Mining Frequent Graph Patterns with Differential Privacy

Entong Shen
North Carolina State University
eshen@ncsu.edu

Ting Yu
North Carolina State University
tyu@ncsu.edu

ABSTRACT

Discovering frequent graph patterns in a graph database offers valuable information in a variety of applications. However, if the graph dataset contains sensitive data of individuals such as mobile phone-call graphs and web-click graphs, releasing discovered frequent patterns may present a threat to the privacy of individuals. *Differential privacy* has recently emerged as the *de facto* standard for private data analysis due to its provable privacy guarantee. In this paper we propose the first differentially private algorithm for mining frequent graph patterns.

We first show that previous techniques on differentially private discovery of frequent *itemsets* cannot apply in mining frequent graph patterns due to the inherent complexity of handling structural information in graphs. We then address this challenge by proposing a Markov Chain Monte Carlo (MCMC) sampling based algorithm. Unlike previous work on frequent itemset mining, our techniques do not rely on the output of a non-private mining algorithm. Instead, we observe that both frequent graph pattern mining and the guarantee of differential privacy can be unified into an MCMC sampling framework. In addition, we establish the privacy and utility guarantee of our algorithm and propose an efficient neighboring pattern counting technique as well. Experimental results show that the proposed algorithm is able to output frequent patterns with good precision.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

Keywords

Differential privacy; graph pattern mining

1. INTRODUCTION

Frequent graph pattern mining (FPM) is an important topic in data mining research. It has been increasingly applied in a variety of application domains such as bioinformatics, cheminformatics and social network analysis. Given a graph dataset $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$, where each D_i is a graph, let $gid(G)$ be the set of IDs of graphs in $\mathcal{D}$ which contain G as a subgraph. G is a frequent pattern if its count $|gid(G)|$ (also called *support*) is no less than a user-specified *support threshold* f. Frequent subgraphs can help the discovery of common substructures, and are the building blocks of further analysis, including graph classification, clustering and indexing. For instance, discovering frequent patterns in social interaction graphs can be vital to understand functioning of the society or dissemination of diseases.

Meanwhile, publishing frequent graph patterns may impose potential threat to privacy, if the graph dataset contains private information of individuals. In many applications, each graph (rather than a node) is associated with an individual and may be sensitive. For example, the click stream during a browser session of a user is typically a sparse subgraph of the underlying web graph; in location-based services, a database may consist of a set of trajectories, each of which corresponds to the locations of an individual in a given period of time. Other scenarios of frequent pattern mining with sensitive graphs may include mobile phone call graphs [23] and XML representation of profiles of individuals. Therefore, extra care is needed when mining and releasing frequent patterns in these graphs to prevent leakage of private information of individuals.

Recently, the model of differential privacy [9] was proposed to restrict the inference of private information even in the presence of a strong adversary. It requires that the output of a differentially private algorithm is nearly identical (in a probabilistic sense), whether or not a participant contributes her data to the dataset. For the problem of frequent graph mining, it means that even an adversary who is able to actively influence the input graphs cannot infer whether a specific pattern exists in a target graph. Although tremendous progress has been made in processing *flat* data (e.g. relational and transactional data) in a differentially private manner, there has been very little work (discussed in Section 7) on differentially private analysis of graph data, due to the inherent complexity in handling the structural information in graphs.

In this paper we propose the first algorithm for privacy-preserving mining of frequent graph patterns that guarantees differential privacy. Recently several techniques [3, 17] have been proposed to publish frequent *itemsets* in a transactional database in a differentially private manner. It would seem attractive to adapt those techniques to address the problem of frequent *subgraph*[1] mining. Unfortunately, compared with private frequent itemset mining, the private FPM problem imposes much more challenges. First, graph datasets do not have a set of well-defined dimensions (i.e.,*items*), which is required by the techniques in [17]. Second, counting graph patterns is much more difficult than counting itemsets (due to graph

[1]We use 'graph pattern' and 'subgraph' interchangeably.

KDD' 13, August 11–14, 2013, Chicago, Illinois, USA.

isomorphism), which makes the size of the output space not immediately available in our problem. This prevents us from applying the techniques in [3]. We will explain the distinctions between [3, 17] and our work with more details in Section 2.3.

Contributions. The major contributions of this paper are summarized as follows:

1. For the first time, we introduce a differentially private algorithm for mining frequent patterns in a graph database. Our algorithm, called *Diff-FPM*, makes novel use of a Markov Chain Monte Carlo (MCMC) random walk method to bypass the roadblock of an output space with unknown size. This enables us to apply the *exponential mechanism*, which is an essential approach to achieving differential privacy.

2. Our approach provides provable privacy and utility guarantee on the output of our algorithm. We first show that our algorithm gives (ε, δ)-differential privacy, which is a relaxed version of ε-differential privacy. We then show that when the random walk has reached its steady state, *Diff-FPM* gives ε-differential privacy.

3. In order to propose a neighboring pattern more efficiently in MCMC sampling, we develop optimization techniques that significantly reduce the number of invocations to the subgraph isomorphism test subroutine. Experiment shows that our techniques can reduce the time to propose a neighboring pattern by an order of magnitude.

4. We conduct an extensive experimental study on the effectiveness and efficiency of our algorithm. With moderate amount of privacy budget, *Diff-FPM* is shown to output private frequent graph patterns with at least 80% precision.

2. PRELIMINARIES

2.1 Frequent Graph Pattern Mining

Frequent graph pattern mining (FPM) aims at discovering the subgraphs that frequently appear in a graph dataset. Formally, let $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ be a sensitive graph database which contains a multiset of graphs. Each graph $D_i \in \mathcal{D}$ has a unique identifier that corresponds to an individual. Let $G = (V, E)$ be a (sub)graph pattern, the graph identifier set $gid(G) = \{i : G \subseteq D_i \in \mathcal{D}\}$ includes all IDs of graphs in $\mathcal{D}$ that contain a subgraph isomorphic to G. We call $|gid(G)|$ the *support* of G in $\mathcal{D}$. The FPM algorithm can be defined either as returning all subgraph patterns whose supports are no less than a user-specified threshold f, or as returning the top k frequent patterns given an integer k as input. One can easily convert one version to the other. All graphs we consider in this paper are undirected, connected and labeled. Note that each node has a label and multiple nodes can have the same label.

2.2 Differential Privacy

Differential privacy [9] is a recent privacy model which provides strong privacy guarantee. Informally, a data mining or publishing procedure is differentially private if the outcome is insensitive to any particular record in the dataset. In the context of graph pattern mining, let $\mathcal{D}, \mathcal{D}'$ be two *neighboring datasets*, i.e., $\mathcal{D}$ and $\mathcal{D}'$ differ in only one graph (by adding or removing an individual), written as $||\mathcal{D} - \mathcal{D}'|| = 1$. Let $\mathcal{D}^n$ be the space of graph datasets containing n graphs.

