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
# {dear_shengzh, *susan, *chengxiang, *iejr}@bupt.edu.cn
† Math and Computer Science Department, Emory University, Atlanta, GA
5lxiong@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.

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 frequent subgraph 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 \(\epsilon\)-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 \(\epsilon\)-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 \(\epsilon\)-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 \((\epsilon, \delta)\)-differential privacy, which is a relaxed version of \(\epsilon\)-differential privacy. In contrast, we formally prove our \(DFG\) algorithm satisfies the standard \(\epsilon\)-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 [23], Cheng et al. introduce a differentially private FIM algorithm. They propose a generalized sparse vector technique to identify frequent itemsets, which perturbs the threshold and the support of each candidate itemset.

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

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

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

A fundamental concept for achieving differential privacy is sensitivity [9]. It is used to measure the largest change in the outputs over any two neighboring databases.
Theorem 1 For any function \( f : D \to \mathbb{R}^n \) with sensitivity \( \Delta f \), the algorithm \( A \)
\[ A(D) = f(D) + \text{Lap}(\Delta f/\epsilon) \]
achieves \( \epsilon \)-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 draws 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 \( u : D \times \mathcal{R} \to \mathbb{R} \), the algorithm \( A \)
\[ \Pr[A(D) = r] \propto \exp\left(\frac{\epsilon \times u(D,r)}{2 \Delta u}\right) \]
satisfies \( \epsilon \)-differential privacy, where \( \Delta u \) 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 \( A_1, \ldots, A_t \) be \( t \) algorithms, and each algorithm satisfies \( \epsilon_i \)-differential privacy. A sequence of algorithms \( A_i(D) \) over database \( D \) provides \((\sum \epsilon_i)\)-differential privacy.

Theorem 4 (Parallel Composition). Let \( A_1, \ldots, A_t \) be \( t \) algorithms, each satisfies \( \epsilon_i \)-differential privacy. A sequence of algorithms \( A_i(D_i) \) over disjoint databases \( D_1, \ldots, D_t \) provides \( \max(\epsilon_i) \)-differential privacy.

IV. A STRAIGHTFORWARD APPROACH

In this section, we introduce a straightforward approach to make FGM satisfy \( \epsilon \)-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 \( S_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_n}\} \) 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 \( M_f \). We uniformly assign the support computations of \( S_i \) a privacy budget \( \frac{\epsilon}{M_f} \). Moreover, since a single input graph can affect the support of each candidate \( i \)-subgraph by at most one, the sensitivity \( \Delta_i \) of computing \( S_i \) is the number of candidate \( i \)-subgraphs. Thus, adding Laplace noise \( \text{Lap}(\frac{\Delta_i \times \epsilon}{M_f}) \) in \( S_i \) achieves \( \frac{\epsilon}{M_f} \)-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 \( \epsilon \)-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 \( \epsilon \)-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.

In this section, we introduce our frequent subgraph identification (FI) 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 \(\theta\); Privacy Budgets \(\epsilon_2\); Maximal size of frequent subgraph \(M_g\);

Output:
- Frequent Subgraphs \(F\);

\begin{enumerate}
\item \(F \leftarrow \emptyset\); \(\epsilon_b = \frac{\log 2}{M_g}\); \(\epsilon_c = \frac{1-a(n)(\log 2)}{M_g}\);
\item for \(i\) from 1 to \(M_g\) do
\item \(C_i \leftarrow \text{generate the set of candidate } i\text{-subgraphs} \); \(\epsilon_2\)
\item \(n_i = \text{binary estimation}(C_i, \epsilon_b, \theta)\); \(\epsilon_2\)
\item \(F_i = \text{conditional exponential}(C_i, \epsilon_c, \theta, n_i)\); \(\epsilon_2\)
\item \(F \leftarrow F_i\);
\end{enumerate}

end for

8: return \(F\);

Alg. 1 shows the main steps of our \(FL_1\) 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 (1-1)-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 \(\epsilon_2\)-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 \(\text{Lap}(\log 2 |C_i|/\epsilon_2)\) in this computation satisfies \((\epsilon_2/\log 2 |C_i|)\)-differential privacy.

