Output Space Sampling for Graph Patterns *

Mohammad Al Hasan
Rensselaer Polytechnic Institute
110, 8th Street
Troy, NY
alhasan@cs.rpi.edu

Mohammed J. Zaki
Rensselaer Polytechnic Institute
110, 8th Street
Troy, NY
zaki@cs.rpi.edu

ABSTRACT

Recent interest in graph pattern mining has shifted from finding all frequent subgraphs to obtaining a small subset of frequent subgraphs that are representative, discriminative or significant. The main motivation behind that is to cope with the scalability problem that the graph mining algorithms suffer when mining databases of large graphs. Another motivation is to obtain a succinct output set that is informative and useful. In the same spirit, researchers also proposed sampling based algorithms that sample the output space of the frequent patterns to obtain representative subgraphs. In this work, we propose a generic sampling framework that is based on Metropolis-Hastings algorithm to sample the output space of frequent subgraphs. Our experiments on various sampling strategies show the versatility, utility and efficiency of the proposed sampling approach.

1. INTRODUCTION

Interest in graph mining has recently been extended to several interdisciplinary domains, like in cheminformatics [18], bioinformatics [13], medical informatics [1], social sciences [3], etc. Except cheminformatics where the graphs are small and sparse (except aromatic rings, most of the chemical graphs are cycle-free), graphs in these domains are large and dense, for which traditional graph mining algorithms do not scale. For example, we ran graph mining algorithms (DMTL [8], gaston [24]) on a small cell-graph [1] dataset that contains only 30 graphs with an average vertex count of 2184 and an average edge count of 36945. For this graph dataset, none of the existing algorithms could finish in 2 full days for 50% support on a dual-core 2.2 GHz machine with 2GB of memory. [7] also reported similar problem with protein interaction network graphs. This motivates the need to find algorithms that can find a small set of interesting and useful patterns instead of attempting to enumerate the entire set of patterns.

The motivation of finding smaller number of patterns also comes from the *information overload* problem which is caused by the large output set of subgraphs. Remedy to this problem is sought in *compression* or *summarization* algorithms that obtain a succinct set of frequent patterns that are representative (in the sense of cluster center) [32, 17], or from a statistical summary [33, 31] of the entire pattern-set. However, the majority of these algorithms target only the itemset pattern. In the graph domain, Yan et. al. [34] proposed an algorithm, which employs *structural proximity* and *frequency descending mining* to reduce the searchable portion of the candidate subgraph space to obtain a small set of subgraph patterns with higher discriminatory score. Very recently, a greedy subgraph feature selection algorithm, named CORK [23] was proposed. It is embedded in the gSpan [35] mining process and it can provide an approximation guaranty with respect to a sub-modular quality criteria. Though these approaches solve the information overload problem by applying efficient pruning criteria, it is not clear how scalable they are for databases of large graphs. Furthermore, they are not generic in the sense that when the interestingness criteria of the subgraphs changes, the subgraph space pruning criteria of these algorithms may have to be redesigned to reflect the change.

In subgraph pattern mining, the interestingness of a subgraph is defined based on the anticipated usages of the pattern. For exploratory data mining, frequent subgraphs with sufficiently high support suffice; for classification task, high quality discriminatory subgraphs are desirable, etc. In [27], authors list twenty-one different interestingness measures; though they are formulated with the set pattern in mind, many of them can easily be adapted for subgraph patterns. Frequency based interestingness (generally known as *minimum support*) is the most popular in pattern mining because of its anti-monotonicity; but unfortunately, many other interestingness measures do not have this property. For instance, measures like chi-square significance are neither monotonic nor anti-monotonic (however, they are convex as proved by [21]). For such interestingness measures, the candidate sub-graph search space cannot be effectively pruned with a minimum threshold value, since unlike the case of frequency it can happen that a specialization of an un-interesting pattern can turn out to be interesting and vice-versa. So, the search space is very large and special pruning mechanisms specifically tailored for these measures are required.

In this paper, we propose the notion of *output space sam-*

*This work is supported in part by NSF Grants EMT-0829835, and CNS-0103708, and NIH Grant 1R01EB0080161-01A1

VLDB '09, August 24-28, 2009, Lyon, France

pling, which samples interesting subgraph patterns without enumerating the entire set of candidate frequent patterns. The sampling algorithm performs a random walk on the candidate subgraph partial order and returns subgraph samples when the walk converges to a desired stationary distribution. The stationary distribution is chosen based on the interestingness of the subgraphs in the sample space, i.e., we want to obtain samples from a distribution that matches a predefined distribution. For instance, one may want to sample a set of frequent patterns from a uniform distribution whereas another may want to sample from a preferential distribution where patterns with higher significance score assume proportionally high generation (visitation) probability.

Output space sampling has three significant benefits. First, it is scalable in the sense that obtaining k samples is much cheaper than running the mining algorithm in a complete manner to obtain all qualified (say, frequent or significant) patterns. Thus, it immediately solves the *lack of scalability* and *information overload* problem. The output sample set also contains high quality patterns with a statistical guaranty. Second, the algorithm is generic in terms of *interestingness* criteria and the type of the pattern. Since the interestingness is defined as a function that takes a pattern and returns a numeric score value, changing the function i.e., the interestingness measure, does not alter the algorithmic framework. This generic nature gives a user the power to try different sampling distributions to further evaluate the interestingness measures so that she can find the one that best suites the application. Finally, the algorithm is immediately parallelizable, as by running m different random walks simultaneously, we can practically obtain m-fold speed-up, as there is no data or process dependency among these walks. This is a very desirable feature in the light of the recent trend in data mining where massive efforts are being made to exert the full benefit of multiple cores of modern CPUs [20, 5].

Our work has the following contributions:

- We propose the idea of *output space sampling* in the domain of frequent subgraph mining.
- The sampling approach that we propose is generic and is equally applicable to different kinds of patterns.
- We make extensive experiments to prove the sampling quality and the algorithm's performance and effectiveness on large real-life graphs.

2. RELATED WORK

In pattern mining, researchers proposed numerous algorithms [19, 35, 14, 24] to mine the complete set of frequent subgraphs. There are also algorithms for mining maximal [15] and closed [36] frequent subgraphs. These algorithms are efficient for datasets of small to moderate sized graph (upto few hundred vertices), but they do not scale well for datasets of larger graphs as demonstrated in [7]. The latter defines pattern representativeness and proposes a randomized algorithm called ORIGAMI, which mines a representative set of frequent subgraphs instead of the complete set. Thus it solves the lack of scalability problem. Authors in [7] further argued that the majority of knowledge mining applications for frequent subgraphs do not require the entire frequent pattern-set.

Beside *lack of scalability*, frequent subgraph mining methods also face *information overload* problem due to the size of the output space (number of frequent patterns) which prompted several researchers to propose algorithms for summarization of frequent patterns [32, 33, 31, 17, 34]. Though majority of these algorithms consider the summarization of itemset patterns, [34, 23] consider graph patterns. In another recent work in the graph domain, Hasan et. al. proposed MUSK [12], a sampling algorithm to uniformly sample maximal frequent subgraphs that uses Markov Chain Monte Carlo (MCMC) algorithm. MUSK, though proposed as a frequent subgraph summarization algorithm, is one of the first algorithms that aims to sample the output space of patterns; however, the sampling is limited to the maximal patterns only; so the approach is very different than our approach. In MUSK, the authors employ the idea of random walk on a weighted graph so that the sum of the weights associated to a maximal pattern is constant. But, in this research, we use the Metropolis-Hastings sampling, which is based on proposal distribution and sample rejection. Besides MUSK, the closest work that we found is the one by Boley et.al [2] which is proposed very recently. It uses a randomized algorithm to uniformly sample frequent itemsets. The objective of their work is to approximately estimate the size of the output space by taking only polynomial number of samples. But, the sampling algorithm that they propose is very specific and works only for the itemset pattern.

Though sampling output space is relatively novel, sampling input space i.e., the data, has a long history in frequent pattern mining [28, 9, 6]. These algorithms generally find prospective candidate patterns from a small sample of the entire transaction-set, so that the *support counting* can be performed efficiently. Sometimes, the frequent patterns in these methods are frequent only in the statistical sense. However, sampling output space is much more difficult than sampling the input space, since the size and the elements of the output space is not available immediately.

In our sampling algorithm, we use the Metropolis-Hastings [25] framework. We found another recent work [16] that uses Metropolis sampling in the domain of graph mining. The objective of their work is to obtain a model that finds subgraphs that approximate a given degree distribution. They use Metropolis algorithm and Simulated Annealing (SA) to solve the problem.