DEFINITION 1 (ε-DIFFERENTIAL PRIVACY). *A randomized algorithm $\mathcal{A}$ is ε-differentially private if for all neighboring datasets $\mathcal{D},\mathcal{D}' \in \mathcal{D}^n$, and any set of possible output $\mathcal{O} \subset Range(\mathcal{A})$:*

$$\Pr[\mathcal{A}(\mathcal{D}) \in \mathcal{O}] \leq e^{\varepsilon} \Pr[\mathcal{A}(\mathcal{D}') \in \mathcal{O}].$$

The parameter $\varepsilon > 0$ allows us to control the level of privacy. A smaller ε suggests more limit posed on the influence of a single graph. Typically, the value of ε should be small ($\varepsilon < 1$). ε is usually specified by the data owner and referred as the *privacy budget*. In section 5.1 our discussion is related to a weaker notion called (ε, δ)-differential privacy [8], which allows a small additive error factor of δ.

DEFINITION 2 ((ε, δ)-DIFFERENTIAL PRIVACY). *A randomized algorithm $\mathcal{A}$ is (ε, δ)-differential private if for all neighboring datasets $\mathcal{D},\mathcal{D}' \in \mathcal{D}^n$, and any set of possible output $\mathcal{O} \subset Range(\mathcal{A})$:*

$$\Pr[\mathcal{A}(\mathcal{D}) \in \mathcal{O}] \leq e^{\varepsilon} \Pr[\mathcal{A}(\mathcal{D}') \in \mathcal{O}] + \delta.$$

A popular technique in applying differential privacy is the Laplace mechanism [9], which adds noise following Laplace distribution to the numeric output of a function. Applying the Laplace mechanism in our problem means adding noise to the support of all possible patterns and selecting the patterns with the highest noisy supports. However, this would be infeasible since it is computationally prohibitive to enumerate all possible patterns in any non-trivial sized graph mining problem.

Exponential Mechanism. A general technique of applying differential privacy is the exponential mechanism [20]. It not only supports non-numeric output but also captures the full class of differential privacy mechanisms. The exponential mechanism considers the whole *output space* and assumes that each possible output is associated with a real-valued utility score. By sampling from a distribution where the probability of the desired outputs are exponentially amplified, the exponential mechanism (approximately) finds the desired outputs while ensuring differential privacy.

Formally, given input space $\mathcal{D}^n$ and output space $\mathcal{X}$, a score function $u : \mathcal{D}^n \times \mathcal{X} \to \mathbb{R}$ assigns each possible output $x \in \mathcal{X}$ a score $u(\mathcal{D}, x)$ based on the input $\mathcal{D} \in \mathcal{D}^n$. The mechanism then draws a sample from the distribution on $\mathcal{X}$ which assigns each x a probability mass proportional to $\exp(\varepsilon u(\mathcal{D}, x)/2\Delta u)$, where $\Delta u = \max_{\forall x, \mathcal{D}, \mathcal{D}'} |u(\mathcal{D}, x) - u(\mathcal{D}', x)|$ is the sensitivity of the score function. Intuitively, the output with a higher score is exponentially more likely to be chosen. It is shown that this mechanism satisfies ε-differential privacy [20].

THEOREM 1. *[20] Given a utility score function $u : \mathcal{D}^n \times \mathcal{X} \to \mathbb{R}$ for a dataset $\mathcal{D}$, the mechanism $\mathcal{A}$,*

$$\mathcal{A}(\mathcal{D}, x) \triangleq \textit{return } x \textit{ with probability} \propto \exp(\frac{\varepsilon u(\mathcal{D}, x)}{2\Delta u})$$

gives ε-differential privacy.

The exponential mechanism has been shown to be a powerful technique in finding private medians [6], mining private frequent itemset [3, 17] and more generally adapting a deterministic algorithm to be differentially private [22]. Our *Diff-FPM* algorithm works by carefully applying the exponential mechanism. In this process we must overcome several critical challenges, which are identified next.

2.3 Challenges and Strategies

There has been work [3, 17] on mining frequent *itemsets* in a transaction dataset under differential privacy. However, the shift

from transactions to graphs poses significant new challenges. In [17], transaction datasets are viewed as high-dimensional tabular data, and the proposed approach projects the input database onto lower dimensions. However, graph datasets do not have a well defined set of *items*, i.e., dimensions, which renders the approach in [17] inapplicable in our FPM problem. In [3], two methods are proposed which make use of a notion of truncated frequency. However, those methods cannot be used in our problem due to the following fundamental challenges:

Support Counting. Obtaining the support of a graph pattern is much more difficult than counting itemsets. An itemset pattern can be represented by an ordered list or a bitmap of item IDs Checking the existence of an itemset in a transaction only takes $O(1)$ time, while checking whether a subgraph pattern exists in a graph is NP-complete due to subgraph isomorphism.

Unknown Output Space. The output space $\mathcal{X}$ in our problem contains a finite number of graph patterns which may or may not exist in the input dataset. Under differential privacy, any pattern in the output space should have non-zero probability to be in the final output. The probability of sampling a pattern x from the output space is

$$\pi(x) = \frac{\exp(\varepsilon u(x)/2\Delta u)}{C}, \tag{1}$$

where $C = \sum_{x\in\mathcal{X}} \exp(\varepsilon u(x)/2\Delta u)$ is the normalizing constant according to Theorem 1. The most straightforward way to compute C requires enumerating all the patterns in the output space. In [3], a technique is proposed to apply the exponential mechanism without enumerating if the size of the output space is known. However, unlike [3], in which the output space size can be obtained by simple combinatorics (i.e., $\binom{m}{l}$ patterns of size l given an alphabet of size m), *the size of the output space $\mathcal{X}$ in our problem is not immediately available (due to graph isomorphism[2])*, which prohibits us from applying exponential mechanism directly. Therefore we cannot apply the same techniques as in [3].

Given the analysis above, we need to develop new ways to overcome the issue of an unknown $|\mathcal{X}|$. Note that although the *global* information on the output space is not accessible, we do have the *local* information on any specific pattern – given any pattern x, we can immediately calculate its utility score $u(x)$. In addition, the unknown normalizing constant C is common to all patterns. That is, given any pair of patterns x_1, x_2, the ratio of probability mass $\pi(x_1)/\pi(x_2)$ is available without knowing the exact probabilities, according to Eq.(1). Such scenarios, where one needs to draw samples from a probability distribution known up to a constant factor, also arise in statistical physics when analyzing dynamic systems, where Markov Chain Monte Carlo (MCMC) methods are often used. Inspired by that, our idea is to perform a random walk based on locally computed probabilities. By carefully choosing the neighbor and the probability of moving in each step using the Metropolis-Hastings (MH) method [24], the random walk will converge to the target distribution, from which we can output samples. Next we discuss the details of our *Diff-FPM* algorithm.

3. PRIVATE FPM ALGORITHM

3.1 Overview

The key challenge of handling graph datasets is the unknown output space when applying the exponential mechanism. The *Diff-FPM* algorithm meets the challenge by unifying frequent pattern mining and applying differential privacy into an MCMC sampling framework. The main idea of *Diff-FPM* is to simulate a Markov chain by performing an MCMC random walk in the output space. Our goal is that when the random walk reaches its steady state, the stationary distribution of the Markov chain matches the target distribution π in Eq.(1). In Section 3.2.1 we will explain in detail how to apply the Metropolis-Hastings (MH) method in our problem to achieve this goal. Before that, we need to define the state space in which we perform the random walk.