In this method, like the binary search, we only need to obtain the support of at most \(\lfloor\log 2 |C_i|\rfloor\) candidate \(i\)-subgraphs. By the sequential composition property [9], we can see our binary estimation method satisfies \(\epsilon_2\)-differential privacy. \(\square\)

Algorithm 2 Binary Estimation

Input:
- Candidate \(i\)-subgraphs \(C_i\); Privacy Budget \(\epsilon_2\); Threshold \(\theta\);

Output:
- The number of frequent \(i\)-subgraphs \(n_i\);

1: \(s \leftarrow 0\); \(\text{high} \leftarrow |C_i| \cdot 1\);
2: \(\text{while } s < \text{high} \text{ do} \)
3: \(m \leftarrow \lfloor(s + \text{high})/2\rfloor\);
4: \(s_m \leftarrow \text{get support of the candidate } i\text{-subgraph with the } m\text{-th largest support} \);
5: \(n_{m} \leftarrow s_m \cdot \text{Lap}(\log 2 |C_i|/\epsilon_2)\);
6: \(\text{if } n_{m} > 0 \text{ then} \)
7: \(\text{return } |C_i| \cdot m\);
8: \(\text{else if } n_{m} > 0 \text{ then} \)
9: \(\text{high} \leftarrow m + 1\);
10: \(\text{else if } n_{m} < 0 \text{ then} \)
11: \(\text{low} \leftarrow m + 1\);
12: \(\text{end if} \)
13: \(\text{end while} \)
14: \(\text{return } |C_i| \cdot 1 \cdot \text{high}\);

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 \(\epsilon_c\), we divide it into two parts \(\epsilon_{c_1}\) and \(\epsilon_{c_2}\), 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 \(\text{Lap}(\frac{1}{e_2})\) 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 $\epsilon_c$; Threshold $\theta$;
The number of frequent i-subgraphs $n_i$;
Output:
Frequent i-subgraphs $F_i$;
1: $\epsilon_1 \leftarrow \beta \epsilon_c, \epsilon_2 \leftarrow (1-\beta)\epsilon_c$;
2: for $j$ from 1 to $n_i$ do
3: Set set $\varnothing$;
4: for each subgraph $g$ in $C_i$ do
5: $n_s_g = s_g + \text{Lap}(\frac{2\epsilon_1}{n_i})$;
6: if $n_s_g \geq \theta$ then
7: Add $g$ into set;
8: end if
9: end for
10: $g_j \leftarrow$ select a subgraph from set without replacement such that Pr[Selecting subgraph $g] \propto \exp(-\frac{\epsilon_2 x_g}{n_i})$, where $s_g$ is the true support of subgraph $g$;
11: $F_i = g_j$;
12: end for
13: return $F_i$;

Privacy Analysis of Conditional Exponential Method.

Theorem 6 The conditional exponential method (i.e., Algorithm 3) satisfies $\epsilon_c$-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

$$ u(g, D) = \begin{cases} 0 & \text{if } s_g(D) > \theta \\ \exp(-\frac{\epsilon_2 x_g}{2n_i}) & \text{if } n_s_g(D) \geq \theta \end{cases} $$

where $s_g(D)$ and $n_s_g(D)$ 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 $f(g, D) = \exp(-\frac{\epsilon_2 x_g}{2n_i})$. Then, we can prove that

$$ \exp(-\frac{\epsilon_2 c}{2n_i}) \leq f(g, D_1) = \frac{f(g, D_2)}{f(g, D_2)} \leq \exp(\frac{\epsilon_2 c}{2n_i}). \quad (1) $$

In this method, we add Laplace noise $\text{Lap}(\frac{2\epsilon_2}{n_i})$ 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

$$ \frac{\text{Pr}[\text{noise} = x]}{\text{Pr}[\text{noise} = x + 1]} = \frac{\exp(-\frac{\epsilon_2 c}{2n_i})}{\exp(-\frac{\epsilon_2 c}{2n_i})} = \exp(\frac{\epsilon_1}{2n_i}|X + 1-x|) \leq \exp(\frac{\epsilon_1}{2n_i}). \quad (2) $$

