support of g_1 to $\frac{v \times v_1}{v + v_1}$, while the variance of the noisy support of g_2 is not changed. In contrast, if n_2 is added into p_3, the variance of the noisy support of g_2 is updated to $\frac{v \times v_2}{v + v_2}$ while the variance of the noisy support of g_1 is not changed. Suppose $v_1 > v_2$. Then,

$$v_1 + \frac{v_2 \times v}{v_2 + v} - (\frac{v_1 \times v}{v_1 + v} + v_2)$$
$$= \frac{v_1 \times v_2 \times (v_1 - v_2) + (v_1^2 - v_2^2) \times v}{(v_1 + v) \times (v_2 + v)} > 0$$

That is, $\frac{v_1 \times v}{v_1 + v} + v_2 < v_1 + \frac{v_2 \times v}{v_2 + v}$. Thus, if we can only add one node into a path, we should select the node whose associated graph has the largest noisy support variance, such that the overall accuracy can be improved the most.

Based on the above analysis, we propose a *path extension method*, which iteratively inserts nodes at the first and last positions of each constructed path. This method is shown in Alg. 6. In particular, each node in the lattice has an attribute v, which represents the variance of the noisy support of its associated graph. In Alg. 6, we first compute the attribute v of each node in the lattice based on the path set *PS* (line 2).

Then, for each constructed path p, we iteratively insert nodes at its first position. Based on the count accumulation method, if a node is inserted at the first position of a path, the noisy support variances of the other graphs on this path are all increased. Thus, whether to insert a node at the first position of a path relies on the overall change of the noisy support variances. Specifically, for the first node in path p, we first find all its parent nodes from the lattice, and select the node n_p whose attribute v_p is the largest (line 8). Then, we compute the expected decrement dec of v_p if n_p is inserted at the first position of p (line 9). We also compute the expected increment of the attribute v of the other nodes in p, and sum them up to get the overall increment inc (line 10). If dec is no larger than inc, the process of inserting nodes at the first position of p is ended. Otherwise, we insert n_p at the first position of p and update the attribute v of each node in p (line 15).

After that, we iteratively insert nodes at the last position of path p. Unlike inserting nodes at the first position of a path, if a node is inserted at the last position, the attribute v of the other nodes in this path is not affected. Specifically, for the last node in p, we first find all its child nodes from the lattice. Then, we find the child node n_c whose attribute v_c is the largest (line 22) and insert it at the last position of p (line 23). Next, we update the attribute v_c of n_c based on path p (line 24). This process is applied recursively until the set of the child nodes of the last node in p is empty.

Algorithm 6 Path Extension

Input:
Lattice L; Path Set *PS*;
Output:
Improved Path Set PS';

```
1: PS' ← ∅;
2: Compute the attribute v of each node in L based on PS;
3: for each path p in PS do
4:     /**** Iteratively Inserting Nodes at the First Position ****/
5:     n_f ← get the first node in p;
6:     parentSet ← find the set of all parent nodes of n_f in L;
7:     while parentSet ≠ ∅ do
8:         n_p ← find the node with largest attribute v from parentSet;
9:         dec ← get expected decrement of n_p's attribute v_p;
10:        inc ← sum expected increment of attribute v of nodes in p;
11:        if dec ≤ inc then
12:            break;
13:        end if
14:        Insert n_p at the first position of p;
15:        Update the attribute v of each node in p;
16:        parentSet ← find the set of all parent nodes of n_p in L;
17:    end while
18:    /**** Iteratively Inserting Nodes at the Last Position ****/
19:    n_l ← get the last node in p;
20:    childSet ← find the set of all child nodes of n_l in L;
21:    while childSet ≠ ∅ do
22:        n_c ← find the node with largest attribute v from childSet;
23:        Insert n_c at the last position of p;
24:        Update the attribute v_c of n_c;
25:        childSet ← find the set of all child nodes of n_c in L;
26:    end while
27:    Add p into PS'.
28: end for
29: return PS';
```

D. Privacy Analysis of ND_2 Approach

In what follows, we give the privacy guarantee of our lattice-based noisy support derivation (*ND*$_2$) approach (i.e., Alg. 4).