Partial Order Full Graph. To facilitate the MH-based random walk in the output space, we define the Partial Order Full Graph (POFG) as the state space of the Markov chain on which the sampling algorithm run the simulation. Each node in POFG corresponds to a unique graph pattern and each edge in POFG represents a possible 'extension' (add or remove one edge) to a neighboring pattern. Naturally, each node in the POFG has three types of neighbors: *sub-neighbor* (by removing an edge), *super-backward neighbor* (by connecting two existing nodes) and *super-forward neighbor* (by adding and connecting to a new node).

EXAMPLE 1. *Figure 1 shows a simple graph dataset containing 3 graphs and its POFG. The dashed patterns have support smaller than 2 in the dataset. Pattern $A-A-C$ has two sub-neighbors, one super-backward neighbor and several super-forward neighbors (only one shown in Figure 1(b)). Self-loops and multi-edges are not considered in this example and thus are excluded from the output space.*

At a higher level, the random walk starts with an arbitrary pattern and proceeds to an adjacent pattern with certain probability in each step. Since the transition decision is made solely based on local information (related to the neighborhood of the current pattern), there is no need to construct the global POFG explicitly. When the random walk has reached its steady state, the probability of being in state x follows exactly the target distribution $\pi(x)$ in Eq.(1). Then the current state is drawn as a sampled pattern. Since the frequent patterns have larger probabilities in the target distribution, they are more likely to appear in the final output.

3.2 Detailed Descriptions

3.2.1 The Diff-FPM Algorithm

The core of the *Diff-FPM* algorithm is a careful application of the MH method. The MH method is a Markov Chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a target probability distribution for which direct sampling is difficult. It only requires that a function proportional to the probability mass be calculable.

Suppose we want to generate a random variable X taking values in $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$, according to a target distribution π, with

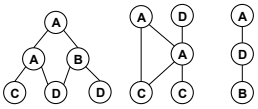
$$\pi(x_i) = \frac{b(x_i)}{C}, \qquad x_i \in \mathcal{X}$$

where all $b(x_i)$ are strictly positive, $|\mathcal{X}|$ is large, and the normalizing constant $C = \sum_{i=1}^{|\mathcal{X}|} b(x_i)$ is difficult to calculate. The MH method first constructs an $|\mathcal{X}|$-state Markov chain $\{X_t, t = 0, 1, \ldots\}$ on $\mathcal{X}$ whose evolution relies on an arbitrary proposal transition matrix $Q = \big(q(x, y)\big)$ in the following way:

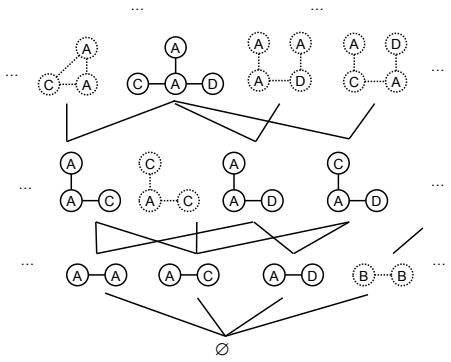
1. When $X_t = x$, generate a random variable Y satisfying $P(Y = y) = q(x, y)$, $y \in \mathcal{X}$
2. Given $Y = y$, let

$$X_{t+1} = \begin{cases} y & \text{with probability } \alpha_{xy}, \\ x & \text{with probability } 1 - \alpha_{xy}, \end{cases}$$

[2]A detailed analysis on the size of the output space can be found in the full version [26].



(a) Graph database with 3 graphs



(b) Part of POFG of Figure 1(a)

Figure 1: Example graph database and POFG

where $\alpha_{xy} = \min\left\{\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right\} = \min\left\{\frac{b(y)q(y,x)}{b(x)q(x,y)}, 1\right\}$. It means that given a current state x, the next state is proposed according to the proposal distribution Q. $q(x, y)$ is the probability mass of state y among all possible states given the current state is x. With probability α_{xy}, the proposal is accepted and the chain moves to the new state y. Otherwise it remains at state x. It follows that $\{X_t, t = 0, 1, \dots\}$ has a one-step transition probability matrix P:

$$P(x,y) = \begin{cases} q(x,y)\alpha_{xy}, & \text{if } x \neq y \\ 1 - \sum_{z \neq x} q(x,z)\alpha_{xz}, & \text{if } x = y \end{cases}$$

It can be shown that for the above P, the Markov chain is reversible and has a stationary distribution π, equal to the target distribution. Therefore, once the chain has reached the steady state, the sequence of samples we get from the MH method should follow the target distribution.

EXAMPLE 2. *Consider a random walk on the POFG illustrated in Figure 1(b). Suppose the current state of the walk is 'A-A-D' (pattern x). Following the MH method, one of pattern x's neighbors needs to be proposed according to a proposal distribution $q(x, y)$. For simplicity, in this example each neighbor has an equal probability to be proposed, i.e., $q(x, y) = 1/|N(x)|$, where $N(x)$ is the neighbor set of x. Assuming 'A-D' (pattern y) is proposed and $|N(x)| = 5$, $|N(y)| = 10$, $b(\cdot) = \exp(|gid(\cdot)|/2)$, the probability of accepting the proposal is calculated as $\alpha_{xy} = \min\{\frac{\exp(3/2)\cdot(1/10)}{\exp(2/2)\cdot(1/5)}, 1\} = 0.82$. We can then draw a random number between 0 and 1 to decide whether walking to pattern y or staying at x.*

The description of the *Diff-FPM* algorithm above can be summarized in Algorithm 1. The input consists of the raw graph dataset $\mathcal{D}$, a support threshold f and the privacy budget $\varepsilon = \varepsilon_1 + \varepsilon_2$. If the top-$k$ frequent patterns are desired, we first run non-private FPM algorithms such as gSpan [29] to get the support threshold f, i.e., the support of the kth frequent pattern. If one only needs k patterns whose supports are no less than a threshold, f can be directly provided to the algorithm. At a higher level, Algorithm 1 consists

Algorithm 1: *Diff-FPM* algorithm

input : Graph dataset $\mathcal{D}$, threshold f, privacy budget $\varepsilon_1, \varepsilon_2$
output: A set S of k private frequent patterns

1 **for** $i = 1$ *to* k **do**
2 Choose any pattern in the output space as seed pattern;
3 **while** *True* **do**
4 Propose a neighboring pattern y of current pattern x according to the proposal distribution (Eq. 2);
5 Accept the proposed pattern with probability $\alpha_{xy} = \min\left\{\frac{\exp(\varepsilon_1 u(y)/2k\Delta u)q_{yx}}{\exp(\varepsilon_1 u(x)/2k\Delta u)q_{xy}}, 1\right\}$;
6 **if** *convergence conditions are met* **then**
7 Add current pattern to S and remove it from the output space;
8 **break**;
9 (Optional) for each pattern in S, perturb its true support by Laplace mechanism with privacy budget ε_2/k;

of two phases: sampling and perturbation. The sampling phase includes k applications of the exponential mechanism via MH-based random walk in the output space.