3. PROBLEM FORMULATION

A subgraph mining algorithm accepts a graph database, $\mathcal{D}$, an interestingness function, $score : \mathcal{F} \rightarrow \mathbb{R}$, that maps a subgraph to a numerical qualitative (interestingness) score and a numeric value, *threshold*, that denotes the specific minimum value for the interestingness score. The output set of the algorithm is all possible subgraphs of the database graphs whose *score* exceeds the given *threshold*. In case of frequent subgraph mining, the *score* of a subgraph g is the *frequency* of g; where frequency is denoted by the number of the graphs in $\mathcal{D}$ where g occurs and the *threshold* value is called the *minimum support*. This is the popular frequent graph mining problem. But, one may also provide different *score* function or even a conjunction of multiple *score* functions together with multiple threshold values.

The above formulation poses the subgraph mining as a constraint search problem, which has an input space which is the span of the database graphs and a feasible output

space which is the span of the interesting subgraphs. Here, we abuse the linear algebraic term, *span* to denote the combinatorial subgraph space. This space grows exponentially as the graphs in the input space become larger. Now, for a given problem instance, by *sampling output space*, we mean to sample one feasible subgraph from a user-specified discrete distribution. In case, the user wants to sample in proportion to the interestingness score, the discrete distribution can be constructed from the interestingness value of all the feasible subgraphs in the output space. For example, in case of frequent subgraph mining, if the user specifies that the interestingness score of a subgraph is its support value, then the normalized vector (to make it a probability vector) of the support values of all frequent subgraphs is the desired sampling distribution. One may also want to choose a uniform distribution, for which the probability of choosing each frequent subgraph is equal to $\frac{1}{|\mathcal{F}|}$, where $|\mathcal{F}|$ is the total number of frequent subgraphs.

3.1 Challenges

The main challenge of output space sampling in the pattern mining domain is that the search space is not immediately available. As we outline in the introductory Section, it is also infeasible to enumerate all the subgraphs in the feasible search space; after all, it is this infeasibility that motivates someone to embrace sampling. Here, "enumerate" means to find the occurrence list (named as *gidset* in Section 4.2) of the subgraph by performing the support counting step. This is the costliest task when the database graphs are large. The second challenge comes from the sampling objective that aims to sample from a given distribution. Note that we do not have the *score* value of *all* the frequent subgraphs in the feasible search space; even worse, we do not know the size of the search space. The only tool that we have is that after we enumerate a subgraph we can immediately calculate its interestingness score from the support-list. The sampling [1] probability of this subgraph g that we want is $\frac{score(g)}{Z}$, where Z is the normalizing constant, whose value is equal to $\sum_{f \in \mathcal{F}} score(f)$.

Problems of the above nature generally arise in statistical physics while analyzing dynamic systems (for instance, finding low energy state in molecular simulation) and they are solved by an elegant method named Metropolis-Hastings (MH) [25]. It is a Monte Carlo Markov Chain (MCMC) algorithm that performs a random walk on the search space with a locally computable probability transition matrix. We also use the MH algorithm for output space sampling, but adapt it appropriately based on our sampling requirements.

Our choice of MH framework for output space sampling has some further justifications. For example, due to the Markovian nature of this random walk, it does not remember earlier states of the random walk; so the algorithm consumes much less memory for the exploration purpose. On the other hand, this is a problem for traditional subgraph mining algorithms (for example, DMTL[8], Gaston[24]) that store embeddings of the frequent patterns. For large datasets, many of these algorithms terminate after exhausting the virtual memory.

[1]sampling and enumeration are different, as it happens frequently that we enumerate a pattern and then decide not to visit (sample) that pattern

4. BACKGROUND

In this section, we define several key concepts that will be used throughout the paper.

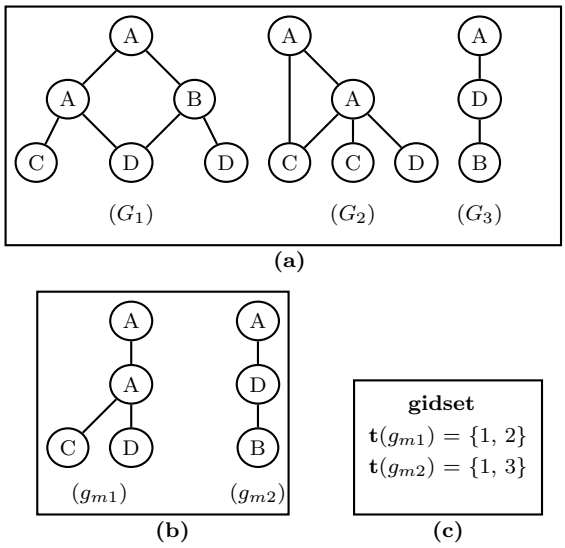


Figure 1: (a) Graph database with 3 graphs (b) Maximal frequent graphs of (a) with support 2 (c) gidset of the maximal graphs g_{m1} and g_{m2}

4.1 Graphs and Subgraphs

A graph $G = (V, E)$, consists of a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, and a set of edges $E = \{(v_i, v_j) : v_i, v_j \in V\}$. Let L_V and L_E be the set of vertex and edge labels, respectively, and let $\mathcal{V} : V \rightarrow L_V$ and $\mathcal{E} : E \rightarrow L_E$ be the labeling functions that assign labels to each vertex and edge. The *size* of a graph G, denoted $|G|$ is the cardinality of the edge set (i.e., $|G| = |E|$). A graph of size k is also called a k-graph. A graph is *connected* if each vertex in the graph can be reached from any other vertex. All (sub)graphs we consider are undirected, connected and labeled.

A graph $G_1 = (V_1, E_1)$ is a *subgraph* of another graph $G_2 = (V_2, E_2)$, denoted $G_1 \subseteq G_2$, if there exists a 1-1 mapping $f : V_1 \rightarrow V_2$, such that $(v_i, v_j) \in E_1$ implies $(f(v_i), f(v_j)) \in E_2$. Further, f preserves vertex labels, i.e., $\mathcal{V}(v) = \mathcal{V}(f(v)))$, and preserves edge labels, i.e., $\mathcal{E}(v_1, v_2) = \mathcal{E}(f(v_1), f(v_2))$. f is also called a *subgraph isomorphism* from G_1 to G_2. If $G_1 \subseteq G_2$, we also say that G_2 is a super-graph of G_1. Note also that two graphs G_1 and G_2 are *isomorphic iff* $G_1 \subseteq G_2$ and $G_2 \subseteq G_1$. Let $\mathcal{D}$ be a set of graphs, then we write $G \subseteq \mathcal{D}$ if $\forall D_i \in \mathcal{D}, G \subseteq D_i$. G is said to be a *maximal common subgraph* of $\mathcal{D}$ iff $G \subseteq \mathcal{D}$, and $\nexists H \supset G$, such that $H \subseteq \mathcal{D}$.

4.2 Mining Frequent Graphs

Let $\mathcal{D}$ be a database (a multiset) of graphs, and let each graph $D_i \in \mathcal{D}$ have a unique graph identifier. Denote by $\mathbf{t}(G) = \{i : G \subseteq D_i \in \mathcal{D}\}$, the *graph identifier set (gidset)*, which consists of all graphs in $\mathcal{D}$ that contain a subgraph isomorphic to G. The *support* of a graph G in $\mathcal{D}$ is then given as $|\mathbf{t}(G)|$, and G is called *frequent* if $|\mathbf{t}(G)| \geq \pi^{\min}$, where $\pi^{\min}$ is a user-specified minimum support (*minsup*) threshold. A frequent graph is *closed* if it has no frequent super-graph with the same support. A frequent graph is *maximal*

if it has no frequent super-graph. Denote by $\mathcal{F}, \mathcal{C}, \mathcal{M}$ the set of all frequent, all closed frequent, and all maximal frequent subgraphs, respectively. By definition, $\mathcal{F} \supseteq \mathcal{C} \supseteq \mathcal{M}$. Fig. 1($a$) shows a database with 3 graphs. With a minimum support $\pi^{\min} = 2$, there are nine frequent and two maximal frequent graphs; the latter set is shown in Fig. 1(b); their corresponding gidsets are shown in Fig. 1(c). All possible (connected) subgraphs of the maximal frequent graphs are frequent.