Theorem 8 *Our ND_2 approach is ϵ_3-differentially private.*

Proof: In Alg. 4, we first build a lattice based on the identified frequent subgraphs. We only utilize the inclusion relation between identified frequent subgraphs without accessing the input database. Thus, we can safely use this lattice. Moreover, as shown in Alg. 5, to find a path set to cover all the frequent subgraphs, we do not use any other information but only rely on the lattice. Thus, the constructed path set does not breach the privacy either. For the path extension method, since it only depends on the lattice and the constructed paths, we can safely use it without privacy implications.

Suppose the resulting path set is *PS* = $\{p_1, p_2, ..., p_t\}$. We uniformly assign each path in *PS* a privacy budget ϵ_3/t. Then, we use our count accumulation method to obtain the noisy support of the graphs in each path. As shown in Thm. 7, it achieves ϵ_3/t-differential privacy on each path. Based on the sequential composition property [9], it overall satisfies ϵ_3-differential privacy. After that, if we obtain multiple noisy supports of a graph, we combine them to get a more accurate result. This is done without reference to the input database, so the results still satisfy differential privacy. Overall, our ND_2 approach is ϵ_3-differentially private. □

VIII. *DFG* Algorithm

A. *DFG Algorithm Description*

Our *DFG* algorithm is shown in Alg. 7. In particular, we first estimate the maximal size M_g of frequent subgraphs. To estimate M_g, we compute an array $\zeta = \{\zeta_1, ..., \zeta_n\}$, where ζ_i is the maximal support of i-subgraphs (line 1). For a given threshold θ, the maximal size M_g is the number of elements in ζ larger than θ. Thus, we can utilize our binary estimation method to estimate the number of elements in ζ larger than θ, and set maximal size M_g to be such number (line 2).

After that, we utilize our frequent subgraph identification approach to identify frequent subgraphs from the input database (line 3). At last, we leverage our lattice-based noisy support derivation approach to compute the noisy support of each identified frequent subgraph (line 4).

Algorithm 7 *DFG* Algorithm

Input:
Graph Database D; Threshold θ; Privacy Budgets ϵ_1, ϵ_2, ϵ_3;
Output:
A Map R of Frequent Subgraphs to Noisy Supports;
1: $\zeta \leftarrow$ get the maximal support of subgraphs with different sizes;
2: $M_g \leftarrow$ get the maximal frequent subgraph size based on ζ using ϵ_1;
3: $F \leftarrow$ *frequent_subgraph_identification* (D, θ, ϵ_2, M_g);
4: $R \leftarrow$ *lattice-based_noisy_support_derivation* (F, ϵ_3);
5: *return* R;

B. *Privacy Analysis of DFG Algorithm*

Theorem 9 *Our DFG algorithm is ϵ-differentially private.*

Proof: In our *DFG* algorithm, based on the binary estimation method, we first estimate the maximal size M_g of frequent subgraphs by using ϵ_1. According to Thm. 5, we can see that it satisfies ϵ_1-differential privacy.

Then, we utilize our frequent subgraph identification (FI_1) approach to privately identify frequent subgraphs in order of increasing size. For the identification of frequent i-subgraphs, we first use our binary estimation method to estimate the number of frequent i-subgraphs, which achieves ϵ_b-differential privacy. After that, we leverage our conditional exponential method to privately select frequent i-subgraphs. Based on Thm. 6, it guarantees ϵ_c-differential privacy. For our FI_1 approach, by the sequential composition property [9], we can see that it satisfies $((\epsilon_b+\epsilon_c)\times M_g)$ = ϵ_2-differential privacy.