Initially, we select an arbitrary pattern in the output space to start the walk (Line 2). At each step, we propose a neighboring pattern y of the current pattern x according to a *proposal distribution* (Line 4). The proposal distribution does not affect the correctness of the MH method, so we defer the details to Section 3.2.3. The proposed pattern is then accepted with probability α_{xy} as in the MH-algorithm (Line 5), where $u(\cdot)$ is the score function with Δu being the sensitivity of $u(\cdot)$. We explore the design space of the score function in the next paragraph. When the Markov chain has converged (see Section 3.3 for convergence diagnostic), we output the current pattern and remove it from the output space (Line 6 to 8). We then start a new walk until k patterns have been sampled. Finally, if one wants to include the support of each output pattern as well, the count of each pattern is perturbed by adding $Lap(k/\varepsilon_2)$ noise (Line 9).

3.2.2 Score Function Design

Choosing the utility score function is vital in our approach as it directly affects the target distribution. A general guideline is that the patterns with higher supports should have higher utility scores in order to have larger probabilities to be chosen according to exponential mechanism. Under this guideline, given an input database $\mathcal{D}$, the most straightforward choice is to let $u(x, \mathcal{D}) = |gid(x)|$ for any pattern x. In this case, the sensitivity Δu is exactly 1 since the support of any subgraph pattern may vary by at most 1 with the addition or removal of a graph in the dataset. This is also the score function we use in the experiment.

3.2.3 Proposal Distribution

Although in theory the proposal distribution can be arbitrary, it can significantly impact the efficiency of the MH method by affecting the mixing time (time to reach steady state). A good proposal distribution can improve the convergence speed by increasing the accept rate α_{xy} in the MH method. On the contrary, if the proposed pattern is often rejected, the chain can hardly move forward. It has been suggested that one should choose a proposal distribution close to the target distribution [11]. In our problem setting, it is preferable to make a distinction between the patterns having support no less than f (referred as *frequent* patterns) and those whose supports are lower (referred as *infrequent* patterns). Given a current state x, we denote the set of *frequent* neighbors of x as $N_1(x)$ and the set of

infrequent neighbors as $N_2(x)$. Since $|N_2(x)|$ is usually larger than $|N_1(x)|$, we will balance the probability mass assigned to $N_1(x)$ and $N_2(x)$ by introducing a tunable parameter η $(0 < \eta < 1)$. Our heuristic based proposal distribution is formally described below:

$$Q(x,y) = \begin{cases} \eta \times \frac{1}{|N_1(x)|}, & \text{if } y \in N_1(x) \\ (1-\eta) \times \frac{1}{|N_2(x)|}, & \text{if } y \in N_2(x) \end{cases} \quad (2)$$

In the experiment we use $\eta > 0.5$ such that a frequent pattern has a higher probability to be proposed than an infrequent pattern. If any of $N_1(x)$ or $N_2(x)$ is empty, its probability mass will be re-distributed (by setting $\eta = 0$ or $\eta = 1$ respectively). Note that the choice of the proposal distribution does not impact the privacy and utility guarantee of *Diff-FPM*.

3.2.4 Pattern Removal

In line 6 to 8 of Algorithm 1, after the convergence conditions are met and a sample pattern g is outputted, we need to exclude g from the output space by connecting g's neighbors and removing g in the POFG. In our implementation this is done by replacing g by all the neighbors of g whenever g appears in some pattern's neighborhood. Note that we do not output multiple patterns when the chain has converged. This is because once a pattern is sampled, it should be excluded from the output space and thus have zero probability to be chosen. Therefore adjustment to the output space is necessary after each sample. For the same reason we do not run multiple chains at once.

3.3 Convergence Diagnostics

The theory of MCMC sampling requires that samples are drawn when the Markov chain has converged to the stationary distribution, which is also our target distribution π. The most straightforward way to diagnose convergence is to monitor the distance between the target distribution π and the distribution of samples $\hat{\pi}$. In practice, however, π is often known only up to a constant factor. To deal with this problem, several online diagnostic tests have been developed in the MCMC literature [11] and used in random walk based sampling of graphs [12].

Online diagnostics rely on detecting whether the chain has lost its dependence on the starting point. We adopt a standard convergence test called the Geweke diagnostic [10]. The Geweke diagnostic takes two non-overlapping parts (usually the first 0.1 and last 0.5 proportions) of the Markov chain and see if they are from the same distribution. Specifically, let X be a sequence of samples of our metric of interest and X_1, X_2 be the two non-overlapping subsequences. Geweke computes the Z-score: $Z = \frac{E(X_1)-E(X_2)}{\sqrt{Var(X_1)+Var(X_2)}}$. With increasing number of iterations, X_1 and X_2 should move further apart and become less and less correlated. When the chain has converged, X_1 and X_2 should be identically distributed with $Z \sim N(0,1)$ by law of large numbers. We can declare convergence when Z has continuously fallen in the $[-1,1]$ range. Since the samples in our problem are graph patterns rather than a scalar, we may need to monitor multiple scalar metrics related to different properties of the sampled pattern and declare convergence when all these metrics have converged.

We need to acknowledge that these convergence diagnostic tools from the MCMC literature are heuristic *per se*. Verifying the convergence remains an open problem if the distribution of samples is not directly observable. Even so, *Diff-FPM* still achieves (ε, δ)-differential privacy if there exists a small distance between the target and simulation distributions, as we will show in Lemma 2 in Section 5.

Algorithm 2: The EEN algorithm

input : Pattern x, graph dataset $\mathcal{D}$, support threshold f
output: $N_1(x)$, $N_2(x)$
1 Initialize N_1,$N_2 \leftarrow \emptyset$ (x omitted for brevity);
2 Find membership bitmap B_x using VF2 isomorphism test;
3 Populate sub-neighbors N^b, super-back neighbors N^p_{back}, super-forward neighbors N^p_{fwd};
/* Explore sub-neighbors N^b */
4 **if** $sum(B_x) \geq f$ **then** $N_1 \leftarrow N_1 \cup N^b$;
5 **else for** $x' \in N^b$ **do**
6 | **if** SUB_IS_FREQ (x', B_x) **then** $N_1 \leftarrow N_1 \cup \{x'\}$;
7 | **else** $N_2 \leftarrow N_2 \cup \{x'\}$;
/* Explore super-back neighbors N^p_{back} */
8 **if** $sum(B_x) < f$ **then** $N_2 \leftarrow N_2 \cup N^p_{back}$;
9 **else**
10 | $\forall x' \in N^p_{back}$, initialize dictionary $H[x'] = 0$;
11 | **for** $i \leftarrow 1$ *to* $|\mathcal{D}|$ **do**
12 | | Find set $\mathcal{M}$ of all mappings between D_i and x;
13 | | **for** $x' \in N^p_{back}$ **do**
14 | | | **if** $H[x'] < f$ *and* $|\mathcal{D}| - i + H[x'] \geq f$ **then**
15 | | | | Let (u,v) be the back edge, i.e., $x = x' \diamond (u,v)$;
16 | | | | **for** $m \in \mathcal{M}$ **do**
17 | | | | | **if** $m(u), m(v)$ *are adjacent in* D_i **then**
18 | | | | | | $H[x'] \leftarrow H[x'] + 1$;
19 | | | | | | break;
20 | **for** $x' \in N^p_{back}$ **do**
21 | | **if** $H[x'] \geq f$ **then** $N_1 \leftarrow N_1 \cup \{x'\}$;
22 | | **else** $N_2 \leftarrow N_2 \cup \{x'\}$;
23 Explore super-forward neighbors N^p_{fwd} similarly as N^p_{back}, details in [26];
24 **return** N_1, N_2;