Similarly, we also have

$$ \frac{\text{Pr}[\text{noise} = x + 1]}{\text{Pr}[\text{noise} = x]} \leq \exp(\frac{\epsilon_1}{2n_i}). \quad (3) $$

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

$$ \text{Pr}[s_g(D_1) + \text{noise} \geq \theta] = \text{Pr}[\text{noise} \geq \theta - s_g(D_1)] = \int_{\theta-s_g(D_1)}^{\infty} \text{Pr}[\text{noise} = x] \, dx \leq \int_{\theta-s_g(D_2)-1}^{\infty} \text{Pr}[\text{noise} = x] \, dx = \int_{\theta-s_g(D_2)}^{\infty} \text{Pr}[\text{noise} = x-1] \, dx \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[\text{noise} = x] = e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[\text{noise} \geq \theta - s_g(D_2)] \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[s_g(D_2) + \text{noise} \geq \theta]. $$

That is, we have

$$ \text{Pr}[n_s_g(D_1) \geq \theta] \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[n_s_g(D_2) \geq \theta]. \quad (4) $$

Moreover, we can also prove

$$ \text{Pr}[\text{noise} \geq \theta - s_g(D_1)] = \int_{\theta-s_g(D_1)}^{\infty} \text{Pr}[\text{noise} = x] \, dx \geq \int_{\theta-s_g(D_2)-1}^{\infty} \text{Pr}[\text{noise} = x] \, dx = \int_{\theta-s_g(D_2)}^{\infty} \text{Pr}[\text{noise} = x-1] \, dx \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[\text{noise} = x] = e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[\text{noise} \geq \theta - s_g(D_2)] \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[s_g(D_2) + \text{noise} \geq \theta]. $$

That is, we also have

$$ \text{Pr}[n_s_g(D_1) \geq \theta] \geq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[n_s_g(D_2) \geq \theta]. \quad (5) $$

In a similar way, we can see that

$$ \text{Pr}[n_s_g(D_1) < \theta] \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[n_s_g(D_2) < \theta], \quad (6) $$

and

$$ \text{Pr}[n_s_g(D_1) < \theta] \geq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[n_s_g(D_2) < \theta]. \quad (7) $$

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

$$ \text{Pr}[\text{Selecting subgraph } G \text{ from } C_i \text{ in } D_1] = \sum_{g \in C_i} 0 \times \text{Pr}[n_s_g(D_1) < \theta] + f(G, D_1) \times \text{Pr}[n_s_g(D_1) \geq \theta] = \sum_{g \in C_i} f(g, D_1) \times \text{Pr}[n_s_g(D_1) \geq \theta] \leq \sum_{g \in C_i} f(g, D_2) \times e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[n_s_g(D_2) \geq \theta] \leq e^{\frac{\epsilon_2 c}{2n_i}} \times f(G, D_2) \times e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[n_s_g(D_2) \geq \theta] \leq e^{\frac{2\epsilon_2 c}{2n_i}} \times f(G, D_2) \times \text{Pr}[n_s_g(D_2) \geq \theta] \leq 0 \times \text{Pr}[n_s_g(D_2) < \theta] + f(G, D_2) \times \text{Pr}[n_s_g(D_2) \geq \theta] \leq e^{\frac{\epsilon_2 c}{2n_i}} \text{Pr}[\text{Selecting subgraph } G \text{ from } C_i \text{ in } D_2]. $$

Based on the above analysis, we can see that, in our conditional exponential method, each subgraph selection guarantees $\epsilon_c$-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 $\epsilon_c$-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
from the first node in a frequent subgraph to say a graph
Algorithm 4 Lattice-based Noisy Support Derivation

to 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 (ND2) approach. The core of our ND2 approach is to leverage the inclusion relation between subgraphs [10] to reduce the amount of added noise. In our ND2 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 \( n_g1 \) to node \( n_g2 \) is introduced if graph \( g2 \) can be expanded to graph \( g1 \) 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 ND2 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 \( \epsilon_S \);

**Output:**
- A Map \( M \) of Frequent Subgraphs to Noisy Supports;

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

Alg. 4 shows the main steps of our ND2 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 ND2 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 \( g_1, g_2, \ldots, g_{|p|} \), where \( g_1 \subseteq g_2 \subseteq \cdots \subseteq g_{|p|} \). 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 \( D_{|p|} \) that includes the remaining input graphs. Then, we further divide sub-database \( D_{|p|} \) based on graph \( g_{|p|-1} \). This process continues until \( D \) is partitioned into \(|p|+1\) disjoint sub-databases \( D_T, D_1, \ldots, D_{|p|} \).