At last, we use our lattice-based noisy support derivation (ND_2) approach to compute the noisy support of identified frequent subgraphs. In Thm. 8, we show that our ND_2 approach satisfies ϵ_3-differential privacy. In summary, by the sequential composition property [9], we can conclude our *DFG* algorithm achieves $(\epsilon_1+\epsilon_2+\epsilon_3)$ = ϵ-differential privacy. □

IX. Experiments

In this section, we evaluate the performance of our *DFG* algorithm. We compare it with the following two algorithms. The first is the straightforward approach proposed in Sec. IV (referred to as *naive*), which satisfies ϵ-differential privacy. The second is the algorithm proposed in [4] (referred to as *DFPM*), which guarantees the weaker (ϵ, δ)-differential privacy. We implement all these algorithms in Java. The experiments are conducted on a PC with Intel Core2 Duo E8400 CPU (3.0GHz) and 4GB RAM. Due to the randomness of the algorithms, we run every algorithm ten times and report the average results. In these experiments, we use the *relative threshold* (i.e., the percentage of input graphs). In *DFG*, we allocate the total privacy budget ϵ as follows: ϵ_1 = 0.1ϵ, ϵ_2 = 0.5ϵ and ϵ_3 = 0.4ϵ. The default value of ϵ is 0.2. In Sec. IX-B, we also present the experimental results when ϵ is varied.

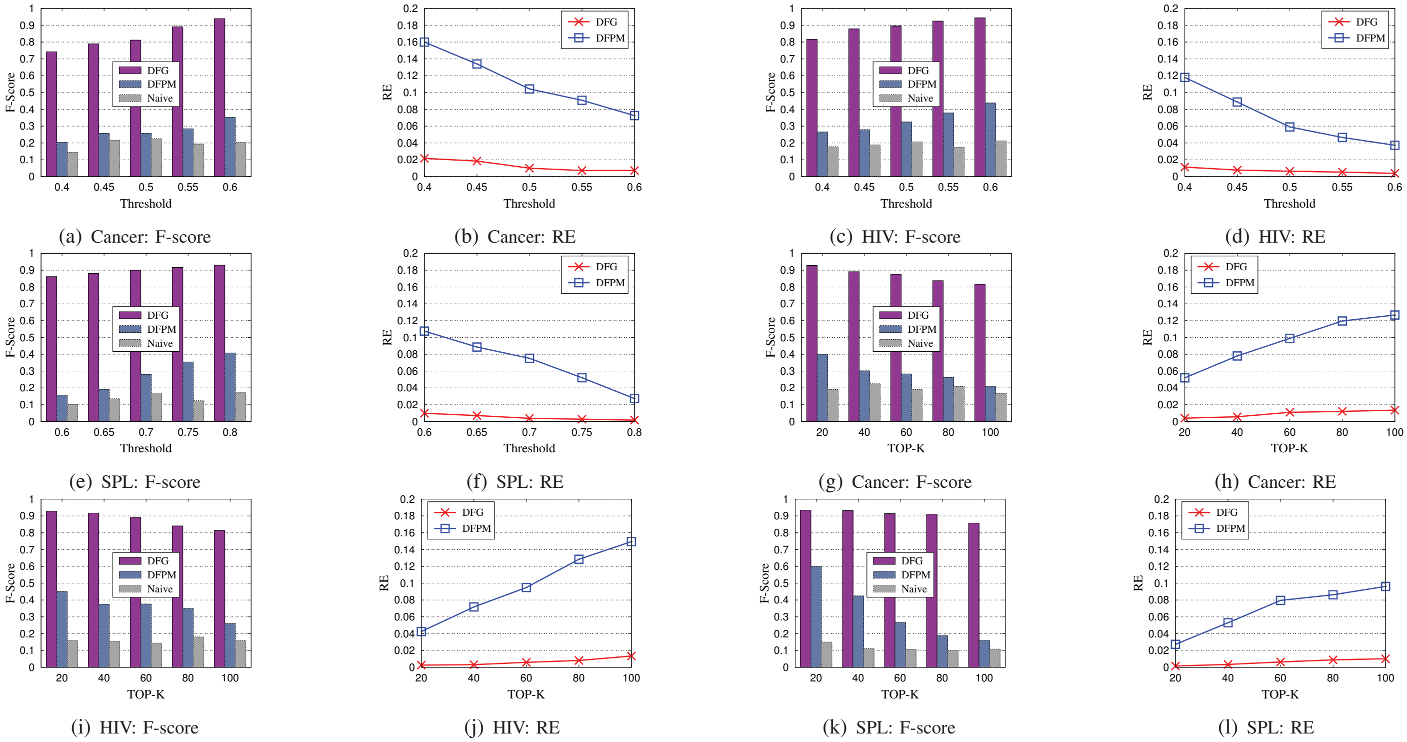
TABLE I
Summary of Graph Datasets

Dataset	♯Graphs	Avg.size	Max.size	♯Vertex Labels
Cancer	32557	28.3	236	67
HIV	42689	27.5	247	65
SPL	53804	47.5	831	104