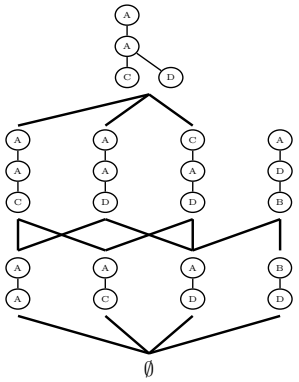


Figure 2: Partial Order Graph of Frequent Subgraphs

4.3 Frequent Graph Partial Order

The set of all frequent subgraphs forms a partial order with respect to the subgraph relationship, $\subseteq$, which is referred to as the *partial order graph (POG)*. Every node in POG corresponds to a distinct frequent graph pattern, i.e., each graph in POG is the canonical representative (with the minimal DFS code [35]) for all other graphs isomorphic to it. Every edge in POG represents a possible extension of a frequent pattern to a larger (by one edge) frequent pattern. The maximal elements in POG correspond to $\mathcal{M}$. The bottom element in the partial order is the empty graph (which is frequent by default). Algorithms for enumerating all frequent subgraphs typically traverse the POG in either depth-first or breadth-first manner, starting from the bottom. Since, a graph can be constructed in many different ways (depending on the order in which edges are added), starting from the empty pattern, there are multiple paths leading to a node in the partial order. Thus different nodes in the POG have different degrees. We use d_g to denote the degree of a node g in the POG graph. Figure 2 shows the POG for the example data in Figure 1.

4.4 Uniform Sampling

Consider a problem instance that has many feasible solutions. For this, *uniform sampling* is an algorithmic process which returns one of these solutions uniformly at random. For a frequent graph mining problem instance which consists of an input graph database and a user defined minimum support, the problem of uniform sampling would return one frequent subgraph out of all frequent subgraphs (feasible solutions) with a uniform probability. Such sampling is interesting when the enumeration of the entire sample space is infeasible; most likely reasons for that might be the enormous size of the sample space or the ineffectiveness of the algorithm that enumerates the objects from that sample space.

4.5 Markov Chains

A Markov chain is a discrete-time stochastic process defined over a set of states S, in terms of a matrix P of *transition probabilities*. The set S is either finite or countably infinite. The transition probability matrix P has one row and one column for each state in S. The Markov chain is in one state at any time, making state-transitions at discrete time-stamps $t = 1, 2, \ldots$ and so on. The entry $P(i, j)$ in the transition probability matrix is the probability that the next state will be j, given that the current state is i. For all $i, j \in S$, we have $0 \leq P(i, j) \leq 1$, and $\sum_j P(i, j) = 1$, i.e., all the rows add up to 1.

A *stationary distribution* for the Markov chain with transition matrix P is a probability distribution π, such that:

$$\pi = \pi P \tag{1}$$

Here π is a **row-vector** of size $|S|$. Thus, the stationary distribution is the left eigen-vector of the matrix P with an eigenvalue of 1. We use $\pi(i)$ to denote the i'th component of this vector. A Markov chain is reversible if it satisfies the *detailed balance equation* below:

$$\pi(u)P(u, v) = \pi(v)P(v, u), \forall u, v \in S \tag{2}$$

Reversibility is a sufficient, but not necessary condition for π to be a stationary distribution of the Markov chain. A Markov chain is ergodic if it has a stationary distribution.

If the state space S of a Markov chain is the set V of a graph $G(V, E)$, and if for any two vertices $u, v \in V$, $(u, v) \notin E$ implies that $P(u, v) = 0$, then the process is also called a *random walk* on the graph G. In other words, in a random walk on a graph, the state transitions occur only between the adjacent vertices.

4.6 Metropolis-Hastings (MH) Algorithm

The objective of the MH algorithm is to sample with a target distribution. The main idea is to simulate a Markov chain such that the stationary distribution of this chain coincides with the target distribution [25]. Assume that we want to generate a random variable X taking values in $\mathscr{X} = \{1, \ldots, n\}$, according to a target distribution π, with

$$\pi(i) = \frac{b_i}{C}, \quad i \in \mathscr{X} \tag{3}$$

where it is assumed that all b_i are strictly positive, n is large, and the normalizing constant $C = \sum_{i=1}^{n} b_i$ is difficult to calculate. MH first constructs an n-state Markov chain $X_t, t = 0, 1, \ldots$ on $\mathscr{X}$ whose evolution relies on an arbitrary transition matrix $\mathrm{Q} = (q_{ij})$ in the following way:

- When $X_t = i$, generate a random variable Y satisfying $\mathbb{P}(Y = j) = q_{ij}, \quad j \in \mathscr{X}$

- If $Y = j$, let

$$X_{t+1} = \begin{cases} j & \text{with probability } \alpha_{ij} \\ i & \text{with probability } 1 - \alpha_{ij} \end{cases} \tag{4}$$

  Where,

$$\alpha_{ij} = \min\left\{\frac{\pi(j)\ q_{ji}}{\pi(i)\ q_{ij}}, 1\right\} = \min\left\{\frac{b_j\ q_{ji}}{b_i\ q_{ij}}, 1\right\} \tag{5}$$

It follows that $\{X_t, t = 0, 1, \ldots\}$ has a one-step transition probability matrix P, given by

$$P(i,j) = \begin{cases} q_{ij}\ \alpha_{ij}, & \text{if } i \neq j \\ 1 - \sum_{k \neq i} q_{ik}\ \alpha_{ik}, & \text{if } i = j \end{cases} \quad (6)$$

For the above P, the Markov chain is reversible and has a stationary distribution π, equal to the target distribution. Here, Q and α_{ij} are called proposal distribution, and acceptance probability respectively.

5. SAMPLING ALGORITHMS

In this section, we show different sampling algorithms for sampling the output space of the frequent subgraph patterns.

State Space of the Random Walk: The frequent pattern partial order graph (POG) works as the state space of the Markov chain on which the sampling algorithms run their simulation. Unlike traditional graph mining algorithms, like gSpan [35], or DMTL [8], they walk on the full edge-set of the POG. Put another way, candidate generation in sampling algorithms allows all possible one-edge extensions without restricting them to be on the right-most path [35, 8]. The important point to note here is that the algorithms construct the partial order graphs locally around the current node. If the current node represents pattern p, its neighbors consist of nodes corresponding to all frequent super-patterns that have one more edge than p, and all sub-patterns that have one-edge less than p. The random walk chooses one neighbor (super- or sub-pattern) according to its transition probability (which varies based on the desired sampling distribution). The local construction of POG is important as it avoids the construction of the entire POG, which would require finding all the frequent patterns.

However, we still need to prove one crucial fact that we can design a random walk on POG that is indeed ergodic. The following Lemma holds.

LEMMA 5.1. *The random walk on POG as defined above converges to a stationary distribution.*

PROOF. To achieve an stationary distribution, a random walk needs to be finite, irreducible, and aperiodic [22]. First, POG is finite since the number of frequent patterns is finite. Second, for any two nodes u and v in POG, there exists a positive probability to reach from one to other, since every pattern can reach and can in turn be reached from the $\emptyset$ pattern. Since at least one path exists between any two patterns via the $\emptyset$ pattern, the random walk is irreducible. Third, the POG is a layered graph (each layer contains patterns of the same size), so one can make it a bipartite graph by accumulating the vertices from alternate layers in one partition; thus, the random walk on POG can be periodic. However, such periodicity can easily be removed by adding a self-loop with probability $\frac{1}{2}$ at every vertex of the POG [26]. Thus the claim is proved. □

5.1 Convergence rate of random walk

One important aspect of any MCMC algorithm (including MH, which is essentially a special kind of MCMC algorithm) is the rate at which the initial distribution converges to the desired distribution. The convergence rate of a random walk has been studied extensively in spectral graph theory [10], since it plays an important role in obtaining efficient MCMC algorithms. A Markov chain is called *rapidly mixing* if it is close to stationary after only a polynomial number of simulation steps, i. e., after $poly(\lg n)$. Note that, n (the number of states) can be exponentially large with respect to the algorithm input. An algorithm that is *rapidly mixing* is considered efficient.

A method to measure the convergence rate is to find the *spectral gap* of the transition probability matrix P. P has n real eigenvalues $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n-1} \geq -1$. Then, the *spectral gap* is defined as $\lambda = 1 - \max\{\lambda_1, |\lambda_{n-1}|\}$. Since the absolute values of all the eigenvalues are less than one with the largest eigenvalue λ_0 be exactly one, the spectral gap is always between 0 and 1. The higher the spectral gap, the faster the convergence [11]. For output space sampling, the entire P is not available to us, so it is generally difficult to measure the spectral gap.