**Theorem 7** Our count accumulation method guarantees \( \epsilon_p \)-differential privacy.

**Proof:** In this method, we compute the noisy values of \( \alpha_{|p|}, \alpha_{|p|-1}, \ldots, \alpha_1 \). For \( i \) from 1 to \( |p| \), as an input graph can affect \( \alpha_i \) by at most one, the sensitivity of computing \( \alpha_i \) is one. Thus, adding Laplace noise \( \text{Lap}(1/\epsilon_p) \) to each \( \alpha_y \), where \( 1 \leq y \leq |p| \), satisfies \( \epsilon_p \)-differential privacy. Moreover, as sub-databases \( D_1, \ldots, D_{|p|} \) are mutually disjoint, by the parallel composition property [9], the privacy budgets used in computing \( \alpha_1, \alpha_2, \ldots, \alpha_{|p|} \) do not need to accumulate. For the graphs on path \( p \), as their noisy supports are obtained based on the noisy values of \( \alpha_1, \ldots, \alpha_{|p|} \), we can safely use them without privacy implications. Therefore, our count accumulation method satisfies \( \epsilon_p \)-differential privacy. \( \square \)
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 $\text{Lap}(|p|/\epsilon_p)$ 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 $\text{Lap}(1/\epsilon_p)$ to $\alpha$, which has variance $2/\epsilon_p^2$ for the $i$-th graph $g_i$, its noisy support $\text{sup}_i$ is equal to the sum of the noisy values of $\alpha_i$, $\alpha_{i+1}$, ..., $\alpha_{|p|}$. Since the noise added to $\alpha_i$, $\alpha_{i+1}$, ..., $\alpha_{|p|}$ is generated independently, the variance of $\text{sup}_i$ is the sum of the variances of $\alpha_i$, $\alpha_{i+1}$, ..., $\alpha_{|p|}$, 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 $\epsilon_p$. We allocate each path in $PS$ a privacy budget $\epsilon_3/\ell_t$. 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 $2|t^2(|p_m| - i + 1)/\epsilon_p^2$. 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 $\cup_{p \in PS} ns(p) = NS$, 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 $O(|NS|^3)$.