In the experiments, we use three publicly available real datasets: *Cancer* [26], *HIV* [26] and *SPL* [27]. The characteristics of these datasets are summarized in Tab. I. Besides, we employ two widely used metrics: *F-score* [7] and *Relative Error* (*RE*) [6]. *F-score* is used to measure the utility of generated frequent subgraphs, while *RE* is used to measure the error with respect to the true supports of frequent subgraphs.

A. *Frequent Subgraph Mining*

Fig. 5(a) - 5(f) show the performance of these three algorithms with different values of threshold. For *DFPM*, it only guarantees the weaker (ϵ, δ)-differential privacy. Moreover, *DFPM* is designed for top-k FGM. In this experiment, we consider the scenario where *DFPM* sets k to be the number of frequent subgraphs for the given threshold. Notice that, such setting might violate the privacy. However, even with these privacy relaxations, we observe *DFG* significantly outperforms *DFPM*. In *DFPM*, to find a frequent subgraph, it first randomly generates a graph and then performs random walk in the output space. The transition is determined based on the supports of the current graph and its neighboring graphs. However, if the first generated graph is in a region where the supports of all neighboring graphs are very low, the random walk will seldom move and an infrequent subgraph will be output, which negatively affects the utility of the results.



(a) Cancer: F-score (b) Cancer: RE (c) HIV: F-score (d) HIV: RE

(e) SPL: F-score (f) SPL: RE (g) Cancer: F-score (h) Cancer: RE

(i) HIV: F-score (j) HIV: RE (k) SPL: F-score (l) SPL: RE

Fig. 5. Frequent Subgraph Mining

For algorithm *naive*, as its results in metric RE are always orders of magnitudes larger than the other comparison algorithms (usually above 1000), we omit it to ensure the readability of the figures. Fig. 5(a) - 5(f) show *DFG* significantly outperforms *naive*. In *naive*, it perturbs the support of all candidate subgraphs. However, as discussed in Sec. IV, a large number of candidate subgraphs are generated in the mining process, causing a large amount of perturbation noise. Thus, *naive* gets poor performance in terms of both F-score and RE.

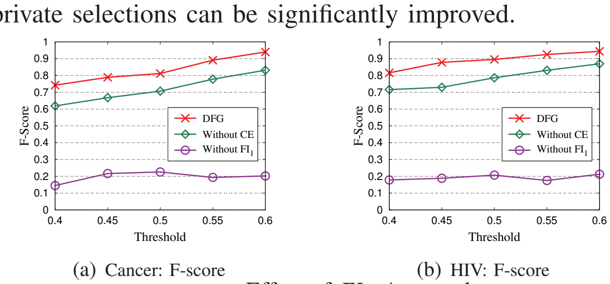
We also compare the performance of these algorithms for top-k FGM. To extend *DFG* and *naive* to discover the k most frequent subgraphs, we adapt them by setting the threshold to be the support of the k-th frequent subgraph. To avoid privacy breach, we add Laplace noise to that computation since the sensitivity of such computation is one. Fig. 5(g) - 5(l) show the results by varying the k parameter from 20 to 100. We can see our *DFG* algorithm achieves better performance.

B. Effect of Privacy Budget

Fig. 6 shows the performance of these three algorithms for mining top-50 frequent subgraphs under varying privacy budget ϵ on datasets Cancer and HIV. We can see *DFG* consistently gains better performance at the same level of privacy. All these algorithms perform in a similar way: the utility of the results is improved when ϵ increases. This is because, when ϵ increases, a smaller amount of noise is required and a lower degree of privacy is guaranteed.

