Interactive Pattern Mining on Hidden Data: A Sampling-based Solution *

Mansurul Bhuiyan, Snehasis Mukhopadhyay, and Mohammad Al Hasan
Dept. of Computer and Info. Science, Indiana University–Purdue University, Indianapolis, USA
mbhuiyan@iupui.edu, smukhopa@cs.iupui.edu, alhasan@cs.iupui.edu

ABSTRACT

Mining frequent patterns from a hidden dataset is an important task with various real-life applications. In this research, we propose a solution to this problem that is based on Markov Chain Monte Carlo (MCMC) sampling of frequent patterns. Instead of returning all the frequent patterns, the proposed paradigm returns a small set of randomly selected patterns so that the clandestinity of the dataset can be maintained. Our solution also allows interactive sampling, so that the sampled patterns can fulfill the user's requirement effectively. We show experimental results from several real life datasets to validate the capability and usefulness of our solution; in particular, we show examples that by using our proposed solution, an eCommerce marketplace can allow pattern mining on user session data without disclosing the data to the public; such a mining paradigm helps the sellers of the marketplace, which eventually boost the marketplace's own revenue.

Categories and Subject Descriptors

H [**Information Systems**]: Miscellaneous; I [**Computing Methodologies**]: Miscellaneous

Keywords

MCMC Sampling, Interactive Pattern Mining

1. INTRODUCTION

Frequent pattern mining plays a key role in exploratory data analysis. Over the last two decades, researchers invented various efficient algorithms for mining patterns of varying degrees of complexity–such as, sets [1, 16], sequences [30], trees [31] and graphs [21, 29]. All these algorithms assume that the entire dataset is available to the analyst before the mining task is commenced. This assumption holds if the data owner and the data analyst are the same entity; however, in real life, data may be owned by an enterprise and an analyst who mines the data in quest of interesting patterns may be an outsider who does not have full access on the data. The data owner, though interested in maintaining the confidentiality of the data, wants to help the analyst by providing her a curated set of frequent patterns that would benefit her the most. A real-life example is provided in the following paragraph.

User session data, which contains the collection of related query-set that a user executes in a session, is a valuable data source for an eCommerce marketplace; it helps the marketplace to build features such as query suggestion [18], query segmentation [24], and product recommendation [20]. To maintain the competitive advantage, the data manager of the marketplace keeps the user session data inaccessible. However, the marketplace also has incentives to make a summary of the user session data, say, a random sample of frequent itemsets of user queries available to its sellers so that the sellers are aware of the demand trends in the marketplace. Availability of frequent itemsets of user session queries also helps the sellers in choosing an informative title for their product to facilitate effective matching of seller's product with the buyer's query—eventually, boosting the marketplace's revenue. Since different sellers may be interested in different sets of queries, the key challenge for the marketplace in this task is to find a mechanism to assist a seller by providing frequent itemsets of queries that would benefit that specific seller the most—all without risking the leakage of significant part of the session data.

There exist quite a few works that perform data analytics over databases that are hidden behind a form-like query interface [11, 10, 12]. Similar to these works, we consider the task of mining frequent patterns from a hidden dataset; however, the similarity ends right there, as we adopt a different system setup, and a different methodology. In our setup, the owner of the hidden dataset allows a data analyst to obtain random samples of frequent patterns from the dataset. Thus, by returning a (small) random subset of frequent patterns, instead of returning all the frequent patterns, the data owner maintains the confidentiality of the data in the sense that using these samples the analyst can not reconstruct the entire dataset with high level of accuracy; Dinur et. al. [13] named such a dataset a private dataset. It is relevant to mention that though the exact reconstruction of a dataset from the list of frequent itemsets is $\mathcal{NP}$-complete, approximate reconstruction is possible when all the frequent itemsets and their supports are available [27].

Besides the above, the sampling mechanism in our mining

*This research is supported by an NSF CAREER Award (IIS-1149851)

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

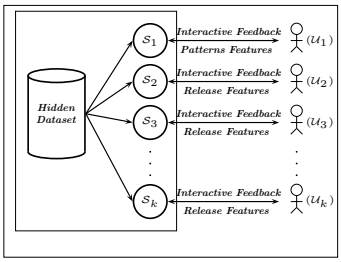


Figure 1: Sampling based Interactive Pattern Mining on a Hidden Dataset

system is interactive so that the analyst can obtain the most fruitful mining results. Given that the dataset is hidden, the analyst only has a vague idea about the dataset's content, so interactive mining is essential so that the sampling process can retrieve the patterns that best serve the analyst. Simply put, our system enables the analyst to provide feedbacks on the quality of the patterns that she receives; the sampling mechanism uses these feedbacks to bias the sampling towards the preferences of the analyst, so that the patterns obtained from subsequent samples are more useful to the analyst.

Even if the data is not hidden, the interactive sampling of frequent patterns is useful on its own merit. For example, in an interdisciplinary research, the domain data may not be hidden to the domain expert; however, she has little knowledge about the quality of the patterns that she obtains by mining those data. Effective discovery of useful patterns requires domain knowledge; on the other hand, effective incorporation of domain knowledge requires an interactive console. Existing pattern mining algorithms are not interactive; further, they generate an enormous list of frequent patterns, in which, a high degree of redundancy prevails. Such a tool is hardly helpful for a domain scientist. An interactive pattern mining approach that enables the domain scientist to sample a small set of relevant frequent patterns is a better alternative.

Figure 1 illustrates the framework that we devise to mine patterns from a hidden dataset. In this figure, we show a set of k analysts, $u_1, u_2, \ldots, u_k$, that are simultaneously accessing frequent patterns from a hidden dataset. For a specific user, u_i, a dedicated sampler, s_i is assigned to facilitate the interactive mining session. Based on u_i's feedback on the sampled patterns, s_i updates the sampling distribution so that the analyst u_i is served in the most fruitful way. It is easy to see that the overall framework is generic and is easily applicable for sampling itemsets, trees, sequences and graphs.

We summarize the contributions of this work as below:

• We propose an interactive pattern mining system that is based on sampling of frequent patterns. Instead of enumerating all the frequent patterns in the dataset, the above system uses a sampler for returning a small set of randomly selected frequent patterns to the analyst. Through an interactive console, it (the system) enables the analyst to provide feedbacks on the quality of the sampled patterns. Based on these feedbacks, the mining engine updates its sampling criteria so that in subsequent iterations it samples patterns that are of better quality.

• The above interactive pattern mining system enables mining of frequent patterns from hidden data. It is evident that the sampler only relies on user's feedback to drive the pattern selection process, so a user does not need access to the actual dataset on which the mining is being accomplished. Rather, the user exploits the interactive pattern mining system to obtain the desired patterns.

• Our sampling-based pattern mining paradigm also overcomes the *information overload* problem that is caused by the large number of frequent patterns in a traditional pattern mining task.

2. RELATED WORKS

We discuss the related works under three different categories.