### Algorithm 5  Path Construction

**Input:**
- Lattice $L$;

**Output:**
- A Path Set $PS$;

```
1: PS ← ∅;
2: TN ← find all the nodes with in-degree 0 in L;
3: Sort nodes in TN in decreasing order based on their heights;
4: while $TN \neq ∅$ do
5:   $p ← B, n ←$ get a node from $TN$;
6:   Add node $n$ into path $p$;
7:   while true do
8:     childSet ← find the set of the child nodes of $n$;
9:     if childSet == ∅ then break;
10:   end if
11:   $n ←$ find the node with highest height from childSet;
12:   Add node $n$ into path $p$;
13: end while
14: Remove the nodes in $p$ from $L$;
15: Add new nodes with in-degree 0 into $TN$; Add $p$ into $PS$;
16: end while
17: 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 $n_e$ is one of its child nodes. If $n_p$ and $n_e$ are not connected after removing node $n$, a new edge from $n_p$ to $n_e$ is introduced. For example, in Fig. 4, after constructing path $p_1$, we remove node $n_{4}$, which causes nodes $n_{12}$ and $n_{15}$ not connected. In this case, a new edge from $n_{12}$ to $n_{15}$ 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.
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 p3 in Fig. 4, its first node n3 has two parent nodes n1 and n2. For node n1, it is already contained in path p1. Suppose g1 is the associated graph of n1. Based on path p1, we can obtain a noisy support sup1 of g1 with variance v1. Moreover, if we add n1 into path p3, we can utilize p3 to get another noisy support sup′ 1 for g1 with variance v′ 1. Then, we can aggregate these two noisy supports and get a new noisy support \( \frac{v_1 \times \text{sup} + v_1 \times \text{sup}'}{\text{v}_1 + \text{v}_1} \) with variance \( \frac{v_1}{\text{v}_1 + \text{v}_1} \). Clearly, \( \frac{v_1 \times \text{sup} + v_1 \times \text{sup}'}{\text{v}_1 + \text{v}_1} \) 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 n1 and n2 can be added into path p3. Let g1 and g2 be the associated graph of n1 and n2, and v1 and v2 be the variance of the noisy support of g1 and g2, respectively. After adding n1 or n2 into p3, we can utilize p3 to obtain another noisy support of g1 or g2. Suppose the variance of the new noisy support is v. In particular, if n1 is added into p3, we can update the variance of the noisy support of g1 to \( \frac{v_1 \times \text{sup} + v_1 \times \text{sup}'}{\text{v}_1 + \text{v}_1} \), while the variance of the noisy support of g2 is not changed. In contrast, if n2 is added into p3, the variance of the noisy support of g2 is updated to \( \frac{v_2 \times \text{sup} + v_2 \times \text{sup}'}{\text{v}_2 + \text{v}_2} \), while the variance of the noisy support of g1 is not changed. Suppose \( v_1 > v_2 \). Then,

\[
\text{v}_1 + \frac{v_2 \times v}{v_2 + v} - (\frac{v_1 \times v}{v_1 + v} + v_2) = \frac{v_1 \times v \times (v_1 - v_2) + (v_2^2 - v_2^2) \times v}{(v_1 + v) \times (v_2 + v)} > 0
\]

That is, \( \frac{\text{v}_1 \times \text{v} \times v_2}{\text{v}_1 + v_2} < v_1 + \frac{v_2 \times v}{v_2 + v} \). 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 \( \text{dec} \) 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 \( \text{inc} \) (line 10). If \( \text{dec} \) is no larger than \( \text{inc} \), 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: \( PS' \leftarrow \emptyset \);
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: \( n_f \leftarrow \) get the first node in \( p \);
6: parentSet \( \leftarrow \) find the set of all parent nodes of \( n_f \) in L;
7: while parentSet \( \neq \emptyset \) do
8: \( n_p \leftarrow \) find the node with largest attribute \( v \) from parentSet;
9: \( \text{dec} \leftarrow \) get expected decrement of \( n_p \’s \ attribute \( v_p \);
10: \( \text{inc} \leftarrow \) sum expected increment of attribute \( v \) of nodes in \( p \);
11: if \( \text{dec} \leq \text{inc} \) then break;
12: end if
13: Insert \( n_p \) at the first position of \( p \);
14: Update the attribute \( v \) of each node in \( p \);
15: parentSet \( \leftarrow \) find the set of all parent nodes of \( n_p \) in L;
16: end while
17: \( *** \) Iteratively Inserting Nodes at the Last Position \(* * * * *)
18: \( n_l \leftarrow \) get the last node in \( p \);
19: childSet \( \leftarrow \) find the set of all child nodes of \( n_l \) in L;
20: while childSet \( \neq \emptyset \) do
21: \( n_c \leftarrow \) find the node with largest attribute \( v \) from childSet;
22: Insert \( n_c \) at the last position of \( p \);
23: Update the attribute \( v_c \) of \( n_c \);
24: childSet \( \leftarrow \) find the set of all child nodes of \( n_c \) in L;
25: end while
26: Add \( p \) into \( PS' \);
27: return \( PS' \);

D. Privacy Analysis of ND2 Approach