C. Effect of FI_1 Approach

In this experiment, we study how our frequent subgraph identification (FI_1) approach affects the performance of *DFG* on datasets Cancer and SPL. From Fig. 7, we can see, without using FI_1, simply perturbing the support of candidate subgraphs to identify frequent subgraphs produces poor results. Moreover, we also compare *DFG* to a private algorithm which uses the exponential mechanism to privately select subgraphs from candidate subgraphs. From Fig. 7, we can see, by using our conditional exponential (CE) method, the utility of the results is obviously improved. It is in line with our analysis: by effectively reducing the candidate set, the accuracy of the private selections can be significantly improved.

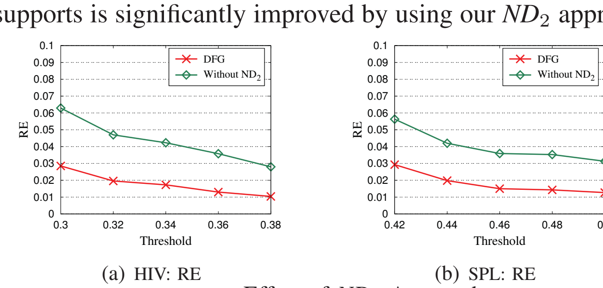


(a) Cancer: F-score (b) HIV: F-score

Fig. 7. Effort of FI_1 Approach

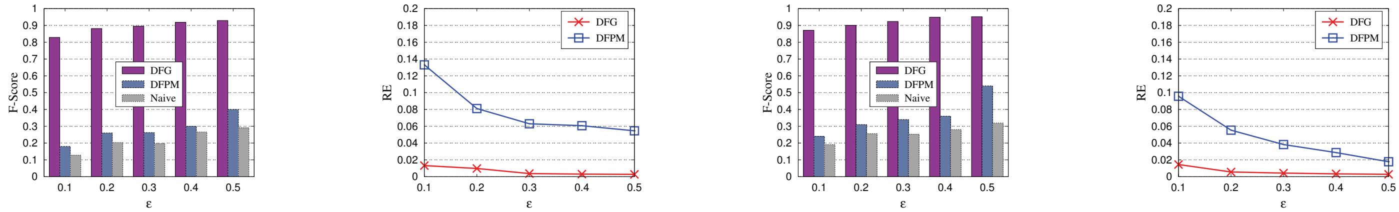
D. Effect of ND_2 Approach

We also study the effect of our lattice-based noisy support derivation (ND_2) approach. We compare *DFG* to a private algorithm which directly perturbs the support of each identified frequent subgraph. Fig. 8 shows the accuracy of the noisy supports is significantly improved by using our ND_2 approach.



(a) HIV: RE (b) SPL: RE

Fig. 8. Effort of ND_2 Approach



(a) Cancer: F-score (b) Cancer: RE (c) HIV: F-score (d) HIV: RE

Fig. 6. Effect of Privacy Budget

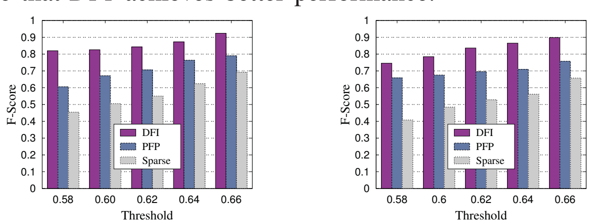
E. Benefit of FI_1 Approach

To better understand the benefit of our frequent subgraph identification (FI_1) approach, we apply it to frequent itemset mining FIM (referred to as *DFI*). In particular, given the candidate i-itemsets, *DFI* uses our binary estimation method to estimate the number of frequent i-itemsets, and uses our conditional exponential method to privately select itemsets from candidate i-itemsets. In this experiment, we compare *DFI* to the state-of-the-art on differentially private FIM algorithm PFP [11]. In addition, we also apply the Sparse Vector Technique [28] to FIM. Specifically, given the candidate i-itemsets, it uses the Sparse Vector Technique to privately identify frequent i-itemsets from candidate i-itemsets.