2.1 Data Analytics over Hidden Databases

Data analytics over hidden databases is an active research direction; typically, these hidden databases are part of the deep Web that can be accessed only by a form-like query interface. Along this research direction, various tasks are considered, such as, crawl the hidden Web [25], obtain a random tuple from a hidden database [11, 12], retrieve top-k records by sampling [23], and estimate the size and other aggregates over a hidden database [10]. These works are related to our work in the sense that we consider frequent pattern mining, also over the hidden databases. However, we do not assume a form-like interface to access the database, rather we assume that the data owner provides a random sampler to interact with the database and the sampler is guided by the data analyst through interactive feedbacks.

2.2 Interactive Pattern Mining

Surprisingly, existing works on interactive pattern mining (IPM) are sparse [7, 5, 14]. One of the main techniques that the existing works follow is constraint-based mining, where a user can add additional constraints as an interactive input. The mining system considers these constraints to filter the output set to tailor it according to the user's requirements. Setting constraints is effective for some mining problems, but for many others, applying hard constraints may yield sub-optimal results. Also, designing constraints to guide mining is not easy, particularly while mining from a hidden dataset; an analyst first needs to explore some example patterns before she can think about the constraints that would work best for her specific applications. This can best be described as *"I don't know what I am looking for, but I would definitely know if I see it"*.

Few methods on interactive mining exist that are not constraint-guided—probably the most similar to our work is [28]. However, our method differs from it in various ways. First, the above method requires to have a complete set of frequent patterns up-front, whereas our method unifies the exploration and selection of the frequent patterns using MCMC based random walk—thus, for many datasets, our method may not explore all the frequent patterns. Second, our method uses binary feedback instead of rank order feedback that the above method adopts—the latter puts more burden on the user, and in case of a hidden dataset, users may have little information to rank the patterns properly.

In interactive patterns mining, a user decides whether a pattern is interesting or not. There also exist a few notable works [22, 3, 26], which mine a small set of interesting patterns by defining novel interestingness metrics for frequent patterns. The scope of these works is complementary to the work that we present in this paper.

2.3 Frequent Pattern Sampling

For mining frequent patterns from a hidden dataset, we use an MCMC based random walk on the frequent pattern space. In recent years, various other works also follow a similar approach—most notable among these are [4, 19, 17]. In [19], Hasan et. al. proposed a mining framework, that they called *output space sampling*, which can sample frequent patterns using a user-defined distribution. The sampling framework that we devise in this work is similar to the work in [19]; however in [19], the sampling distribution remains static throughout the mining session, whereas in our work, the sampling distribution keeps changing in response to the user's feedback. Also, in [19], the random walk is performed over the edge of the Partial Order Graph (POG) (defined formally in Section 4); our work deviates from this by performing random walk over a state-transition graph in which a random graph is overlaid on top of the POG graph—the above modification improves the convergence of an MCMC walk significantly.

3. PROBLEM STATEMENT

In this section, we formally define the problem of interactive pattern mining from hidden data. Given a hidden dataset [1] $\mathcal{D}$, and a user u, an interactive pattern mining system (IPM) returns a sequence of randomly selected frequent patterns $p_1, p_2, \ldots p_t$ from $\mathcal{D}$ to u; optionally u can send feedbacks $f_1, f_2, \ldots f_t$ to the IPM regarding the quality of the patterns; in response to the feedback, IPM iteratively updates its sampling distribution so that the randomly selected patterns that it returns to u in subsequent iterations align better with u's interest. In the following paragraphs, we will detail the followings: (1) how does the IPM system find a pattern? (2) how does the user u form a feedback on a pattern?, and (3) how does the feedback from u is incorporated in the mining process to obtain a better pattern in subsequent iterations?

Given that user u has no access to the dataset, and the data owner has no intention to share all the frequent patterns, traditional frequent pattern mining algorithms are of no use in this scenario. We also assume that the dataset is large; therefore the complete enumeration of all the frequent patterns may be infeasible. So, the IPM adopts the output space sampling (OSS)-like [19] paradigm to obtain a frequent pattern and share it with the user u. However, OSS does not provide an option for interactive pattern mining, it merely allows the user to mine from a user-defined sampling distribution. So, to accommodate an interactive mining setup, IPM adapts the sampling distribution using u's feedback, so that it respects u's interest progressively. If we assume that u has an underlying *interestingness function*, γ, that maps each frequent pattern to a non-negative real value (higher value stands for more interesting), the adaptation of sampling distribution by using the feedback translates to the inverse problem of learning the γ function.

An important question that is yet to be answered is what is the composition of u's feedback on a pattern. In this work, we consider a simple feedback mode, where the response of the user u on a pattern p is purely binary: *interesting* and *non-interesting*. One can always consider a more complex feedback mechanism, may be in the form of a profile vector, in which the user u represents the pattern p as a composition of a set of sub-patterns and uses a probabilistic vector to define her interest in each of these parts; however, we do not explore this option in this work. Now, under the scope of a binary response, the set of frequent patterns can be divided into two groups: useful and non-useful, and it is left to the IPM to deduce the underlying discriminating function by which the user u is building her opinion.

[1] the words *dataset*, and *database* are used synonymously this this document

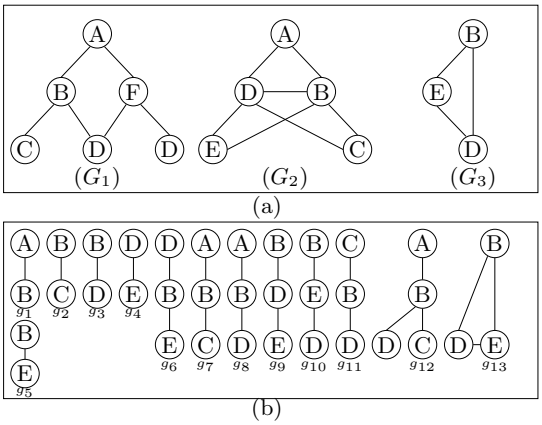


Figure 2: (a) Graph database with 3 graphs (b) Frequent subgraph of (a) with $minsup = 2$

We like to clarify that our problem formulation is generic and it considers the task of mining all kinds of patterns, such as, sets, sequences, trees or graphs. In most of our discussion in this paper, we will refer to the term "pattern" and will dictate the discussion considering a graph pattern, but with minor or almost no adjustment the discussion will fit for different kinds of patterns. In the experiment section, we will show results involving both itemset and graph patterns.

4. BACKGROUND

In this section, we define several key concepts that will be useful to understand our research work.