4. EFFICIENT EXPLORATION OF NEIGHBORS (EEN)

We have discussed so far the core of the *Diff-FPM* algorithm and seemingly it could be run straightforwardly. However, without certain optimization, the computation cost might render the algorithm impractical to run. The most costly operation in the *Diff-FPM* algorithm is proposing a neighbor of the current pattern x. According to the proposal distribution in Eq.2, this requires knowledge on the support of each pattern in x's neighbors $N(x)$. Due to the fact that subgraph isomorphism test is NP-complete, obtaining the support of each neighbor might become a computation bottleneck.

To overcome this problem, we have developed an efficient algorithm (called EEN) to explore the neighborhood of a pattern by observing the connection between neighboring patterns and their isomorphic mappings.

4.1 The EEN Algorithm

The task of neighbors exploration can be described as: given a pattern x, find the set of frequent neighbors $N_1(x)$ and infrequent neighbors $N_2(x)$, as in the proposal distribution (Eq.2). A naive way to populate $N_1(x)$ and $N_2(x)$ is to test each neighbor of x against the graph dataset $\mathcal{D}$. However, this is extremely inefficient since $|N(x)| \cdot |\mathcal{D}|$ isomorphism tests are required, where $|\mathcal{D}|$ is the number of graphs in $\mathcal{D}$. A basic optimization would be using the monotonic property of frequent patterns: if x is a frequent pattern, any subgraph of x should be frequent too; likewise, an infrequent

pattern's super-graph must be infrequent. However, explicit isomorphism testing is still required for exploring the super-neighbors of x if x is frequent or x's sub-neighbors if x is infrequent.

The EEN algorithm is able to further reduce the number of isomorphism tests. Observing that x and y only differ in one edge for all $y \in N(x)$, the main idea of is to re-use the isomorphic mappings between x and $D_i \in \mathcal{D}$ and examine whether any of the isomorphic mappings can be retained after extending an edge. The EEN algorithm is formally presented in Algorithm 2 and is described in the following.

Algorithm 2 takes pattern x, graph dataset $\mathcal{D}$ and support threshold f as input and returns $N_1(x)$ and $N_2(x)$. First, pattern x is tested against each graph in $\mathcal{D}$ and the result is stored in $B_x = \{i | x \subseteq D_i, D_i \in \mathcal{D}\}$, which is the set of IDs of graphs containing pattern x (line 2). The subgraph isomorphism algorithm we use is the VF2 algorithm [5]. Next we populate three types of neighbors of x: sub-neighbors N^b, super-back neighbors N^p_{back} and super-forward neighbors N^p_{fwd} (line 3), and handle them differently.

Explore sub-neighbors (line 4 to 7). For N^b , if x is frequent, the entire set N^b should be frequent. If x is infrequent, each pattern in N^b is examined by the boolean sub-procedure SUB_IS_FREQ. SUB_IS_FREQ takes a sub-neighbor x' of x and B_x as input and returns whether x' is frequent. First we find $B_E = \bigcap_{e \in x'} B_e$, the intersection of ID sets of all edges in pattern x'. Then subgraph isomorphism test is only needed for the graphs $D_i \in B_E \backslash B_x$. The set of IDs of graphs that succeed the test together with B_x comprise $B_{x'}$. Finally the procedure returns the frequentness of x' by comparing f and the size of $B_{x'}$.

Explore super-back neighbors (line 8 to 22). For N^p_{back}, if x is infrequent, the entire N^p_{back} must be infrequent. Otherwise, we test whether $x' \in N^p_{back}$ is a subgraph of D_i for each D_i. In this part, the EEN algorithm does not require any additional subgraph isomorphism test at all. This is achieved by re-using the isomorphism mappings between the base pattern x and D_i and reasoning upon that. In line 12 we find all the subgraph isomorphism mappings $\mathcal{M} : V^n_x \rightarrow V^n_{D_i}$, which can be obtained at the same time when computing B_x in line 2 as part of the VF2 algorithm. Note that the subgraph isomorphism package we use is complete, i.e., it can return all the mappings. Suppose x is extended to x' by connecting node u and v (line 15). If any of the isomorphism mappings $m \in \mathcal{M}$ is preserved with the edge extension (i.e., $m(u)$ and $m(v)$ are adjacent in D_i), x' must be a subgraph of D_i. Otherwise if none of the mappings can be preserved, x' is not a subgraph of D_i.

In the above process, we use a dictionary H to keep track of the number of graphs in $\mathcal{D}$ so far that contains x' as a subgraph, i.e., $H[x']$ maintains $|\{D_i | x' \subseteq D_i\}|$ for the D_i tested so far. Line 14 ensures that the isomorphism extension test is only performed when $H[x']$ has not reached f.

Explore super-forward neighbors. For N^p_{fwd}, the algorithm is similar to the procedures of exploring super-back neighbors, except that the extension test is now on a forward edge instead of a back edge. The details are available in [26] due to space limit.

5. PRIVACY AND UTILITY ANALYSIS

The proof of the lemmas and theorems in this section can be found in [26].

5.1 Privacy Analysis

In this part we establish the privacy guarantee of *Diff-FPM*. We show both the sampling and perturbation phases preserve privacy, and then we use the composition property of differential privacy to show the privacy guarantee of the overall algorithm.

In the sampling phase, our target probability distribution $\pi(\mathcal{D}, \cdot)$ equals $\frac{\exp(\varepsilon_1 u(\mathcal{D}, \cdot)/2k\Delta u)}{C}$ for a given dataset $\mathcal{D}$. If samples were drawn directly from this distribution, it would achieve strict $\frac{\varepsilon_1}{k}$-differential privacy due to the exponential mechanism. Since we use MCMC based sampling, the distribution of the samples $\hat{\pi}(\mathcal{D}, \cdot)$ will *approximate* $\pi(\mathcal{D}, \cdot)$, i.e. the two distributions are *asymptotically* identical. In real simulation, there may be a small distance between the two distributions. To quantify the impact on privacy when a small error is present, we use the *total variation distance* [24] to measure the distance of the two distributions at a given time:

$$||\hat{\pi}(\cdot) - \pi(\cdot)||_{TV} \equiv \max_{T \subset \mathcal{X}} |\hat{\pi}(T) - \pi(T)| \tag{3}$$

which is the largest possible difference between the probabilities that $\pi(\cdot)$ and $\hat{\pi}(\cdot)$ can assign to the same event.