TABLE II
TRANSACTION DATASET CHARACTERISTICS

Dataset	♯Transactions	♯Items	Avg.length
Pumsb-star	49046	2088	50.5
Accidents	340183	468	33.8

In the experiments, two public transaction datasets are used. A summary of the characteristics of these datasets is illustrated in Tab. II. We show the experimental results in Fig. 9. We can see that *DFI* achieves better performance.



(a) Pumsb-star (b) Accidents

Fig. 9. Applying FI_1 Approach to Frequent Itemset Mining

X. CONCLUSION

In this paper, we investigate the problem of designing a differentially private FGM algorithm. We introduce a novel differentially private FGM algorithm, called *DFG*. In this algorithm, we first present a frequent subgraph identification approach to privately identify frequent subgraphs from the input graphs. In this approach, a binary estimation method is proposed to estimate the number of frequent subgraphs for a certain graph size, and a conditional exponential method is proposed to improve the accuracy of the private subgraph selections through candidates pruning. Then, we devise a lattice-based noisy support derivation approach to compute the noisy support of each identified frequent subgraph, where a series of methods has been proposed to improve the accuracy of the noisy supports. Through privacy analysis, we prove that our *DFG* algorithm satisfies ϵ-differential privacy. Extensive experiments on real datasets show that our *DFG* algorithm can privately find frequent subgraphs with high data utility.

ACKNOWLEDGMENT

The work was supported by National Natural Science Foundation of China under grant 61502047, the National Institute of Health (NIH) under award #R01GM114612, and the Patient-Centered Outcomes Research Institute (PCORI) under award #ME-1310-07058.

REFERENCES

[1] C. Dwork. Differential privacy. In *ICALP*, 2006.
[2] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Base Syst*, 2002.
[3] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.
[4] E. Shen and T. Yu. Mining frequent graph patterns with differential privacy. In *KDD*, 2013.
[5] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.
[6] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: frequent itemset mining with differential privacy. In *VLDB*, pages 305–316, 2012.
[7] C. Zeng, J. F. Naughton, and J-Y. Cai. On differentially private frequent itemset mining. In *VLDB*, 2012.
[8] S. Xu, S. Su, X. Cheng, Z. Li, and L. Xiong. Differentially private frequent sequence mining via sampling-based candidate pruning. In *ICDE*, 2015.
[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.
[10] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.
[11] S. Su, S. Xu, X. Cheng, Z. Li, and F. Yang. Differentially private frequent itemset mining via transaction splitting. *TKDE*, 2015.
[12] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. In *VLDB*, 2011.
[13] S. Chen and S. Zhou. Recursive mechanism: towards node differential privacy and unrestricted joins. In *SIGMOD*, 2013.
[14] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. In *VLDB*, 2014.
[15] J. Zhang, G. Cormode, C. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *SIGMOD*, 2015.
[16] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *KDD*, 2010.
[17] X. Cheng, S. Su, S. Xu, and Z. Li. Dp-apriori: A differentially private frequent itemset mining algorithm based on transaction splitting. *Computers & Security*, 2015.
[18] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, 2000.
[19] H. Li, L. Xiong, and X. Jiang. Differentially private synthesization of multi-dimensional data using copula functions. In *EDBT*, 2014.
[20] J. Lee and C. Clifton. Top-k frequent itemsets via differentially private fp-trees. In *KDD*, 2014.
[21] Y. Chen and A. Machanavajjhala. On the privacy properties of variants on the sparse vector technique. *CoRR*, 2015.
[22] L. Bonomi and L. Xiong. A two-phase algorithm for mining sequential patterns with differential privacy. In *CIKM*, 2013.
[23] C. Xiang, S. Su, S. Xu, P. Tang, and Z. Li. Differentially private maximal frequent sequence mining. *Computers & Security*, 2015.
[24] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *TKDE*, 2004.
[25] H. Kellerer, U. Pferschy, and D. Pisinger. *Introduction to NP-Completeness of knapsack problems*. Springer-Verlag, 2004.
[26] Nci. http://cactus.nci.nih.gov/download/nci.
[27] Fda. http://cactus.nci.nih.gov/download/fda.
[28] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 2013.