In what follows, we give the privacy guarantee of our lattice-based noisy support derivation (ND2) approach (i.e., Alg. 4).
Theorem 8 Our ND₂ approach is $\epsilon_3$-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 $PS = \{p_1, p_2, ..., p_l\}$. We uniformly assign each path in $PS$ a privacy budget $\epsilon_3/l$. 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 $\epsilon_3/l$-differential privacy on each path. Based on the sequential composition property [9], it overall satisfies $\epsilon_3$-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 ND₂ approach is $\epsilon_3$-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 $\zeta_i$ is the maximal support of $i$-subgraphs (line 1). For a given threshold $\theta$, the maximal size $M_\theta$ is the number of elements in $\zeta$ larger than $\theta$. Thus, we can utilize our binary estimation method to estimate the number of elements in $\zeta$ larger than $\theta$, and set maximal size $M_\theta$ 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 $\theta$; Privacy Budgets $\epsilon_1, \epsilon_2, \epsilon_3$;
Output:
A Map $R$ of Frequent Subgraphs to Noisy Supports;
1: $\zeta \leftarrow$ the maximal support of subgraphs with different sizes;
2: $M_\theta \leftarrow$ the maximal frequent subgraph size based on $\zeta$ using $\epsilon_1$;
3: $F \leftarrow$ frequent_subgraph_identification ($D$, $\theta$, $\epsilon_2$, $M_\theta$);
4: $R \leftarrow$ lattice_based_noisy_support_derivation ($F$, $\epsilon_3$);
5: return $R$;

B. Privacy Analysis of DFG Algorithm

Theorem 9 Our DFG algorithm is $\epsilon$-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 $\epsilon_1$. According to Thm. 5, we can see that it satisfies $\epsilon_1$-differential privacy.

Then, we utilize our frequent subgraph identification ($FL_1$) 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 $\epsilon_i$-differential privacy. After that, we leverage our conditional exponential method to privately select frequent $i$-subgraphs. Based on Thm. 6, it guarantees $\epsilon_i$-differential privacy. For our $FL_1$ approach, by the sequential composition property [9], we can see that it satisfies $((\epsilon_1+\epsilon_2)M_g) = \epsilon_2$-differential privacy.

At last, we use our lattice-based noisy support derivation (ND₂) approach to compute the noisy support of identified frequent subgraphs. In Thm. 8, we show that our ND₂ approach satisfies $\epsilon_3$-differential privacy. In summary, by the sequential composition property [9], we can conclude our DFG algorithm achieves $(\epsilon_1+\epsilon_2+\epsilon_3) = \epsilon$-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 $\epsilon$-differential privacy. The second is the algorithm proposed in [4] (referred to as DFPM), which guarantees the weaker $(\epsilon, \delta)$-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 each 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 $\epsilon$ as follows: $\epsilon_1 = 0.1\epsilon$, $\epsilon_2 = 0.5\epsilon$, and $\epsilon_3 = 0.4\epsilon$. The default value of $\epsilon$ is 0.2. In Sec. IX-B, we also present the experimental results when $\epsilon$ is varied.

| TABLE I |
| SUMMARY OF GRAPH DATASETS |
| Dataset | #Graphs | Avg size | Max size | #Vertices |
| Cancer | 32597 | 283 | 236 | 67 |
| HIV | 42689 | 275 | 247 | 65 |
| SPL | 53804 | 475 | 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 $(\epsilon, \delta)$-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.
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 $\epsilon$ 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 $\epsilon$ increases. This is because, when $\epsilon$ increases, a smaller amount of noise is required and a lower degree of privacy is guaranteed.

C. Effect of FI Approach

In this experiment, we study how our frequent subgraph identification (FI) approach affects the performance of DFG on datasets Cancer and SPL. From Fig. 7, we can see, without using FI, 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.

D. Effect of ND Approach

We also study the effect of our lattice-based noisy support derivation (ND) 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 approach.
E. Benefit of FI₁ Approach

To better understand the benefit of our frequent subgraph identification (FI₁) 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 [28]. 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.

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.

**TABLE II**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Transactions</th>
<th>Items</th>
<th>Avg.length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pumsb-star</td>
<td>40046</td>
<td>2988</td>
<td>53.5</td>
</tr>
<tr>
<td>Accidents</td>
<td>340183</td>
<td>468</td>
<td>33.8</td>
</tr>
</tbody>
</table>

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**