However, convergence rate can also be approximated by direct simulation, using the *variation distance* measure. The variation distance at time t with initial state x, $\Delta_x(t)$ is defined as the statistical difference between distribution $P^t(x, .)$ and $\pi(.)$, which is: $\frac{1}{2}\sum_{y \in \Omega} |P^t(x,y) - \pi(y)|$ [11].

5.2 Uniform Sampling of Frequent Patterns

Our formal goal is to simply obtain a uniform sample of the set of all frequent subgraphs. In this sampling, the feasible set consists of all subgraph patterns in the POG that are frequent for a user defined *minsup* value. This sampling is very useful to get some intuitive estimates about the output space of the frequent patterns. One main problem in any frequent pattern mining is the choice of support value [37]. By taking a small number of frequent patterns with uniform sampling for different support values, a user can obtain a quick estimate of the average pattern-sizes for different supports, which may help him to choose a suitable support value. In [2], the authors recently showed how uniform sampling can be used to obtain an approximate count of the size of $\mathcal{F}$ by taking only a polynomial number of samples (but for itemsets only).

Below we show, how to choose the transition probability matrix to obtain uniform samples of frequent graph patterns from the output space.

5.2.1 Computing Transition Probability Matrix

A general random walk that selects each outgoing edge with equal probability does not achieve the desired uniform distribution, since different frequent patterns are of different sizes, and consequently the number of neighbors (in the POG) adjacent to a frequent pattern can vary a lot. Thus, a frequent pattern is oversampled if the corresponding node in the POG has a high degree compared to another frequent pattern with lower degree. In fact, the stationary distribution of a node is directly proportional to the degree of that node [10].

To obtain a uniform stationary distribution, we need to modify the probability transition matrix to compensate for the different values for the degrees of different nodes of POG. The following Lemma (mentioned in exercise 6.9 of [22]) is useful in this regard. The proof is simple that we provide below.

LEMMA 5.2. *An ergodic random walk achieves a uniform stationary distribution if and only if its transition probability matrix is doubly stochastic.*

PROOF. If π_u is a row vector that defines uniform probability distribution of a random walk with n states and P is the appropriate transition probability matrix, we have from Equation (1), $\pi_u = \pi_u P$. Since, there are n states, from the uniformity assumption, $\pi_u = (\frac{1}{n})\mathbf{1^T}$; substituting this in the above equation, we obtain $\frac{1}{n}\mathbf{1^T} = \frac{1}{n}\mathbf{1^T}P$. This means that the sum of the column vectors of P is equal to 1 for each column of the matrix, i.e., matrix P is *column stochastic* and furthermore, since P is a transition probability matrix, it is *doubly stochastic*. This proves the "only if" part and since an ergodic random walk has a unique stationary distribution, the reverse also holds. □

An easy way to obtain a doubly stochastic transition probability matrix is to make the matrix symmetric, i.e. to make $P = P^T$. A symmetric matrix P is row stochastic because it is transition probability matrix and by symmetry it is also column stochastic. Now, to sample frequent pattern uniformly by performing a random walk on the POG, we can use the following probability matrix, P: (see the footnote [2])

$$P(u,v) = \begin{cases} \frac{1}{max(d_u, d_v)} & \text{if } u \neq v \text{ and } v \in \text{adj}(u) \\ 1 - \sum_{x \in \text{adj}(u)} P(u,x) & \text{if } u = v \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

Here, u and v are two arbitrary nodes in POG, and d_x denotes the degree of the node x and adj(x) is the set of neighbors of a node x. If u and v are two neighboring nodes, $P(u,v)$ and $P(v,u)$ is equal to $min\{\frac{1}{d_u}, \frac{1}{d_v}\}$, which makes P a symmetric matrix. A self-loop of appropriate probability ensures that the matrix P remains row stochastic. So, based on Lemma 5.2 the above random walk obtains a uniform sampling of the frequent patterns, given that we can design an ergodic random walk on POG. Also note that the random walk is reversible as is satisfies the balance Equation (2).

The above algorithm is just an adaptation of classical Metropolis-Hastings (MH) algorithm where the proposal distribution is same as the transition probability matrix of a general random walk (each neighbor is chosen uniformly, $q(x,y) = 1/d_x$) and the acceptance probability, as suggested by the MH algorithm, is:

$$\alpha_{xy} = \min\left\{\frac{\pi(x)\ d_x}{\pi(y)\ d_y}, 1\right\} = \min\left\{\frac{d_x}{d_y}, 1\right\}$$

Substituting the value of α_{xy} and q_{xy} in Equation (6) we can obtain the Equation (7).

5.2.2 Algorithm

Figure 3 describes the algorithm for uniform frequent subgraph mining. It accepts the graph database ($\mathcal{D}$), minimum support value (*minsup*), and the minimum number of steps (*miniter*) required for the random walk to mix to its stationary distribution. The higher the *miniter* parameter, the better is the uniform sampling since the random walk has better probability to converge to the uniform distribution. However, user can just set it by her timing constraint or by empirical estimation from our discussion in Section 5.1.

```
Uniform_Sampling (D, π^min, miniter):
1.  p = generate_any_frequent_pattern(D, π^min)
2.  d_p = compute_degree(p)
3.  While (true)
4.    Choose a neighbor, q, uniformly from, all possible
      frequent super and sub patterns
5.    d_q = compute_degree(q)
6.    accept_prob = min(d_p/d_q, 1)
7.    if uniform(0,1) ≤ accept_prob
8.      p = q
9.      iter = iter + 1
10.   if iter ≥ miniter
11.     return p
```

Figure 3: Uniform Sampling Algorithm

At line 1, we start with an arbitrary frequent subgraph, p. A single edge frequent subgraph suffices. Then we compute all frequent super-patterns and sub-patterns of p (line 2). The degree of the node that corresponds to the pattern p is just the size of the union of super-patterns and sub-patterns set. Then we choose a pattern from p's neighbors (line 4) with iid distribution and compute the acceptance probability in line 6. If the move is accepted, we increment the current iteration count, otherwise we choose another neighbor identically and repeat the whole process. The process terminates when the iteration count exceeds *miniter* and it returns the currently visiting pattern.

Example: In Figure 2, the subgraph $A-D-B$ (rightmost pattern in the third row from the bottom) has 2 neighbors (0 super-neighbors and 2 sub-neighbors), if the algorithm while visiting the corresponding node chooses the pattern $A-D$ (in line 4 of Figure 3), the acceptance probability (in line 5) is equal to $\frac{2}{4}$, since $A-D$ has 4 neighbors in total. In this way, the rejection step compensates for the non-uniformity in degree by rejecting moves to higher degree nodes more often and vice-versa.

5.3 Support proportional Sampling

Though the MH algorithm is generic enough to sample any distribution (at least in theory), the main challenge in using this algorithm is to adapt the proposal distribution. If the proposal distribution is very different from the desired distribution, the acceptance rates are very low and algorithm's efficiency deteriorates severely. Furthermore, the quality of the sample is also not that good as it becomes difficult to counter-balance the proposal bias by rejection. In this section, we will show how we can use MH to sample a pattern in proportion to its support value by choosing a suitable proposal distribution.

If s_g is the support of any subgraph pattern g, we want to sample a pattern g with a probability, $\frac{s_g}{C}$, where $C = \sum_{z \in \mathcal{F}} s_z$. For this task, we adapt the MH algorithm by choosing a proposal distribution that is conducive to obtain the target sampling by the random walk. The sample space is the same POG that we used for uniform sampling of fre-

[2]Considering the POG graph as layered graph, ergodicity proof of Lemma 5.1 requires the Eq. (7) to be changed as below:

$$P(u,v) = \begin{cases} \frac{1}{2*max(d_u, d_v)} & \text{if } u \neq v \text{ and } v \in \text{adj}(u) \\ \frac{3}{2} - \sum_{x \in \text{adj}(u)} P(u,x) & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}$$

However, we skip this addition in the Eq. (7) for the benefit of simplicity. Further, the addition of a self-loop does not have any effect on the uniform generation claim.

quent subgraphs.