Let $\mathcal{A}(\mathcal{D})$ denote the process of sampling one pattern according to Algorithm 1 (Line 4 to 8). The privacy guarantee that $\mathcal{A}(\mathcal{D})$ offers is described by the following lemma:

LEMMA 2. *Let $\pi(\cdot)$ and $\hat{\pi}(\cdot)$ denote the target distribution and the distribution of samples from $\mathcal{A}(\mathcal{D})$ respectively. Suppose $||\hat{\pi}(\cdot) - \pi(\cdot)||_{TV} \leq \theta$, procedure $\mathcal{A}(\mathcal{D})$ gives $(\frac{\varepsilon_1}{k}, \delta)$-differential privacy, where $\delta = \theta(1 + e^{\varepsilon_1/k})$.*

Note that θ is a function of simulation time t. The following lemma describes the asymptotic behavior and the speed of convergence of the chain :

LEMMA 3. *[24] If a Markov chain on a finite state space is irreducible and aperiodic, and has a transition kernel P and stationary distribution $\pi(\cdot)$, then for $x \in \mathcal{X}$,*

$$||P^t(x, \cdot) - \pi(\cdot)||_{TV} \leq M\rho^t, \qquad t = 1, 2, 3, \ldots \tag{4}$$

for some $\rho < 1$ and $M < \infty$. And

$$\lim_{t \to \infty} ||P^t(x, \cdot) - \pi(\cdot)||_{TV} = 0 \tag{5}$$

It means θ is decreasing at least at a geometric speed and approximates to zero when the simulation is running long enough.

Since the sampling process in Algorithm 1 consists of k successive applications of exponential mechanism based on random walk, we need the following well-known composition lemma to provide privacy guarantee for the entire sampling phase.

LEMMA 4. *[19] Let $\mathcal{A}_1, \ldots, \mathcal{A}_t$ be t algorithms such that $\mathcal{A}_i$ satisfies ε_i-differential privacy, $1 \leq i \leq t$. Then their sequential composition $\langle \mathcal{A}_1, \ldots, \mathcal{A}_t \rangle$ satisfies ε-differential privacy, for $\varepsilon = \sum_{i=1}^{t} \varepsilon_i$.*

Equipped with the results in previous lemmas, we are able to provide the privacy guarantee for Algorithm 1.

THEOREM 5. *Algorithm 1 satisfies ε-differential privacy.*

5.2 Utility Analysis

Because neighboring inputs must have similar output under differential privacy, a private algorithm usually does not return the exact answers. In the scenario of mining top-k frequent patterns, the *Diff-FPM* algorithm returns a noisy list of patterns which is close to the real top-k patterns. To quantify the quality of the output of *Diff-FPM*, we first define two utility parameters, following [3]. Recall that f is the support of the kth frequent pattern, and let β be an additive error to f. Given $0 < \gamma < 1$, we require that with probability at least $1 - \gamma$, (1) no pattern in the output has true

support less than $f - \beta$ and (2) all patterns having support greater than $f + \beta$ exist in the output. The following theorems provide the utility guarantee of *Diff-FPM*. A score function $u(x) = |gid(x)|$ is assumed.

THEOREM 6. *At the end of the sampling phase in Algorithm 1, for all $0 < \gamma < 1$, with probability at least $1-\gamma$, all patterns in set S have support greater than $f-\beta$, where $\beta = \frac{2k}{\varepsilon_1}(\ln(k/\gamma)+\ln M)$ and M is an upper bound on the size of output space.*

The following theorem provides the upper bound of noise added to the true support of each output pattern.

THEOREM 7. *For all $0 < \gamma < 1$, with probability of at least $1 - \gamma$, the noisy support of a pattern differs by at most β, where $\beta = \frac{k}{\varepsilon_2}\ln(1/\gamma)$.*

6. EXPERIMENTAL STUDY

In this section, we evaluate the performance of *Diff-FPM* through extensive experiments on various datasets. Since this is the first work on differentially private mining of frequent graph patterns, the quality of the output is compared with the result from a non-private FPM algorithm and the accuracy is reported. In this section we consider the scenario of mining the top-k frequent patterns.

6.1 Experiment Setup

Datasets. The following three datasets are used in our experiment: *DTP* is a real dataset containing DTP AIDS antiviral screening dataset[3], which is frequently used in frequent graph pattern mining study. It contains 1084 graphs, with an average graph size of 45 edges and 43 vertices. There are 14 unique node labels and all edges are considered having the same label.

The *click* dataset consists of 20K small tree graphs (4 nodes and 3 edges on average) obtained by a graph generator developed by Zaki [30]. To a certain extent, this synthetic dataset simulates user click graphs from web server logs [30], which is a suitable type of data requiring privacy-preserving mining. All the tree graphs in this dataset are sampled from a master tree.

The above two datasets contain graphs that are relatively sparse. To test our algorithm on *dense* graphs, we also use a dataset containing 5K graphs, in which the average node degree is 7. Each graph contains 10 vertices and 35 edges on average. The graph generator [4] we use is specially designed for generating graph datasets for evaluation of frequent subgraph mining algorithms. The size of this graph dataset is comparable to the largest datasets used in previous works [29, 15].

Utility metrics. We evaluate the quality of the output of *Diff-FPM* by employing the following three utility metrics: *Precision*, *Support Accuracy* and *nDCG*[4]. Precision is defined as the fraction of identified top-k graph patterns that are in the true top-k, i.e., $Precision = |\text{True Positives}|/k$. This is the complementary measure of the false negative rate used in [3]. The true top-k patterns are obtained by a non-private graph mining algorithm (gSpan [29] in our experiment). The measure of precision reflects the percentage of desired/undesired patterns in the output, yet it cannot indicate how good or bad the output patterns are in terms of their supports. For example, if $f = 1000$, it is much more undesirable if a pattern with support 10 appears in the output compared to a pattern with support 980, even though the precision may be the same in these two cases. We first define the *relative support error (RSE)* as $RSE = (S_{true} - S_{out})/kf$, where S_{true} and S_{out} are the sum of the supports of the real top-k patterns and sum of the supports of the sampled patterns respectively. This measure reflects the average deviation of an output pattern's support with respect to the support threshold f. In the plots, the *support accuracy* is reported, which equals $1 - RSE$. nDCG is a commonly used metric to compare two ranked lists. This metric is accumulated from the top of the result list to the bottom with the weight of each result discounted at lower ranks. In our problem setting, the top-k patterns are un-ordered. Still, nDCG is able to reveal whether any important pattern is missing in the output.

All experiments were conducted on a PC with 3.40GHz CPU with 8GB RAM. The random walk in the *Diff-FPM* algorithm has a small memory footage due to its Markovian nature. We implemented our algorithm in Python 2.7 with the JIT compiler PyPy[5] to speed up. The default parameters of $\varepsilon = 0.5$, $\eta = 0.8$ and $k = 15$ were used unless specified otherwise. In the experiment we do not release the noisy supports of the patterns in the output (line 9 in Algorithm 1), so all the privacy budget is used in the sampling phase.