4.1 Frequent pattern Mining

Let, $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ be a graph database, where each $G_i \in \mathcal{G}, \forall i = \{1 \ldots n\}$ represents a labeled, undirected and connected graph. $\mathbf{t}(g) = \{G_i : g \subseteq G_i \in \mathcal{G}\}, \forall i = \{1 \ldots n\}$, is the *support-set* of the graph g. This set contains all the graphs in $\mathcal{G}$ that has a subgraph isomorphic to g. The cardinality of the *support-set* is called the *support* of g. g is called frequent if $support \geq \pi^{\mathbf{min}}$, where $\pi^{\mathbf{min}}$ is predefined/user-specified *minimum support (minsup)* threshold. The set of frequent patterns are represented by $\mathcal{F}$. **example:** Figure 2(a) shows a database with 3 graphs (G_1, G_2 and G_3). With $\pi^{\mathbf{min}} = 2$, there are thirteen frequent subgraphs as shown in Figure 2(b).

4.2 Frequent Graph Partial Order

Based on the sub-graph relationship, the set of all frequent sub-graphs ($\mathcal{F}$) forms a partial order called *Subgraph partial order*. The lowest pattern in this partial order is the null ($\emptyset$) pattern. The partial order can be represented as a graph, called *partial order graph (POG)*. Every node in the POG corresponds to a distinct frequent graph pattern, i.e., each graph in POG is the canonical representative (with the minimal DFS code [29]) for all other graphs that are isomorphic to it. Every edge in POG represents a possible extension of a

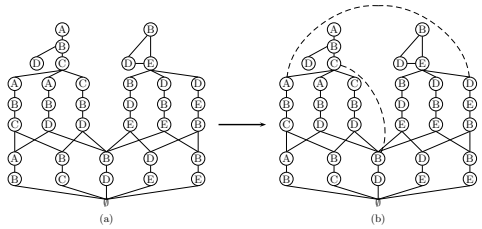


Figure 3: (a) Partial Order Graph (POG) (b) State-transition Graph

frequent pattern to a larger (by one edge) frequent pattern. Algorithms for enumerating all frequent subgraphs typically traverse the POG in either depth-first or breadth-first manner, starting from the lowest pattern. Figure 3(a) shows the frequent subgraph partial order for the graph mining task shown in Figure 2. It is easy to see that for the case of mining other patterns, such as, sets, sequences and trees, a similar POG can be built, using the subset, subsequence or subtree relationship, respectively.

4.3 Markov chains, Random walk, Metropolis-Hastings (MH) Algorithm

A **Markov chain** is the sequence of Markov process over the state space S. The state-transition event is guided by a matrix, T, called *transition probability matrix*. A Markov chain is said to reach a stationary distribution π, when the probability of being in any particular state is independent of the initial condition. Markov chain is reversible if it satisfies the *reversibility condition* $\pi(i)T(i,j) = \pi(j)T(j,i), \forall i,j \in S$. A Markov chain is *ergodic* if it has a stationary distribution. If the state space (S) of a Markov chain is the vertices of a POG and the state transition happens only between adjacent nodes in POG, then this Markov process simulates a random walk on the frequent pattern space ($\mathcal{F}$).

The main goal of the Metropolis-Hastings algorithm is to draw samples from some distribution $\pi(x)$, called the *target distribution*, where, $\pi(x) = f(x)/K$; here K is a normalizing constant which may not be known and difficult to compute. MH algorithm can be used together with a random walk to perform Markov Chain Monte Carlo (MCMC) sampling. For this, the MH algorithm draws a sequence of samples from the target distribution as follows:
i. It picks an initial state (say, x) satisfying $f(x) > 0$.
ii. From current state x, it samples a point y using a distribution $q(x,y)$, referred as *proposal distribution*.
iii. Then, it calculates the *acceptance probability*,

$$\alpha(x,y) = \min\left(\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right) = \min\left(\frac{f(y)q(y,x)}{f(x)q(x,y)}, 1\right) \quad (1)$$

and accepts the proposal move to y with probability $\alpha(x,y)$. The process continues until the Markov chain reaches to a stationary distribution.

As shown in [19], a Metropolis-Hastings (MH) based MCMC algorithm can be used for sampling frequent pattern with a desired sampling distribution. In such a method, the set of frequent patterns ($\mathcal{F}$) is the state space of the random walk and the edges of POG define the possible state transitions. Importantly, the POG are generated locally as needed; for instance, when the random walk resides on a pattern x, the local neighborhood of the pattern x, which consists of all frequent super patterns (whose sizes are larger by one than the size of x) and all frequent sub-patterns (whose sizes are smaller by one than the size of x), is constructed. The random walk then jumps to one of its neighbors using the acceptance probability (Equation 1) of MH algorithm. MH requires to choose a proposal distribution (q), which can simply be the uniform distribution i.e., each of the neighbors of the pattern x in POG is equally likely to be chosen. If y is a neighbor of the pattern x, d_x and d_y are the degrees of the pattern x and y in POG, and γ is the target distribution, following the equation 1, the acceptance probability for choosing the proposal move is as below:

$$\alpha(x,y) = \mathbf{min}\left(\frac{\gamma(y)\cdot(1/d_y)}{\gamma(x)\cdot(1/d_x)}, 1\right) = \mathbf{min}\left(\frac{\gamma(y)d_x}{\gamma(x)d_y}, 1\right) \quad (2)$$

The following Lemma holds:

LEMMA 1. *The random walk on POG converges to a stationary distribution which is equal to the desired target distribution, γ.*
PROOF*: See [19]* ■

5. INTERACTIVE SAMPLING ALGORITHM

We also use MH as the underlying sampling algorithm to mine patterns from the hidden database, $\mathcal{D}$. Each sampler s_i in Figure 1 (b) runs an instance of MH sampling with the objective that MH's target sampling distribution (γ in Equation 2) matches with the desired sampling distribution of the user u_i. Achieving the above objective is challenging because the sampler has no knowledge of the desired sampling distribution. To overcome this difficulty, we sample patterns using MH with a default sampling distribution, and update the default distribution using each of the user feedbacks so that the effective distribution converges towards γ with the updates.

5.1 User's Scoring function

We represent the user's desired distribution as a scoring function, $f : \mathcal{F} \to \mathbb{R}_+ \cup 0$; f maps each pattern in $\mathcal{F}$ (frequent pattern-set) to a non-negative real number. From this scoring function, we can obtain user's desired distribution by $\frac{f(P)}{\sum_{q\in\mathcal{F}} f(q)}$, i.e., the probability of a pattern in the desired distribution is directly proportional to the $f(\cdot)$ (f-score) of that pattern [2].