Generally smaller subgraphs have high support, since as we walk up towards the border of the partial order, subgraphs become increasingly infrequent. So, we need to choose a proposal distribution that is biased to walk downward along the partial order more frequently than walking up. If u and v are two nodes in the POG, $\mathcal{N}_{up}(x)$ and $\mathcal{N}_{down}(x)$ are the up-neighbors (patterns with one extra edge) and down-neighbors (patterns with one edge less) of any pattern x, our proposal distribution is as follows:

$$P(u,v) = \begin{cases} \alpha \times \frac{1}{|\mathcal{N}_{up}(u)|}, & \text{if } v \in \mathcal{N}_{up}(u) \\ (1-\alpha) \times \frac{1}{|\mathcal{N}_{down}(u)|}, & \text{if } v \in \mathcal{N}_{down}(u) \end{cases} \quad (8)$$

for some $\alpha \leq 1$. If a pattern has no up-neighbors (or down-neighbors), the entire probability is distributed uniformly between its neighbors. After the selection based on the proposal distribution is performed, the acceptance probability is computed by Equation 5. We do not show any pseudo-code here as it is identical to Figure 3 except the above changes which are incorporated in Line 4-6 of the algorithm.

The main challenge in the above algorithm is to choose the right value for α. Apparently, a value of α higher than 0.5 is not desirable as it would assign larger portion of the probability mass to the up-neighbors which is not ideal to achieve a support proportional sampling. But, choosing a value which is further away from 0.5 towards 0 may also severely penalize the upward walk in the proposal sampling. The best way to find the right value of α is to choose it dynamically by analyzing the acceptance rate (as computed by Equation 5) in some trial simulations. The value of α which offers the highest acceptance rate should be selected because for this case the proposal distribution is the closest to the target distribution and better sampling quality and performance can be achieved. In the experiment Section, we show results that show the relation between α, acceptance rate, and the sampling quality. We conclude this section with an example.

Example: Let us consider the same example as in Section 5.2. Also assume that we set $\alpha = \frac{1}{3}$. If the corresponding POG nodes of the patterns $A-D-B$ and $A-D$ are p and q respectively, we have $P(p,q) = \frac{1}{2}$ and $P(q,p) = \frac{1}{3\times 3}$, in the proposal distribution. The support values of these patters are 2 and 3 respectively. So, the acceptance probability of the node q while moving from node p (after the proposal based selection is done) is $\min\{\frac{3\times\frac{1}{9}}{2\times\frac{1}{2}}, 1\} = \frac{1}{3}$

5.4 Discriminatory Subgraph Sampling

The objective of discriminatory subgraph sampling is to find subgraphs that can be used as features for graph classification. They can further be used to construct graph kernels for kernel based classification methods [30]. To sample discriminatory subgraphs, we use *delta score* (defined below) as our function, which maps each feasible subgraph in the search space to a real number. It has also been used by Yan et. al. in [34] to find significant subgraphs. For a two class dataset (each graph in $\mathcal{D}$ is either of class 1 or of class 0), *delta score* is defined as follows: If g is a feasible subgraph, we can partition its gidset, $\mathbf{t}(g)$ into two sets based on the class labels. If $\mathcal{G}_0$ and $\mathcal{G}_1$ are these two sets, the delta score of g is $abs(|\mathcal{G}_0| - |\mathcal{G}_1|)$, i.e., it is the difference between the number of database graphs of class 1 and 0 that contain g. The higher the delta score the more the pattern g is discriminatory. While mining discriminatory subgraphs, *minsup* criteria is not required, but a small minsup value can still be used to keep the feasible set relatively small by removing patterns that have very small support across the entire database.

The important consideration of sampling discriminatory subgraphs is the choice of proposal distribution. Unlike support, we have no reason to believe that the size of a pattern has anything to do with its delta score, which is neither monotonic nor anti-monotonic. But, we can intuitively assume that very large (very small) patterns are not discriminatory as they might occur in too few (too many) database graphs respectively. So, ideally we like to be in the middle of the partial order. Hence, our chosen proposal distribution for this case uniformly selects one pattern from the neighbors, which is exactly the same as in algorithm in Figure 3. This strategy samples a pattern in proportion to the degree distribution of the patterns in the POG graph (without rejection step) and we assume that the patterns in the middle of the partial order have more neighbors (sum of both up-neighbors and down-neighbors) than the patterns around the edge. Then, we compute the accept probability as the ratio of corresponding delta-scores. In other words, if the random walk at node i chooses a neighbor j by using the proposal distribution, then it accepts j with probability $\min\{\frac{\text{delta-score}(j)}{\text{delta-score}(i)}, 1\}$. Note that in the above case, we did not use the exact Equation (Equation 2) of the MH algorithm for the acceptance probability, which ensures the satisfiability of detailed balance condition. It does not have the q_{ij} and q_{ji} terms of Equation 4 where $q_i = \frac{1}{d_i}$ (and similarly $q_j = \frac{1}{d_j}$) for this particular proposal distribution. Our experiments on some real-life graph datasets show that the above acceptance probability finds few good discriminatory patterns faster (because, a move towards a pattern with a higher delta score is always accepted), though lack of consideration of the balance equation hurts the mixing rate and the long-term stationary distribution probability.

6. IMPLEMENTATION DETAILS

The objective of output sampling is to quickly obtain a small set of patterns that have the desired characteristics. For that, it is essential that the sampling algorithm enumerates as fewer patterns as possible. In our sampling algorithm, we enumerate all the neighbors of the current pattern. We do so to find the next pattern with respect to the proposal distribution and to subsequently decide whether the proposed move is accepted or not. In this section, we provide the implementation details of the neighborhood enumeration process.

6.1 Computing neighbors of a pattern

A neighbor of a frequent graph pattern g has two components, super-neighbors and sub-neighbors. Graph g_1 is a super-neighbor of g, if $g_1 \supset g$ and $g_1 = g \diamond e$, where e is a frequent edge and $\diamond$ denotes the extension operation. So, a super-neighbor is obtained by adding an edge to g. If the new edge e connects two existing vertices, we call it a back edge. Otherwise, it is called a forward edge. Adding a forward edge always adds a new vertex. To effectively compute the super-neighbors, we pre-compute a data structure called *edge_map*, that stores the frequent edges

in a map $\Phi : L_V \rightarrow \wp(L_V \times L_E)$ which essentially maps a vertex label v_l to all possible tuples of (vertex, edge) label pairs. For example, in *edge_map*, if a vertex-label A is mapped to $\{(A,a),(B,a),(B,b)\}$, then from any vertex that is labeled with A, three different edge extensions, like $(A,A,a),(A,B,a),(A,B,b)$ are possible. The new edge can be either a forward edge or a back edge.

A graph g_2 is a sub-neighbor of g, if $g_2 \subset g$ and $g = g_2 \diamond e$, for some frequent edge e. To obtain g's sub-neighbors, an edge is removed from g. Back edge removal removes an edge, but keeps the resulting graph connected and forward edge removal removes an edge and isolates exactly one vertex which is also removed from the resulting graph. For example, there are 6 sub-neighbors of graph G_1 in Fig. 1(a), that can be obtained by removing the edges $A-A$, $A-B$, $A-D$, $B-D$ (D in middle column in 3rd row), $A-C$, and $B-D$ (rightmost D in the third row) respectively; the first four are back edge removals whereas the last two are forward edge removals.

The above super-neighbor and sub-neighbor computation ensures that the random walk that the sampling algorithm adopts is reversible, i.e., for any pair of pattern p and q that assume a role of (*pattern*, *super-neighbor*) in a forward walk can also assume a (*pattern*, *sub-neighbor*) role in a walk in the reverse direction and vice versa.

6.2 Support Counting

Only the frequent patterns are part of the POG. So, while the algorithm finds a neighbor (say, q) of a pattern p, it also computes the support of q ($|\mathbf{t}(q)|$) to ensure that the pattern q is frequent, i.e., $|\mathbf{t}(q)| \geq \pi^{\min}$. Any infrequent neighbors are discarded. For support computation, we use Ullmann's subgraph isomorphism algorithm [29] with various optimizations. Associated with any frequent graph, we also store its gidset, so that the number of calls to Ullmann's algorithm is as small as possible. Below, we discuss how the **gidset** of a neighbor of a pattern p is computed from p's gidset.

If a pattern q is created from a pattern p by extending an edge e (for the case of super-neighbor), we have $\mathbf{t}(q) \subseteq \mathbf{t}(p)$, and it can be obtained as follows: (1) Intersect $\mathbf{t}(p)$ and $\mathbf{t}(e)$; (2) Perform a subgraph isomorphism test of pattern q against each graph in the result of (1). The identifiers of the database graphs that succeed the test comprise $\mathbf{t}(q)$.