6.2 Experiment Results

Comparison of neighbor exploration methods. In Section 4.1 we proposed the EEN algorithm to efficiently explore the neighborhood of a pattern. We now compare it with two other methods: a *naive* approach which finds the support of each neighbor of the current pattern x and a *basic* approach which uses the monotonic property of frequent patterns (see Section 4.1). Figure 2(a) shows the average iteration time in logarithm of the three methods over three datasets. In each iteration, a neighboring pattern is proposed and then accepted or rejected according to the MH algorithm. Clearly, EEN takes significantly less time in each iteration than the other methods in both datasets, reducing the iteration time by at least an order of magnitude compared to the naive approach. Thus all subsequent results are presented with EEN enabled.

Run time and scalability. Figure 2(d) illustrates the average time taken to output one frequent pattern as the size of the dataset increases. For the full datasets, *click* takes 20 seconds, *DTP* takes about 1 minute and *dense* sits in the middle, although the *click* dataset contains 20K graphs compared to only 1K in the *DTP*. It indicates that the size of each individual graph and the size of the neighborhood have a larger impact on the run time than the total number of graphs in the dataset (note that *DTP* has 14 labels and thus a larger neighborhood of a pattern compared to *dense*). For scalability, all datasets are observed to have linear scale-up in time as the size of graph dataset increases.

Utility result. To test the quality of the output by *Diff-FPM*, we examine the utility metrics introduced above under various parameter settings.

First, Figure 2(b) and Figure 2(c) show the precision and *SA* when we increase the size of the graph dataset from 10% to 100% [6]. An increasing trend of the output quality can be clearly observed here. This is in line with our expectation because achieving differential privacy is more demanding in a small dataset – the larger the number of records in the database, the easier it is to hide an individual record's impact on the output. For all three full datasets, *Diff-FPM* is able to achieve at least 80% on both precision and *SA*.

Figure 3(a) shows the precision when varying privacy budget ε. With a very limited budget ($\varepsilon = 0.1$), only about 30% of samples

[3]http://dtp.nci.nih.gov/docs/aids/aids_data.html

[4]http://en.wikipedia.org/wiki/Discounted_cumulative_gain

[5]http://pypy.org

[6]The data point for *dense* at 10% is absent since the smallest dataset size can be generated is 1K.

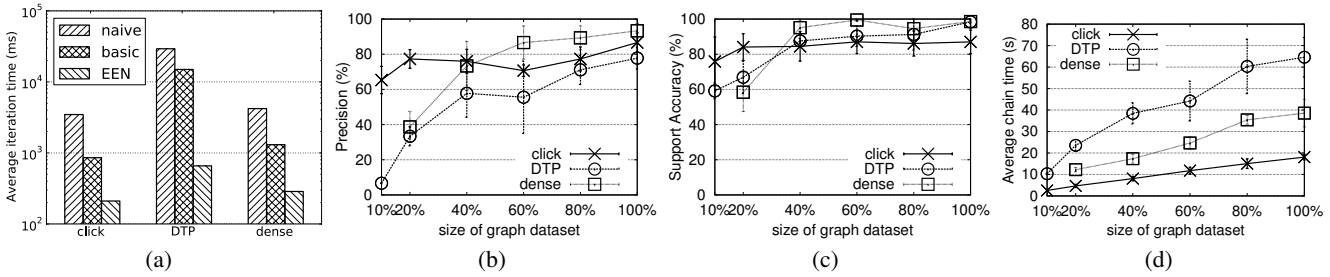


Figure 2: Effectiveness of EEN and impact of graph dataset size

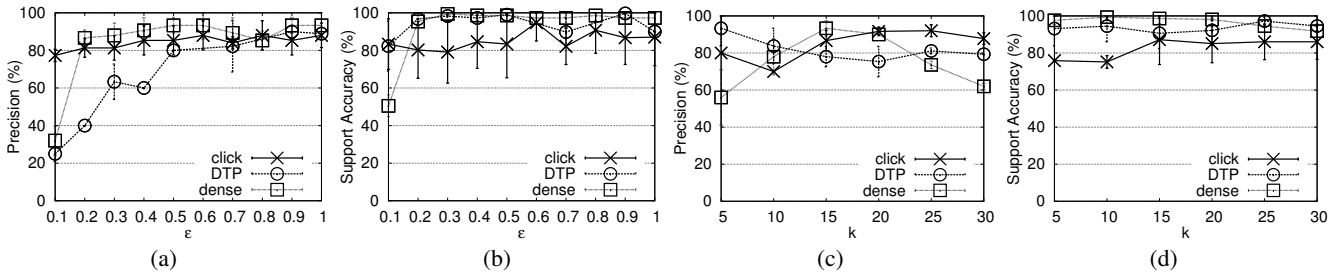


Figure 3: Precision and accuracy versus ε and k

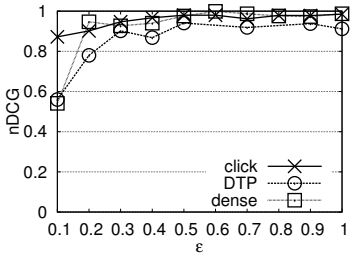


Figure 4: nDCG versus ε

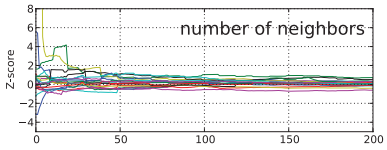


Figure 5: Convergence trace of 20 chains

are from the real top-k patterns for *DTP* and *dense*. This is inevitable due to the privacy-utility tradeoff. As more privacy budget is given, the precision of *Diff-FPM* increases fast. At $\varepsilon = 0.5$, the precisions from all datasets have reached 80%. Further increase in privacy budget does not provide significant benefit on the precision. We observed a similar trend in the support accuracy plot (Figure 3(b)), with less dramatic changes for ε from 0.1 to 0.5.

Figures 3(c) and 3(d) illustrate the impact of the number of patterns in the output. Recall that in each round of sampling, a budget of ε/k is consumed (cf. proof of Theorem 5). Given a certain privacy budget, the more patterns to output, the less privacy budget each sample can use. Thus we expect the average quality of the output to drop as k increases, which is confirmed in the result. Meanwhile, the support accuracy of the output holds well with the increasing number of output, which can be seen in Figure 3(d).

We also report the nDCG of the output with respect to different privacy levels in Figure 4. It can be seen that given moderate amount of privacy budget, the nDCG of the output remains larger than 0.8, suggesting close resemblance (especially on the several most frequent patterns) between the true top-k and the top-k we found.