For the sake of learning, we must impose bias on γ and assume that γ is taken from some family of functions [3]. For this, we use the similarity between interactive pattern mining (IPM) and the task of Query by Example in information retrieval (QEIR) [8]—*given a set of documents that a user has liked, retrieve other documents that he is likely to like*; for IPM, the set of patterns for which the user provides a positive feedback plays a role that is similar to the role of the given documents in a QEIR task. Typically, QEIR uses the vector space model of words and phrases to construct user's preference profile. Similarly, to model γ we also adopt a vector space model of unit-size patterns. For example, unit-size patterns for a graph are its edges; for an itemset, they are the items. In this vector space model, each unit-size pattern has a weight, expressed as b^i where $b \in (1, +\infty)$ and $|i| \in 0, 1, \ldots, |\mathcal{F}|$. The γ-score of a pattern is simply the

[2]In this paper, we will use the phrases *scoring function* and the corresponding *sampling distribution* synonymously.
[3]this argument is known as *No Free Lunch Theorem* in the machine learning domain

multiplication of these weights [4]. If P is a pattern of size k and P_j are the unit-size patterns that compose the pattern P, we have:

$$\gamma(P) = \prod_{j=1}^{k} \mathbf{w}[P_j],\ \forall P_j \in P \text{ and } |P| = k \tag{3}$$

Note that, if two patterns x and y are structurally similar, they will have high overlap between their constituting unit-size patterns; hence, under the above assumption, they will have similar γ-score.

To assign a weight on each unit-size patterns, we assume that the set of frequent patterns are partitioned into two sets: *interesting* ($\mathcal{I}$) and *non-interesting* ($\mathcal{N}$); so, $\mathcal{F} = \mathcal{I} \cup \mathcal{N}$. Also, assume that $\mathcal{F}_1$ is the set of all unit-size frequent patterns. Now, consider a weight vector $\mathbf{w}$ of size $|\mathcal{F}_1|$; each entry $\mathbf{w}[\cdot]$, in this vector corresponds to one of the unit-size frequent patterns; for a unit-size pattern p, $\mathbf{w}[p]$ is equal to b^{u-v}, where $(u, v) \in [0, 1, \ldots, |\mathcal{F}|]$ are the number of interesting and non-interesting patterns (according to a user) in which p is embedded. In this way, if a unit-size pattern is part of x interesting (from the set $\mathcal{I}$) and y non-interesting (from the set $\mathcal{N}$) patterns, its weight will be b^{x-y}.

Example: In Figure 2 (b), if $\mathcal{I} = \{g_1, g_2, g_7, g_{12}\}$. Suppose $b = 2$, then, $\mathbf{w}[A\text{-}B] = 2^{3-1}$ as the unit-size pattern A-B is part of three (g_1, g_7, g_{12}) interesting and one (g_8) non-interesting patterns. On the other hand, $\mathbf{w}[B\text{-}D] = 2^{1-6}$, as this edge appears in 1 interesting and 6 non-interesting patterns. Also, $\gamma(g_7) = 2^4$, which we obtain by multiplying the weights of each of its edges (unit-size patterns). ■

5.2 User interaction Design

For interactive pattern mining, we assume that the γ belongs to the family of the functions described in Section 5.1. Since, γ is unknown to the sampler, the sampler (s_i) uses a proxy distribution (say β) as the effective target distribution, which the sampler updates with each of the feedbacks. If $\beta^0, \beta^1, \ldots, \beta^k$ are sequences of β's along with the updates, then hypothetically, $\mathbf{Lim}_{k\to\infty}\beta^k = \gamma$. Since, β requires to converge to γ, we ensures that each of the β^k distribution also belongs to the same family of distribution as of γ. Thus, the $\beta(\cdot)$ (β-score) of a pattern is computed as the multiplication of the weight of its constituting unit-size patterns, identically as it is computed for the γ-score (see Equation 3).

At initialization, all unit-size patterns have a weight of 1, and sampling using such a β (let's name it β^0) as the target distribution yields a random initial distribution. The sampler starts sampling from the initial distribution by using the MCMC based random walk; suppose, at some step of the walk, it (the sampler) picks a pattern P of length k for user's feedback. If the pattern P has already been used for feedback, the pattern is discarded and the process continues until another pattern is found for which the feedback has not been sought yet. Then, for the selected pattern, the user provides her feedback by indicating whether she likes the pattern P or not. If the feedback is positive, the sampler increases the weight of each of k length-one sub-patterns of P exponentially, i.e. it multiplies the current weight of those length-one sub-patterns by b. On the other hand, if the feedback is negative, the sampler decrease the weight of those length-one sub-patterns by dividing the current weight by b. This weight modification updates the existing sampling distribution β. The sampling step and the interaction step are repeated intermittently as the β function iteratively moves closer to the user's desired distribution γ. The above weight update also ensures that in all β^k, there exists a positive probability to visit any frequent pattern from any other frequent pattern; which maintains the ergodicity of the random walk.

[4]An additive γ function may also be considered, but we notice that such a function has slow convergence compared to multiplicative functions.

LEMMA 2. $\boldsymbol{Lim}_{t\to\infty}\beta^t = \gamma$.

PROOF: β^t is the updated sampling distribution after t iterations; in each of these iterations, exactly one pattern is sent for user's feedback; hence, total patterns for which feedback is sought is t.

Now, consider, any unit-size pattern p, and assume that it appears in $r(\leq t)$ patterns out of those t patterns. Now, let's partition the r patterns into two set, based on their feedback status, interesting or non-interesting. If the size of these sets are r_i and r_n, respectively, according to the weight update mechanism, in β^t the weight of the pattern p, $\mathbf{w}[p]$, is equal to $b^{r_i - r_n}$.

Now, in all sampling distribution, $\beta^k : 0 \leq k \leq \infty$, every interesting/non-interesting pattern in $\mathcal{I}$ and $\mathcal{N}$ has a non-zero probability to be visited; hence, as t goes to infinity, all the patterns in the set $\mathcal{I}$ and $\mathcal{N}$ will be sent to the user (at least once) for feedback; Also, for each pattern, we consider it's feedback exactly once. So, the weight of p in the limiting distribution β^∞ is exactly equal to b^{x-y}, where x and y are the count of interesting and non-interesting patterns of which p is part-of, which is identical to $\mathbf{w}[p]$ of γ. Hence, $\beta^\infty = \gamma$ ■.

Lemma 2 proves that the user interaction that we design works perfectly when the γ function is constructed as was shown in Equation 3. While this is a reasonable assumption about γ, in real life, an interestingness function can be different. Nevertheless, any interestingness function that a user uses should have some structure; alternatively, it should be non-random, where structural similarity between a pair of patterns should have strong influence on the possible γ-scores of these patterns. As a result, the above proxy distribution β should work reasonably well.

5.3 State-transition graph and convergence of MH-based Random Walk

In [19], the authors use partial order graph (POG) as the state-transition graph for sampling frequent patterns. However, by construction, POG is a multi-stage graph—the patterns that have the same size belong to the same stage, and a pattern in stage i have neighbors only from stage $i+1$ and stage $i-1$. Such a graph has a large diameter (at least, as large as the size of the largest frequent pattern), which causes slow mixing for any random walk on it (the graph). The slower the mixing, the worse the sampling quality. In particular, for our task, fast mixing is crucial—we update the effective distribution (β^t) frequently, so we desire that the random walk to mix well before the effective distribution β^t is updated. The mixing time of a random walk is characterized by the number of steps the walk takes to reach its stationary distribution. For a known target distribution, the mixing time can be estimated by computing the spectral

gap [9] of the transition probability matrix P. In short, the higher the spectral gap, the faster the convergence.