If the pattern q is obtained from p by removing an edge e (for the case of sub-neighbor), $\mathbf{t}(q) \supseteq \mathbf{t}(p)$. To compute the gidset of q, we first find $\bigcap_{e \in q} \mathbf{t}(e)$, the intersection of gidset of all edges of the pattern q. It is easy to see that $\bigcap_{e \in q} \mathbf{t}(e) \supseteq \mathbf{t}(q) \supseteq \mathbf{t}(p)$. Then, we perform a subgraph isomorphism test of pattern q against each graph $g \in \bigcap_{e \in q} \mathbf{t}(e) \backslash \mathbf{t}(p)$. The identifiers of the graphs that succeed the test together with $\mathbf{t}(p)$ comprise $\mathbf{t}(q)$.

Many graph mining algorithms, such as gaston [24], DMTL [8] perform support counting by explicitly storing the embeddings of the frequent subgraphs in database graphs. This approach works because the pattern enumeration in these algorithms always constructs the child patterns by extending a parent pattern and the child's embedding list is always a subset of the embeddings of the parent pattern. On the other hand, our implementation performs a direct subgraph isomorphism checking. The reason behind it is that if the random walk visits a sub-pattern from its super-pattern, the embedding list of the sub-pattern cannot be constructed from the embedding list of the super-pattern; the earlier is a superset of the latter.

7. EMPIRICAL RESULTS

In this section, we evaluate the quality of our sampling algorithm for different sampling objectives. First, we want to experimentally evaluate how the sampling distribution matches with the desired distribution. For these experiments, we run our algorithm on smaller datasets with relatively high supports values; so, the size of $|\mathcal{F}|$ is not that large. Then, we take sufficiently large number of samples (say, $c \times |\mathcal{F}|$), so that the sampling distribution can be constructed. Note that, for such smaller datasets, running a complete subgraph mining algorithm is the best option as it would retrieve all the patterns in a shorter time, but we use them just to assess the sampling quality that we can achieve using our algorithm. Later, we show results on large datasets for which traditional graph mining algorithms fail to run.

7.1 Uniform Sampling of Frequent Subgraphs

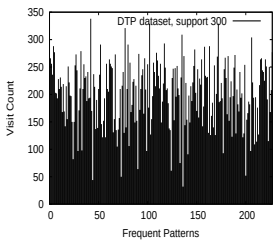
We first show the experimental results for uniform sampling of the frequent patterns. For this experiment, we use the DTP (CM) AIDS antiviral screening dataset [3], which contains 1084 graphs (confirmed moderately active chemical compounds), with an average graph size of 45 edges and 43 vertices. Note that, the majority of the chemical graphs are trees, so more than 90% of the frequent subgraphs are either trees or sequences.

Fig. 4 shows the results with minimum support=300; the exact number of frequent patterns is 227 (125 paths, 94 trees, 8 cyclic graphs). We run our uniform sampler for a total of 45,400 iterations starting from the empty pattern, so that in the case of ideal uniform generation, each frequent pattern would be generated (visited in partial order graph) 200 times. Note that, in this simulation, the patterns that we sample are dependent samples which are different from the samples one would obtain by independent runs of algorithm in Figure 3. However, since we are running very large number of simulation steps, we expect the dependency bias to diminish by the law of large numbers.

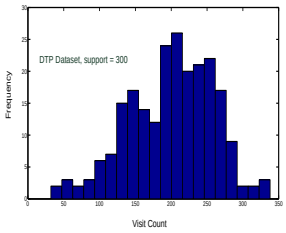
The visit count for each pattern is shown as a vertical bar in the bar chart in Figure 4(a). The chart shows that the visit is fairly uniform. Detailed statistics of visit counts is shown in Fig. 4(d). The minimum, median and the maximum of visit counts are 32, 209 and 338 respectively (mean is trivially 200). We also show the frequency histogram of visit counts in Fig. 4(b), where the x-axis shows the number of times a pattern is visited, and the y-axis shows how many patterns fall in that bin. The histogram is not much different from a normal curve, which one expects for an ideal iid distribution (more discussion on this follows). For this experiment, We also compute the variation distance [4]. In Figure 4(c) we show how it changes as the iteration progresses. After 20,000 iterations the distance value converges to 0.1 which does not reduce much in subsequent iterations. The spectral gap for this random walk is equal to 0.03 only, which suggests that the mixing rate is generally slow.

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

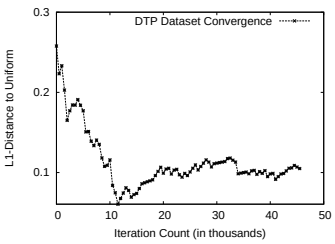
[4] In the def. of variation distance, the term $P^t(x,y)$ denotes the xth entry of state probability vector after time t which we compute empirically; also note, in our experiments y is always an empty pattern.



(a) histogram



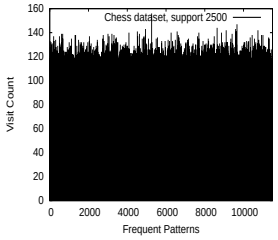
(b) individual counts



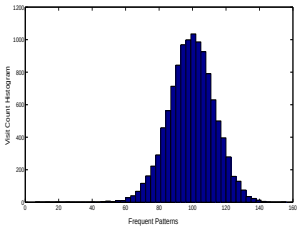
(c) Variation distance

Uniform-Sampling				Ideal	
Maximum	Minimum	Median	Std	Median	Std
338	32	209	59.02	200	14.11

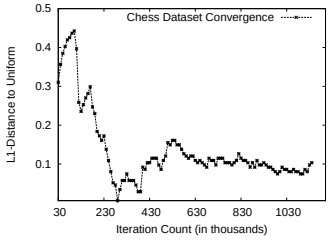
(d) Statistics of uniform sampling on DTP dataset and Comparison with Ideal case



(e) histogram



(f) individual counts



(g) Variation distance

Uniform-Sampling				Ideal	
Maximum	Minimum	Median	Std	Median	Std
156	6	100	13.64	100	10

(h) Statistics of uniform sampling on Chess dataset and Comparison with Ideal case

Figure 4: Experimental Results of Uniform Generation of Frequent Graph

We also contrast the obtained distribution with a perfect uniform sampler. If the dataset has m frequent patterns and we perform the uniform generation for $r \cdot m$ iterations, the number of times (k) an specific pattern will be picked is described by the binomial distribution, $\mathcal{B}(k, n, p)$, where $n = r \cdot m$ and $p = \frac{1}{m}$. The expected number of times a frequent pattern is obtained is $np = r \cdot m \cdot \frac{1}{m} = r$, and the standard deviation is $\sqrt{np(1-p)} = \sqrt{\frac{r(m-1)}{m}}$. If we increase n (by increasing r) sufficiently, the distribution would resemble a normal distribution, for which mean and median values are the same. Thus, in the ideal case, with $r = 200$, and $m = 227$ as in our running example, we expect that the median visitation count is $r = 200$, with a standard deviation of $\sqrt{400(\frac{227-1}{227})} = 14.11$ (as noted in the table in Fig. 4(d)). For our uniform sampler these values are 209 and 59.02 respectively. Note that the results in this table are computed from the averages of 3 different runs to diminish any artifacts of randomness.

One interesting behavior in Figure 4(b) is that the distribution is skewed towards the right with a small peak on the left suggesting a bimodal distribution. We further analyze the result-set to explain the above behavior. Note that the proposal distribution of our uniform sampling algorithm chooses each neighbor of the pattern (a node in the POG) uniformly (line 4 in Figure 3), which favors patterns that have more neighbors in the POG. Although, Metropolis-Hastings algorithm counter-balances this bias by rejecting the proposal distribution at an appropriate rate (line 6-7 in Figure 3), the bias does not eliminate completely because of the poor mixing rate of the POG graph. In our dataset, more than 50% of the patterns are paths. Note that a path graph generally has very few neighbors in the POG. So, they are under-sampled unfairly by the proposal distribution. Our analysis reveals that the visit counts of these patterns are generally lower than the median and they constitute the small peak at the left side of the distribution.

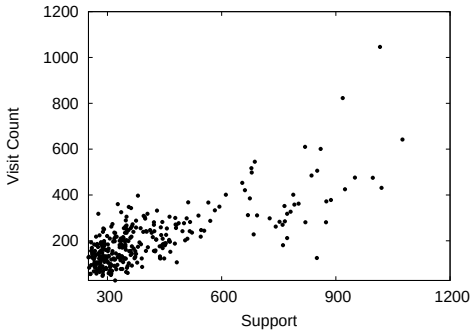
To validate the above explanation, we run our algorithm for itemset patterns. For this case an edge of the itemset POG connects two frequent itemsets of size z and $z-1$, where both have $z-1$ items in common. From a graph point of view, one can consider the itemset dataset as a clique dataset where every item in the itemset is just the vertex label of one of the vertices of the cliques. All the frequent graphs are clique graphs; hence POG is dense and every pattern has a good number of neighbors in POG (in comparison to path graphs in the DTP datasets). We expect to get better uniformity result for this dataset.