Convergence analysis. A decision we have to make is when to stop the random walk and output a sample. In Section 3.3 we introduced Z-score based Geweke diagnostic, which compares the distribution at the beginning and end of the chain. Since MCMC is typically used to estimate a function of the underlying random variable instead of structural data like graphs, we need to choose some properties of the patterns which we will monitor using the Geweke test. The three metrics we use in the experiment are the number of neighbors $N(x)$, the number of frequent neighbors $N_1(x)$ and the number of nodes in the pattern $|x|$. Figure 5 shows the convergence traces of a sample run with $K = 20$ and $\varepsilon = 0.5$ on the *DTP* dataset. Each curve corresponds to the Z-score of a chain over the number of iterations. It can be seen that the Markov chain we design has pretty fast convergence rate thanks to the tuning of the proposal distribution. For each chain, convergence is declared when the Z-scores of all three metrics have fallen within the $[-1, 1]$ range for 20 iterations continuously. In Figure 5, this happens around 150 iterations for most chains.

7. RELATED WORK

Data Mining with Differential Privacy. There exist two approaches to differentially private data mining. In the first approach, the data owner releases an anonymized version of the dataset under differential privacy. And the user has the freedom of conducting any data mining task on the anonymized dataset. We call this the 'publishing model'. Examples include releasing anonymized version of contingency tables [28], data cubes [7] and spatial data [6]. The general idea in these work is to release tables of noisy counts (histograms) and study how to ensure they are sufficiently accurate

for different query workloads. In the other approach, differential privacy is applied to a specific data mining task, such as social recommendations [18] and frequent itemset mining [3]. The problem addressed in this paper falls into this category.

Privacy-Protection of Graphs. The aforementioned works on differentially private data mining all deal with structured data. For graph data, there is plenty of research effort [1] to anonymize a social network graph to prevent node and edge re-identification. But most of them focus on modifying the graph structure to satisfy k-anonymity, which has been proved to be insufficient [1]. Recently, several works [16, 13, 25, 14, 21] emerge to provide private analysis of graph data. Two types of differential privacy have been introduced to handle graph data: node differential privacy and edge differential privacy. It is still open whether any nontrivial graph statistics can be released under node differential privacy due to its inherent large sensitivity (e.g., removing a node in a star graph may result in an empty graph). Hay *et al.* [13] consider the problem of releasing the degree distribution of a graph under a variant of edge differential privacy. More recently, Karwa *et al.* [16] propose algorithms to output approximate answers to subgraph counting queries, i.e., given a query graph H (e.g. a triangle, a k-star), returning the number of edge-induced isomorphic copies of H in the input graph. Unfortunately, their work does not support the case when H is an arbitrary subgraph yet.

In contrast, we have a different problem setting from [16]. First, like [3], our privacy-preserving algorithm is associated with a specific and more complicated data mining task. Second, we consider a graph database containing a collection of graphs related to individuals.

Graph Pattern Mining. Finally, we briefly discuss relevant works on traditional non-private graph pattern mining. Earlier works which aim at finding all the frequent patterns in a graph database usually explore the search space in a certain manner. Representative approaches include *a priori*-based (e.g. [15]) and pattern growth based (e.g. gSpan [29]). Recent works aim at mining *significant* or *representative* patterns with scalability. One way of achieving this is through random walk [2], which also motivates our use of MCMC sampling for privacy preserving purpose. Another remotely related work is [27], which connects probabilistic inference and differential privacy. It differs from this work by focusing on inferencing on the output of a differentially private algorithm.

8. CONCLUDING REMARKS

In this paper we have presented a novel technique for differentially private mining of frequent graph patterns. The proposed solution integrates the process of graph mining and privacy protection into an MCMC sampling framework. Moreover, we have established the theoretical privacy and utility guarantee of our algorithm. Experiments on both synthetic and real datasets show good precision and support accuracy with moderate amount of privacy budget. We also notice the drop in utility with the increase of the number of outputs or the decrease in dataset size, which is inevitable under the requirement of differential privacy.

Acknowledgments. This research is partially supported by the National Science Foundation under the award CNS-0747247.

9. REFERENCES

[1] C. Aggarwal and S. Philip. *Privacy-preserving data mining: models and algorithms*. 2008.

[2] M. Al Hasan and M. J. Zaki. Output space sampling for graph patterns. *Proc. VLDB*, 2(1):730–741, 2009.

[3] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *KDD*, pages 503–512, 2010.

[4] J. Cheng, Y. Ke, and W. Ng. Graphgen: A graph synthetic generator. http://www.cse.ust.hk/graphgen, 2006.

[5] L. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10), 2004.

[6] G. Cormode, C. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially Private Spatial Decompositions. *ICDE*, 2012.

[7] B. Ding, M. Winslett, and J. Han. Differentially private data cubes: optimizing noise sources and consistency. *SIGMOD*, 2011.

[8] C. Dwork, K. Kenthapadi, and F. McSherry. Our data, ourselves: Privacy via distributed noise generation. *Advances in Cryptology*, 2006.

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

[10] J. Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In *Bayesian Statistics*, 1992.

[11] W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC, 1996.

[12] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM*, pages 1–9, 2010.

[13] M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate Estimation of the Degree Distribution of Private Networks. *ICDE*, pages 169–178, Dec. 2009.

[14] S.-S. Ho and S. Ruan. Differential privacy for location pattern mining. In *SIGSPATIAL Workshop on Security and Privacy in GIS and LBS*, pages 17–24. ACM, 2011.

[15] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.

[16] V. Karwa, S. Raskhodnikova, and A. Smith. Private Analysis of Graph Structure. *Proceedings of the VLDB*, 4(11):1146–1157, 2011.

[17] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: frequent itemset mining with differential privacy. *VLDB Endow.*, 5(11), July 2012.

[18] A. Machanavajjhala, A. Korolova, and A. Sarma. Personalized social recommendations-accurate or private? *VLDB*, 4(7), 2011.

[19] F. McSherry and I. Mironov. Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders. In *KDD*, pages 627–636, 2009.

[20] F. McSherry and K. Talwar. Mechanism Design via Differential Privacy. *FOCS*, 2007.

[21] D. Mehmood, B. Shafiq, J. Vaidya, Y. Hong, N. Adam, and V. Atluri. Privacy-preserving subgraph discovery. In *Data and Applications Security and Privacy XXVI*, pages 161–176. Springer, 2012.

[22] N. Mohammed, R. Chen, B. C. M. Fung, and P. S. Yu. Differentially Private Data Release for Data Mining. *KDD*, 2011.

[23] A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea, and A. Joshi. On the structural properties of massive telecom call graphs: findings and implications. In *Proceedings of CIKM*, pages 435–444. ACM, 2006.

[24] R. Rubinstein and D. Kroese. *Simulation and the Monte Carlo method*. Wiley, 2008.

[25] A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao. Sharing graphs using differentially private graph models. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 81–98. ACM, 2011.

[26] E. Shen and T. Yu. Mining frequent graph patterns with differential privacy. http://arxiv.org/abs/1301.7015.

[27] O. Williams and F. McSherry. Probabilistic inference and differential privacy. In *Neural Information Processing Systems (NIPS)*, 2010.

[28] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on Knowledge and Data Engineering*, pages 1200–1214, 2010.

[29] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, 2002.

[30] M. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *Knowledge and Data Engineering, IEEE Transactions on*, 17(8):1021–1035, 2005.