To achieve fast mixing, we overlay a random graph on POG, by introducing random edges in the partial POG in a periodic manner with the exploration of a batch of new nodes of the POG. In Figure 3, we show two such random edges (shown as dotted line in the right figure), between patterns (*A-B-C*, *D-E-F*) and (*A-B-C-D*, *B-D*). Addition of these two edges reduces the diameter of the POG in Figure 3 by 40% (from 5 to 3). The reason for overlaying a random graph is due to the well known fact that a random graph exhibits rapidly mixing property—for instance, the mixing time of an MCMC random walk on a connected Erdos-Renyi random graph is $\Theta(\log^2 n)$ [2]. In all our experimental datasets, introducing random edges increases the spectral gap almost by 100%. Table 1 shows the improvement in the spectral gap for Mushroom, Chess and Connect datasets after setting the $randomedge\_factor$ (explain in Section 5.4) to 20%.

Dataset	spectral gap (μ)	
	POG + Random edge	POG
Mushroom	0.045	0.020
Chess	0.082	0.040
Connect	0.085	0.047

Table 1: Effect of random edges in POG on Spectral gap

Importantly, the addition of random edges maintains the ergodicity of the MCMC walk. This is due to the fact that the random edges are symmetric (allow transition along both the directions of the edge), and addition of these edges maintains the reversibility condition.

```
Interactive_Sampling
(D, minsup, miniter, feedback_count, b):
1.  P = generate_any_frequent_pattern(D, minsup)
2.  d_P = compute_degree(P)
3.  π(P) = compute_β-value(P)
4.  while (true)
5.     choose a neighbor, Q, uniformly from, all possible
       frequent super and sub patterns
6.     d_Q = compute_degree(Q)
7.     π(Q) = compute_β-value(Q)
8.     acceptance_probability = min ((π(Q)d_P)/(π(P)d_Q), 1)
9.     if uniform(0,1) ≤ acceptance_probability
10.       P = Q
11.       iter = iter + 1
12.    if((#ofNodeInCurrentPOG)/(#ofNodeInOldPOG) >= 3)
13.       insert_random_edge()
14.    if iter % miniter == 0 feedback_count >=0
15.       get_and_process_feedback(P, b)
16.    else
17.       goto line 5

compute_degree(D, P, minsup):
1.  super_pat = all_frequent_super_pattern(P)
2.  sub_pat = all_sub_pattern(P)
3.  neighbor_list = super_pat ∪ sub_pat
4.  save the neighbor list in POG data structure
5.  return size(neighbor_list)

compute_β-value(P):
1.  for each single length p in pattern P
2.    value = value * w[p]
3.  return value

get_and_process_feedback(P, b):
1.  if feedback is positive
2.     for each single length p in pattern P
3.        w[p] = w[p] * b
4.  else
5.     for each single length p in pattern P
6.        w[p] = w[p] * 1/b
7.  feedback_count = feedback_count − 1
```

Figure 4: Interactive Sampling Algorithm

5.4 Algorithm

For an incoming user, the IPM creates a new sampler instance which executes the subroutine $Interactive\_Sampling$ shown in this Figure 4. Besides the database, $\mathcal{D}$, the method also takes four more parameters: (1) a minimum support ($0 \leq \pi^{\mathbf{min}} \leq 1$) value, which defines whether a pattern is frequent or not; (2) a minimum number of walks ($miniter$) before the sampler returns a pattern to the user for feedback; (3) Total number of feedback ($feedback\_count$) and (4) a value for the base (b) which is used to update weight of unit size pattern.

In Line 1 of Figure 4, the sampler starts with any arbitrary frequent pattern P; a unit-length frequent pattern suffices. Then, it computes all frequent super and sub-patterns of P (Line 2). The degree of P is the size of the union set of super and sub-patterns. Then it chooses a pattern (Q) from P's neighbors (Line 5) uniformly and computes the acceptance probability in Line 8. If the move is accepted, Q becomes the resident state; otherwise, the sampler chooses another neighbor identically and repeat the whole process. If the user abort the sampling, the infinite while loop breaks.

During the MCMC walk, if the condition in Line 12, i.e., the ratio of the number of nodes in the current POG and the same in the previously marked POG is greater than 3, the sampler inserts a set of random edges to the current POG. We use a variable called $randomedge\_factor$ to control the number of inserted random edges. The value of $randomedge\_factor$ is not that important as long as a small fraction of edges in the state-transition graph are from the random graph; for all our experiment, we keep this fraction equal to 0.20.

The condition on Line 14 checks whether the sampler should send the resident pattern (P) to the user for feedback. If condition succeeds, depending on the feedback's status, the sampler updates the weight of each of the unit-size sub-patterns of P, which forces a change in the target distribution (β).

The above pseudo-code is generic and can easily be adapted for sampling any kind of patterns, e.g. itemsets, trees, sequences or graphs.

Example: Assume a random walk on the POG in Figure 3(b). At some stage of this walk, the resident state of the random walk is the pattern $g7$ (see Figure 2). It has 4 neighbors ($d_{g7} = 4$), one super-pattern ($g12$), two sub-patterns ($g1$ and $g2$), and a random edge. Assume that the existing weights (chosen arbitrarily) of the unit-size patterns $g1, g2, g3, g4$, and $g5$ are $\langle 2, 1, 2, 1, 1\rangle$ in that order. So, the weight of the pattern $g7$ is 2. If the proposal distribution chooses the pattern $g12$, to compute acceptance probability, we first compute the followings: β-score$(g12) = 4$, $d_{g12} = 4$; now the acceptance probability of the proposal move using equation 2 is: $\mathbf{min}\left(\frac{4*(1/4)}{2*(1/4)}, 1\right) = 1$. So, this move will always be accepted. If we send $g12$ to the user and the user likes it, the weight of pattern $g1, g2$, and $g3$ will be incremented by the multiple of b and the new weight vector for the unit-size patterns will be $\langle 2*b, b, 2*b, 1, 1\rangle$. ■

Choosing $miniter$**:** In our interactive pattern mining, the effective sampling distribution is β^t, which changes once a feedback on a pattern is received. So before returning a subsequent sample, the sampler needs to perform a few walk on the frequent pattern space—the parameter, $miniter$, de-

cides the number of walks. Now, exactly how many walks the sampler should perform depends on how fast the random walk mixes, which we can compute from the spectral gap. However, this computation requires the knowledge of P, which the user is unaware of. On the other hand, the sampler can't pre-compute P as it requires the effective target distribution β^t, which is always changing based on user's feedback. Also it may be overkill to compute P for every possible β^t, so the best is to leave this as a parameter to the user who can choose a suitable value based on the feedback interval that suits best for her. In all of our experiments, we fix this interval to be 25, i.e., the sampler returns a pattern to the user after it makes 25 walks.