For this experiment, we use the Chess dataset from UCI Machine Learning Repository, the dataset has 3196 transactions with an average of 10.25 items in each transaction. With a support value of 2500, it finds 11493 frequent patterns. We run our uniform sampler for 1149300 times. The visit count statistics are shown in Figure 4(h). The corresponding visit count bar chart, visit count distribution and the variation distance curve are also shown in Figure 4(e), (f) and (g), respv. It is very evident that much better uniformity is achieved in this experiment, although we sample

each pattern for an average of 100 times. The distribution curve has perfect normal shape, the median and the mean value of the distribution are identical and the standard deviation of the visit count data is almost equal to the ideal case. However, the spectral gap of this POG is also poor; with a value of 0.05 only it is marginally better than the earlier dataset.

α	Accept-rate	Effec-prob		Correl-ation	variation-Dist
		Up	Down		
.66	0.72	0.33	0.15	0.41	0.194
.60	0.76	0.29	0.18	0.61	0.146
.50	0.78	0.21	0.23	0.76	0.144
.40	0.74	0.15	0.29	0.69	0.237
.34	0.69	0.13	0.33	0.60	0.297

(a) Performance Data for different values of α



(b) Scatter Plot

Figure 5: Experimental Results of Support Biased Generation of Frequent Graphs

7.2 Support-biased Sampling

For this experiment we use the DTP dataset with a support value of 250; the number of frequent subgraphs is 323. Like before, our strategy to verify the sampling distribution quality by comparing the visit count value with the support value of a pattern. We run the sampler for a total of 64,600 iterations. We expect patterns with higher support values to have higher visit count.

The first step of this experiment is to choose a right value of α so that the proposal distribution is close to the target distribution as much as possible. For this we execute five independent trial runs with different values of α and compute the acceptance rates (as in Eq. 5). The result is shown is the table in Fig. 5(a). We also compute the effective up and down probabilities (per pattern) in the proposal distribution by using the Eq. 8 for each iteration. The average of these values over all the iterations is shown in Column 3 and 4 of the same table. Finally, we compute the correlation between visit count and support which is shown in the Column 5. The variation distance between the desired distribution (computed by normalizing the support values) and the achieved distribution (computed by normalizing the visit count) is shown in Column 6.

Interestingly, for $\alpha = 0.5$, the algorithm has the best acceptance rate (0.78), which yields the best sampling performance with the highest correlation and the least variation distance. As we deviate away from 0.5, the acceptance rate drops and that adversely affects the sampling performance. The average effective up and down probability per pattern for the case of $\alpha = 0.5$ is 0.21 and 0.23 respectively, which suggests that for this dataset the required bias to downward move is very small that can be achieved by choosing $\alpha = 0.5$. Note that, α and $1 - \alpha$ is just the aggregated probability for the up and down neighbors; hence, the actual probability to move to an up neighbor or to a down neighbor also depends on the number of up and the number of down neighbors. For $\alpha = 0.4$, the effective up and down neighbor probability is 0.15 and 0.29 respectively, which biases downward walk much more severely than the required causing a decreased acceptance rate. In practice for large dataset, such exhaustive search for the best value of α is impractical, hence it should by updated dynamically during the simulation by monitoring the acceptance rates.

In Figure 5(b), we show the scatter plot of visit counts with respect to the support values. From the plot, a linear relation between the visit count and the support value of a pattern is evident. For example, the third highest ranked pattern in terms of support value (1016) is sampled the most (1046). The correlation value among these two variables is very good, which is 0.76; the corresponding p-value is 0 which means that the alternate hypothesis that there is no correlation between visit count and the support value can be rejected with a significance value of 100%.

7.3 Discriminatory Subgraph Sampling

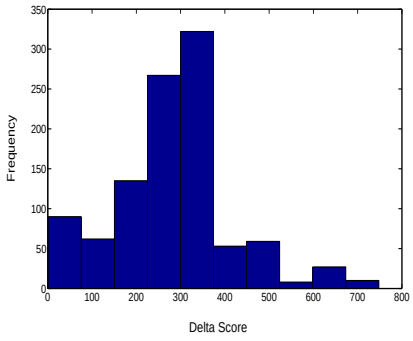
For this experiment, we use the Mutagenicity I dataset that is used in [4]. It has 4337 chemical graphs of two different classes with 2401 and 1936 members in each class. The average vertices and edges in these graphs are 17 and 18 respectively. This dataset is also small and can be mined within a few minutes, but we are using it to show the sampling effectiveness; in particular, we want to see whether our method can obtain high quality discriminatory patterns.

We mined the dataset with a minimum support of 300, for which there are 1034 frequent subgraphs. We also compute the delta-score of these subgraphs. The distribution of the delta-scores for these patterns are shown in Fig. 6(b). The mean and the median delta-score are 284.4 and 295, respectively, with a skewed distribution. The majority of the patterns are not discriminatory or have average discriminatory value. There are only 128 patterns that have delta-score more than 400, which is 13% of the total frequent subgraphs. Since minimum support is 300, there is no infrequent pattern that can have delta score higher than 400. The highest delta-score of a pattern is 748.

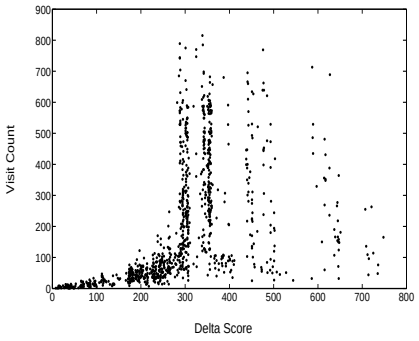
Like before, we run the algorithm for 204800 samples (200 $\times$ 1034) and compute the visit count of each pattern. We show the distribution of the visit count values against the delta-scores of the patterns in Fig. 6(c). The relation is not linear as we wanted it to be, but it is very different from the frequency distribution in Fig. 6(b) as desired. That is, we would like 6(c) to be biased towards patterns with high delta score. Clearly, we sample many patterns that have a delta-score in the range of 450-520 much more than 200 times (the mean value). 43% of the time our sampling algorithm spends on patterns that have delta score more than 350 though they constitute only 29% of the frequent patterns. On the contrary, more than 54% subgraphs have delta-scores less than 300, where our random walk spends less than 30% of the time. Finally, there are some very discriminatory patterns that the algorithm samples less than 200 (the mean value) times. The correlation value is still

Sample No	Delta Score	Rank Rank	% of POG-Explored
1	404	132	5.7
2	644	21	11.0
3	707	10	10.8
4	282	593	2.4
5	646	17	5.5
6	280	595	2.8
7	627	27	3.3
8	709	9	7.7
9	721	5	9.1
10	725	4	8.9
11	280	595	4.1
12	343	320	5.3

(a) Delta Score, Rank and % of POG Explored



(b) Scatter Plot



(c) Delta Score Distribution

Figure 6: Experimental Results of Significance Biased Generation of Frequent Graphs

very high which is 0.48 with a p-value of 0.

The objective of discriminative pattern mining is to use them for classification, so one is more interested to find only the top-k discriminatory patterns. To find the relative ranking of the best discriminatory pattern that the sampling process visits, we modified the sampling algorithm so that it saves the best discriminatory pattern that it has visited up to that point. We started 12 independent random walks, allowing each to take 40 successful steps (if a proposed walk is rejected, we do not count it as a valid step) and report the best discriminatory pattern visited. We also report the percentage of POG it completely enumerated in that process, i.e., these are the patterns for which the *gidset* is computed. The ranking result is shown in Figure 6(a). In top-10, the random walk visited 4 patterns that have delta score rank of 4, 5, 9 and 10, with delta values of 725, 721, 709, and 707 respectively. The best delta value in this dataset is 748. The percentage of lattice visited for these 4 runs is on average less than 10%. The timings for these 40 runs are slightly (5-10%) higher than the proportional (in terms of percentage of POG explored) time for the complete mining algorithm [5]. Note that, we also sample some patterns with poor delta scores. Interestingly, in those cases the process do not explore more than 5% of the POG, i.e. the random walk just got stuck in a local region for these cases.

Dataset	#Graph	#Avg-Vertex	#Avg-Edge
Protein Interaction	3	2154	81607
Cell-Graphs	30	2184	36945

Figure 7: Statistics of Large Graph Datasets

7.4 Sampling Results on Large Graphs

Output space sampling is mostly useful for mining large graphs, for which traditional mining algorithms do not finish in a reasonable amount of time. In the next experiment, we show the effectiveness of the sampling approach over traditional graph mining algorithms. We use two large graph datasets: (1) protein-interaction (PI) graphs taken from [7] and (2) cell-graphs [1]. The cell-graphs have class labels based on whether the corresponding graph belongs to a benign (0) or invasive (1) tissue samples; in this dataset there are an equal number of graphs of either class. The statistics of the datasets are shown in Figure 7. All these experiments were run in a 2.2GHz machine with 2GB RAM running Linux OS.

No complete mining algorithm could mine these graphs for 2 full days of running with 100% support (we tried gSpan [35] and gaston [24]). So, we allow our sampling algorithm and a complete mining algorithm (an adaptation of DMTL [8] that does not save embeddings, since the one that saves embeddings crashes within a few minutes by exhausting the virtual memory) to run for 3 hours. DMTL, which is a depth-first mining algorithm, found 2187 patterns, the largest one with size 34 where the patterns are about 91% similar based on edge-multiset distance [7] (i.e. the proportion of the number of common edges when the graphs are treated as multi-edge sets). Whereas our sampling algorithm traversed 130 patterns (it enumerated 1839 patterns) by uniform sampling, where the maximum sized pattern is 47 and the average similarity between the visited patterns was only 17%.

For the cell-graph dataset, our objective is to find subgraph patterns with high delta score. We run the discriminatory pattern sampling algorithm with a minimum support value of 6 (20%). The leap search [34], and CORK [23] algorithms did not run on this dataset. Both failed with a segmentation fault. But, we could obtain around 26 patterns with delta score more than or equal to 9 in 2 hours of running. Maximum size of the pattern we find is about 21 edges. For the same period of time, the patterns that the DMTL algorithm enumerated had only 3 patterns that had a delta score of 9 or more, though the maximum sized pattern it enumerated had 28 edges.

8. DISCUSSION AND CONCLUSION

Our sampling based approach is good for mining databases of large graphs for which the combinatorial space is prohibitively large. By adopting a random walk approach it samples very dissimilar patterns and reaches different parts of the POG very effectively. The experiments shown above clearly demonstrate this claim.

However our approach also has some limitations. For algorithmic efficiency, the entire graph database should be in memory so that the up and down neighbors of a pattern

[5]For fairness, we compare with the complete mining algorithm that uses similar data structures and algorithms. For instance the complete mining algorithm also uses Ullmann's algorithm for subgraph isomorphism test and it does not save the embeddings in memory, however it uses the rightmost extension with min-dfs-coding [35].

can be obtained quickly. If the database does not fit in the memory, a random walk on the POG becomes highly inefficient. Another important consideration for MH sampling is to choose the proposal distribution appropriately. We have seen in the experiments in Section 7.2 that it has strong consequence on the quality of the sampling. One should try to employ all the prior knowledge in the proposal distribution. The only consideration is that it should be much cheaper to compute with respect to complete enumeration of the neighbors of a pattern.

This work opens up several directions for future research. First, there is scope for improving the mixing rate by adding random edges on the POG after making few simulation walks similar to ORIGAMI [7]. This would embed a random graph on top of POG; since a random graph has very high mixing rate, it would definitely improve the efficiency of the output space sampling. Also, in the discriminatory pattern sampling, it would be good to integrate the sampled pattern within a graph classification framework, and compute the classification accuracy, etc.

9. ACKNOWLEDGEMENTS

We like to thank Bjorn Bringmann for providing us the graph classification datasets.

10. REFERENCES

[1] C. Bilgin, C. Demir, C. Nagi, and B. Yener. Cell-graph mining for breast tissue modeling and analysis. In *IEEE Engineering in Medicine and Biology Society*, 2007.

[2] M. Boley and H. Grosskreutz. A randomized approach for approximating the number of frequent sets. In *IEEE Int'l Conf. on Data Mining*, 2008.

[3] I. Bordino, D. Donato, A. Gionis, and S. Leonardi. Mining Large Networks with Subgraph Counting. In *Proc. of ICDM*, 2008.

[4] B. Bringmann, A. Zimmermann, L. Raedt, and S. Nijssen. Don't be afraid of simpler pattern. In *Proc. of PKDD Conference*, 2006.

[5] C.Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *In Proc. of Neural Information Processing Systems (NIPS)*, 2006.

[6] V. Chakravarthy, V. Pandit, and Y. Sabharwal. Analysis of Sampling Techniques for Association Rule Mining. In *Proc. of 12th International Conf. on Database Theory*, 2009.

[7] V. Chaoji, M. Hasan, S. Salem, J. Besson, and M. Zaki. ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns. *Statistical Analysis and Data Mining*, 1(2):67–84, June 2008.

[8] V. Chaoji, M. Hasan, S. Salem, and M. Zaki. An Integrated, Generic Approach to Pattern Mining: Data Mining Template Library. *Data Mining and Knowledge Discovery Journal*, 17(3):457–495, 2008.

[9] B. Chen, P. Hass, and P. Scheuermann. A new Two-Phase Sampling based Algorithm for discovering Association Rules. In *SIGKDD Proceedings*, pages 462–468, 2002.

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

[11] V. Guruswami. Rapidly mixing markov chains: A comparison of techniques. Technical report, MIT Laboratory of Computer Science, 2000.

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

[13] J. Huan, W. Wang, D. B, J. Snoeyink, J. Prins, and A. Tropsha. Mining Protein Family Specific Residue Packing Patterns from Protein Structure Graphs. In *Proc. of RECOMB*, 2004.

[14] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, 2003.

[15] J. Huan, W. Wang, J. Prins, and J. Yang. SPIN: Mining Maximal Frequent Subgraphs from Graph Databases. In *SIGKDD*, 2004.

[16] C. Hubler, H. Kriegel, K. Borgwardt, and Z. Ghahramani. Metropolis Algorithms for Representative Subgraph Sampling. In *Proc. of ICDM*, 2008.

[17] R. Jin, M. Abu-Ata, Y. Xiang, and N. Ruan. Effective and efficient itemset pattern summarization: regression-based approaches. In *KDD '08: Proc. of SIGKDD*, pages 399–407, 2008.

[18] S. Kramer, L. Raedt, and C. Helma. Molecular feature Mining in HIV data. In *Proc. of SIGKDD*, pages 136–143, 2001.

[19] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *ICDM*, 2001.

[20] L. Li, W. Fu, F. Guo, T. Mowry, and C. Faloutsos. Cut-And-Stitch: Efficient Parallel Learning of Linear Dynamical Systems on SMPs. In *Proc. of SIGKDD*, 2008.

[21] S. Morishita and J. Sese. Traversing Itemset Lattice with Statistical Metric Pruning. *PODS*, pages 226–236, 2000.

[22] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[23] M.Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. Smola, L. Song, P. Yu, X. Yan, and K. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *SIAM Int'l Conf. on Data Mining*, 2009.

[24] S. Nijssen and J. Kok. A quickstart in frequent structure mining can make a difference. In *KDD Proceedings*. ACM, 2004.

[25] R. Y. Rubinstein and D. K. Kroese. *Simulation and the Monte Carlo Method, 2nd Ed.* John Wiley & Sons, 2008.

[26] A. Sinclair. *Algorithms for Random Generation and Counting*. BirkHauser, 1992.

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

[28] H. Toivonen. Sampling Large Databases for Association Rules. In *VLDB Proceedings*, pages 134–145, 1996.

[29] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of ACM*, 23(1):31–42, 1976.

[30] S. Vishwanathan, K. Borgwardt, and N. Schraudolph. Fast computation of graph kernels. In *In Proc. of Neural Information Processing Systems (NIPS)*, 2006.

[31] C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 730–735. ACM, 2006.

[32] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 709–720. VLDB Endowment, 2005.

[33] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing Itemset Patterns: A Profile-Based Approach. In *SIGKDD*, 2005.

[34] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining Significant Graph Patterns by Leap Search. In *SIGMOD Proceedings*. ACM, 2008.

[35] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*, 2002.

[36] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295, New York, NY, USA, 2003. ACM.

[37] S. Zhang, X. Wu, C. Zhang, and J. Lu. Computing the Minimum-Support for Mining Frequent Patterns. *Knowledge and Information Systems*, 15(2):233–257, 2008.