Choosing base (b): The base (b) has a large significance in regard to learning the user's desired distribution, γ. A large value of b sharply increases (or decreases) the weight of a unit-length pattern in response to a positive (or negative) feedback. If we consider the learning of γ as a multi-arm bandit problem, each unit-length pattern can be considered as an arm, and we are trying to find a weight for each arm, so that using this weight in equation 3, we can approximate the user's desired distribution, γ. Under this assumption, the parameter b is simply the exploration-exploitation trade-off parameter. Choosing high value of b imposes higher weights on unit-length patterns that are part of some of the interesting patterns that have already been explored; as a result, the sampler is biased to choose patterns that are composed of these items. However, this bias restricts the sampler to explore other yet-to-be-explored patterns that may be interesting, but are not composed of unit-length patterns that have already been explored. For this reason, a higher value for b increases the sampler's precision since sampler will traverse patterns that are composed of (already known) interesting unit-length patterns; however, this yields poor recall as many other interesting patterns may remain unexplored. It is hard to choose a good operation point on the precision-recall curve, because in a hidden dataset the user has no idea about the number of interesting patterns that exist in the hidden dataset—a value that is needed to compute the recall. We found that b in the range of $(1, 2]$ works well for most of the datasets. We use b=1.75 for all our experiments.

5.5 Implementation Details

The algorithm that we show in Figure 4 is generic, and is not tied to any specific kind of patterns. Underlying the sampling process, the method shown in this algorithm also solves the frequent pattern discovery problem, as the sampler only visits frequent patterns. The task of frequent pattern discovery is embedded in the **compute_degree** method that finds *frequent* super-patterns and sub-patterns of a resident pattern. In our work, the implementation of this part is identical to the Hasan et.al's work on output space sampling—for details, the reader can check the Section 6 of [19].

5.6 Mining Patterns from Hidden Datasets

To mine a pattern from a hidden dataset, a user interface is used through which the user provides her feedback. At the initiation of an interactive mining session with a user u_i, the interactive system creates a new sampler process (say, s_i) that runs in the host machine (that stores the dataset) for managing dedicated communication between u_i and s_i (see Figure 1(b)). The sampler executes the interactive mining algorithm that is shown in Figure 4; the parameters of this algorithm, such as *minsup* and *miniter* can be obtained from the user. Through the user interface, the sampler sends a pattern to the user, gathers her feedback and updates the sampling distribution, by executing the method in Line 15 of Figure 4. The infinite loop terminates once the connection is closed by the user. The above setup enables interactive pattern mining by considering a semi-honest model [15], where each party obeys the protocol.

Dataset	# of transactions	Avg. transaction size	$\pi^{\mathbf{min}}$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{F}_1\rvert$
Chess	3196	37	2100	99262	29
Mushroom	8124	22	1200	53628	45
Connect	67557	43	58000	109664	24
eBay Query	10000	27	30	41130	3368
Mutagenicity-I	4336	17	195	3446	11
Mutagenicity-II	687	14	20	2935	13

Table 2: Dataset Statistics

6. EXPERIMENTS & RESULTS

We conduct several experiments to manifest the main purpose of interactive pattern sampling, which is to mine a set of interesting patterns by utilizing the binary feedback from the user. Our experiments show results both from graph and itemset mining.

6.1 Experiment setup

For our experiment, we require to define an interestingness function (γ) by which the user makes an opinion about a pattern. We mark m% randomly selected unit-size patterns among all the frequent (for a given *minsup*) unit-size patterns as interesting according to a user. With these m% patterns we build a set called I_p. We assume that the user is interested in patterns where the majority of the single length sub-patterns are from the set I_p; thus, the scoring function, γ for a pattern P is defined as, $\gamma(P) = \frac{\sum_{i=1}^{k} I(p_i)}{k} * 100$

Here, $p_1, p_2, \ldots, p_k$ are the unit-length sub-patterns of P, and I is an indicator random variable that defines whether a pattern p_i is from the set I_p or not. The γ-score of a pattern varies between 0 and 100, and the patterns that have γ-score more than 50 are interesting to the user, for which user provides a positive feedback; for the remaining patterns user provides a negative feedback. We set $m = 30\%$ in all of our experiments, unless specified otherwise.

6.2 Dataset

We show experimental results on six real-life datasets shown in Table 2. The first four among them are itemset datasets and the remaining two are graph datasets. We obtain Mushroom, chess and connect dataset from the UCI Machine Learning repository [5]. *eBay Query* is the remaining itemset dataset that we generate by scrapping queries from the *"Related Searches"* [6] feature of eBay. We build a query-graph where related queries are neighbors and populate the row of an itemset dataset by performing short random walks on this graph; in this way, the list of the queries that such a walk visits makes a transaction in the eBay Query dataset. Since related search suggestion is built by using user session

[5] `http://archive.ics.uci.edu/ml/`

[6] When a user search for an item in `www.ebay.com` with a query phrase, the search result page shows (at most) 10 keywords related to the original query.

data [18], this dataset is a metaphorical user session data, where a transaction can be regarded as the set of queries that a user executes in a user session. The remaining two datasets, Mutagenicity-I and Mutagenicity-II are chemical (graph) datasets that we obtained from the authors of [6].

Table 2 also shows some statistics of the datasets. The second and third column shows the number and average size of each transaction, the fifth column shows the number of frequent patterns, considering the minimum support value listed on the fourth column. The sixth column shows the total number of distinct unit-size frequent patterns.

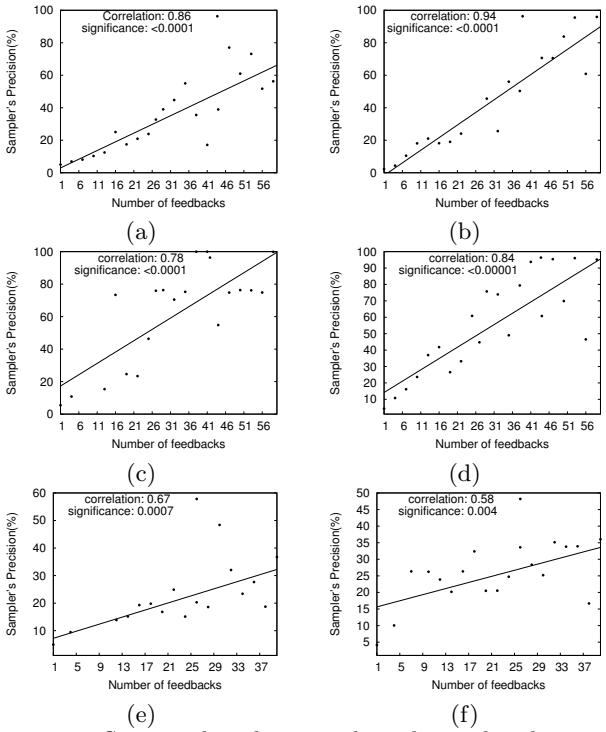


Figure 5: Scatter-plot showing the relationship between the sampler's precision and the number of feedbacks in (a) Mushroom (b) chess (c) eBay (d) connect (d) Muatgenicity-I and (f) Mutagenicity-II. A least-square fitted straight line with positive slope shows that for all the datasets the pair of variables are positively correlated

6.3 Analysis of sampler's performance

The objective of these set of experiments is to show the dependency of sampler's convergence with various parameters of the interactive sampling. We first show that with an increasing number of feedbacks, the sampler converges to a distribution, where the interesting patterns are more likely to be sampled. For this, we run the sampler for a large number of iterations and compute the percentage of iterations in which the sampler visits an interesting pattern; we call this percentage *samplers precision* (SP). We compute SP from many independent runs, each using a different value for the $feedback\_count$. Then we find the correlation between the number of feedbacks and SP. Since the process is randomized, SP value for each $feedback\_count$ is also computed by averaging over 10 runs.

We show our results in Figure 5. As we can see, the number of feedback and the samplers precision are highly correlated—SP increases along with the feedback count. Also note that,the strength of correlation varies across various datasets—specifically, the correlations are poor for graph datasets, and good for itemset datasets. This is expected, because in our scoring function (γ), we are treating a graph as if it is simply a multiset of labeled edges (unit-pattern)—this assumption ignores the neighborhood structure of a graph, which may degrade the sampling quality. Another important observation is that, it does not take many feedbacks for the sampler to achieve a fairly good correlation; in all the plots, the maximum number of feedback that we use is less than 60, which is a negligible fraction of the entire frequent pattern set—see the fifth column of Table 2 for a count of frequent patterns in each of the datasets. 60 is also a reasonable number considering the real-life scenario, where a human inspects the patterns and provides feedback on them.

base(b)	Avg sampler's precision(%)	Avg. recall (%)
1.5	43	70
1.75	56.7	61
2	61.7	43
2.5	96.5	9.6

Table 3: effects of parameter b on Mushroom dataset

To show the effect of b, the exploration-exploitation tradeoff parameter, we run our sampling method for several b values and report average samplers precision and recall. Here the precision is as we defined earlier, and the recall is the fraction of interesting patterns that the sampler have explored out of all the interesting patterns. The result is shown in Table 3, where the precision and recall are averaged over 10 distinct runs; as we expect, an increment of b increases the samplers precision with a reduction in the recall value.

We also study, how the sampler's performance changes, as the fraction of interesting patterns in the frequent pattern set ($\mathcal{F}$) changes. For this, we vary the parameter m which defines what fraction of unit-size items are interesting to the user. For this experiment, we use the Mushroom dataset with a support value of 1200. The number of frequent patterns, i.e., $|\mathcal{F}| = 53628$ and the number of unit-size patterns ($\mathcal{F}_1$) is $= 45$. For $m = 10\%$, on average, 5 unit-size patterns from the set $\mathcal{F}_1$ will be chosen as interesting, and the number of interesting pattern is 56, on average. Samplers precision for this m value is 5.8%, i.e, out of 1000 sampled patterns, 58 patterns are interesting. This seems to be a low number, but considering the fact that a uniform sampler will pick one interesting pattern from every thousand samples ($56 * 1000/53628 \sim 1$), the interactive sampler is 58 times more efficient. For $m = 20\%, 30\%, 40\%$, and 50%, the average samplers precision is 26.5%, 56.7%, 63.2% and 74%, respectively. Sampler's precision increases with the increment of m, as there are more interesting patterns in the search space ($\mathcal{F}$). Results on other datasets are similar, which are not shown due to the page restriction.

6.4 Interactive vs Uniform Sampling

Does user interaction really help? If we let the sampler perform a uniform sampling, will that achieve as good result as an interactive sampling session? We somewhat answer the question in the last paragraph of the earlier subsection, here we answer it in more details. For this, we compare between interactive sampling and uniform sampling. Note that, for uniform sampling, every pattern in $\mathcal{F}$ has equal probability to be sampled; i.e., the γ-score is $\frac{1}{|\mathcal{F}|}$ for all the patterns. In our implementation of interactive sampling, we simply

Dataset	Sampler's Precision(%)	
	Uniform	Interactive
Mushroom	0.182	56.70
Chess	2.293	95.74
Connect	4.53	94.32
eBay Query	15.23	98.50
Mutagenicity-I	1.94	36.76
Mutagenicity-II	1.15	35.00

Table 4: Average sampler's accuracy measure of uniform and interactive sampling.

turned off the feedback and use the above γ to obtain a version of MCMC sampling that would obtain a uniform sampling. For interactive sampling, we fix the $feedback\_count$ to 60 for the itemset datasets i.e. Mushroom, chess, connect, eBay and 40 for the graph datasets i.e. Mutagenicity-I and II. Our comparison metric in this experiment is sampler's precision, which is defined in the description of the previous experiment. The result is shown in Table 4; for all the datasets the interactive sampling significantly improves the sampler's precision. Hasan et.al [19] propose the uniform sampling of frequent patterns as a mechanism of frequent pattern summarization; however the above experiment shows that while mining patterns from a hidden data, an interactive approach will provide more rewarding sampling experience.

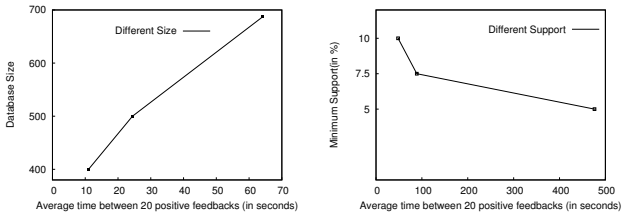
6.5 Timing Analysis

In this experiment, we analyze the execution time of this algorithm for varying minimum support and dataset size [7]. We run our algorithm with Mutagenicity-II dataset with different minimum supports and different database size and compute average time interval between 20 consecutive positive feedbacks. Figure 6 shows that, as expected, the time between successive positive feedbacks increases almost linearly with an increase in the database size or a decrease in the support value; for an example, user's waiting time between successive positive feedback is 24 (480/20) seconds for 5% support, but it is just 3 seconds for 10% support. However, the reader should note that, majority of the time is spent in computing the state-transition graph for MH-based subgraph sampling, so for larger dataset, the sampler could pre-compute the state-transition graph (for typical support threshold) and save it for future use; in that case, the timing performance of the interactive sampling will improve substantially. We conduct an experiment with Mutagenicity-II dataset. Without pre-computation, the execution of 1 million iterations takes 1676 seconds, but the same job executes in only 230 seconds, if we pre-compute the state-transition graph.

6.6 Real-life utility of Interactive Mining: Results from eBay Datasets

In this experiment, the user imitates a seller at eBay marketplace, who is interested to find related queries in the smart-phone, tablet, and game console categories. To accomplish this, the seller performs an interactive pattern mining on the eBay query dataset by providing positive feedback on frequent patterns that are related to the above product categories; some of the frequent patterns that she samples

[7]This experiment is performed in a 2GHz dual-core machine with a 2 GB physical memory.



(a) (b)

Figure 6: Timing performance of IPM w.r.t different database size and minimum support for Mutagenicity-II dataset

google tablet, android tablet, tablet pc 10
iphone 4, iphone 3gs unlocked
samsung galaxy s, samsung vibrant, samsung captivate

Table 5: Some frequent Patterns from eBay Query Dataset

from the hidden user session dataset is shown in Table 5. These patterns can educate her about the buyer demand trend at eBay marketplace; further, they could assist her in choosing a more rewarding title for her product, which would boost her chance to market it. At eBay, a seller's item in surfaced on the search result page if the title of the item contains *all* the words in a user's query [8]. If the seller wants to sell a `samsung galaxy s` smart-phone, the knowledge of the pattern in the third row of Table 5 would encourage her to add the words `vibrant` and `captivate` with the title phrase `samsung galaxy s`, so that her product would also surface against the queries like `samsung vibrant` and `samsung captivate`. Readers who are up-to-date on the latest smart-phone market may know that these two words are chosen by the US cell phone career T-mobile and AT&T to market the `samsung galaxy s` phone with their phone plans. Similarly, a title like, `google tablet android` would match with more user queries than those that would match with the title `google tablet`. In summary, by educating a seller through interactive pattern mining on hidden user session dataset, the marketplace can increase sales transactions and eventually can boost their own revenue.

7. DISCUSSION AND CONCLUSION

This is a novel research problem and our work is just the beginning, so the opportunity of future work is abundant. For example, in this work we have considered a semi-honest model, whereas in real-life a malicious model may be needed. Specifically, an interesting follow-up research could be to understand how to safeguard the interactive mining system from a malicious user who intend to reconstruct the entire dataset by exploiting the sampling system; even more interesting is the scenario when a group of malicious users jointly attack the system to achieve a similar goal. Our work considers a binary feedback, an important future work is to consider a richer interaction model between the sampler and the user. One option is that the sampler returns a set of (random) patterns and the user provides a preference-based ranking of those patterns as her feedback. The sampler then updates the sampling distribution by utilizing this feedback.

Mining interesting patterns from a hidden dataset is an important research task; an effective solution to this task

[8]however, eBay allows 55 characters for the title, so experience sellers always optimize the title of their product

brings information to its end-user, thus democratize the information for its better usages. In this paper, we show examples from eCommerce domain, but such a solution will be useful in many application domains; for instance, in heath informatics research, private mining of patient health record located at a remote server will enable a large number of researchers to find interesting patterns involving diseases, and medicines.

To conclude, in this paper, we introduce a new research problem, i.e., mining frequent pattern from a hidden dataset through interactive pattern sampling. We formulate a solution to this problem by adopting interactive pattern sampling framework, where the effective sampling distribution is continuously updated by binary feedback from the user. We provide real-life examples to illustrate the usefulness of this research task and also show experimental results to validate the feasibility and effectiveness of our solution. Specifically, we show that the number of feedback that a user is required to provide is practical for a real-life scenario.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of VLDB*, pages 487–499, 1994.

[2] I. Benjamini, G. Kozma, and W. N. The mixing time of the giant component of a random graph. 2006.

[3] T. D. Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*.

[4] M. Boley and H. Grosskreutz. Approximating the number of frequent sets in dense data. *Knowledge and Information Systems*, 21:65–89, 2009.

[5] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Exante: Anticipated data reduction in constrained pattern mining. In *Proc. of the 4th PKDD*, pages 59–70, 2003.

[6] S. Bringmann, A. Zimmermann, L. Raedt, and S. Nijssen. Don't be afraid of simpler pattern. In *PKDD*, pages 55–66, 2004.

[7] C. Bucila, J. Gehrke, and D. K. andW. White. Dualminer: A dual-pruning algorithm for itemsets withconstraints. *Data Mining and Knowledge Discovery*, 2003.

[8] T. K. Chia, K. C. Sim, H. Li, and H. T. Ng. A lattice-based approach to query-by-example spoken document retrieval. In *Proc. of the 31st ACM SIGIR conference on Research and development in information retrieval*, pages 363–370, 2008.

[9] R. K. Chung. *Spectral Graph Theory*. Americal Mathematical Society, 1997.

[10] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *Proceedings of the 2010 international conference on Management of data*, pages 855–866, 2010.

[11] A. Dasgupta, N. Zhang, and G. Das. Leveraging count information in sampling hidden databases. In *ICDE '09*, pages 329 –340, 2009.

[12] A. Dasgupta, N. Zhang, and G. Das. Turbo-charging hidden database samplers with overflowing queries and skew reduction. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 51–62, 2010.

[13] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proc. of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, 2003.

[14] B. Goethals, S. Moens, and J. Vreeken. Mime: A framework for interactive visual pattern mining. In *ECML*, pages 757 –760, 2011.

[15] O. Goldreich. *Foundations of Cryptography Volume II Basic Applications*. Cambridge University Press, 2004.

[16] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8, Jan. 2004.

[17] M. Hasan and M. Zaki. Uniform sampling of k maximal patterns. In *SIAM Data Mining*, 2009.

[18] M. A. Hasan, N. Parikh, G. Singh, and N. Sundaresan. Query suggestion for e-commerce sites. In *Proc. of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 765–774, 2011.

[19] M. A. Hasan and M. J. Zaki. Output space sampling for graph patterns. In *Proc. International Conference on Very Large Data Bases (VLDB)*, pages 730–741, 2009.

[20] S. Jagabathula, N. Mishra, and S. Gollapudi. Shopping for products you don't know you need. In *Proc. of the fourth ACM international conference on Web search and data mining*, pages 705–714, 2011.

[21] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *In Proc. of ICDM'01*.

[22] M. Mampaey, N. Tatti, and J. Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *Proc. of the 17th ACM SIGKDD*.

[23] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29, 2004.

[24] N. Mishra, R. Saha Roy, N. Ganguly, S. Laxman, and M. Choudhury. Unsupervised query segmentation using only query logs. In *Proc. of the 20th international conference companion on WWW'11*, pages 91–92.

[25] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proc. of the 27th VLDB*, pages 129–138, 2001.

[26] P. N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proc.of SIGKDD*, pages 32–41, 2002.

[27] Y. Wang and X. Wu. Approximate inverse frequent itemset mining: Privacy, complexity, and approximation. In *Proc. of the 5th ICDM*, 2005.

[28] D. Xin, X. Shen, Q. Mei, and J. Han. Discovering interesting patterns through user's interactive feedback. In *Proc. of the 12th ACM SIGKDD*, 2006.

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

[30] M. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning, 42*, 2001.

[31] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. In *IEEE Tans. on knowledge and data engineering, 17:8*, 2005.