MUSK: Uniform Sampling of k Maximal Patterns.

Conference Paper · April 2009

DOI: 10.1137/1.9781611972795.56 · Source: DBLP

2 authors, including:

Mohammad Hasan
Indiana University-Purdue University Indianapolis
110 PUBLICATIONS **1,925** CITATIONS

SEE PROFILE

MUSK: Uniform Sampling of k Maximal Patterns

Mohammad Al Hasan and Mohammed Zaki
Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY, 12180 *

Abstract

Recent research in frequent pattern mining (FPM) has shifted from obtaining the complete set of frequent patterns to generating only a representative (summary) subset of frequent patterns. Most of the existing approaches to this problem adopt a two-step solution; in the first step, they obtain all the frequent patterns, and in the second step, some form of clustering is used to obtain the summary pattern set. However, the two-step method is inefficient and sometimes infeasible since the first step itself may fail to finish in a reasonable amount of time. In this paper, we propose an alternative approach to mining frequent pattern representatives based on a uniform sampling of the output space. Our new algorithm, MUSK, obtains representative patterns by sampling uniformly from the pool of all frequent maximal patterns; uniformity is achieved by a variant of Markov Chain Monte Carlo (MCMC) algorithm. MUSK simulates a random walk on the frequent pattern partial order graph with a prescribed transition probability matrix, whose values are computed locally during the simulation. In the stationary distribution of the random walk, all maximal frequent pattern nodes in the partial order graph are sampled uniformly. Experiments on various kind of graph and itemset databases validate the effectiveness of our approach.

1 Introduction

Graph data are abundant in numerous scientific and commercial application domains and they continue to grow at a rapid rate. Existing graph mining algorithms are falling far behind in their ability to successfully mine such large and dense graphs. The size of the combinatorial search space of traditional graph mining algorithms is a crucial bottleneck for their applicability in real-life applications. One may opt to abort the mining process once a desired number of frequent patterns are obtained to cope with the scalability issue. However, the partial result obtained may not be good representative set. For instance, if we abort a typical depth first search prematurely, it would have explored only a small fraction of the search space in terms of the labels (or chains). Likewise, if we abort a breadth first search, it would have explored frequent patterns only upto a specific size (or level). Not only is the search space large, complete FPM methods also produce an overwhelming number of patterns. That is, the enormity and redundancy of the output space of patterns is typically beyond the grasp of human analysts.

To cope with the above problems, in recent years researchers have proposed several ideas that compress the frequent pattern set to a smaller summary set containing representative, non-redundant and discriminative patterns. However, most of these use a two-step process; they first extract the entire frequent pattern set and they then summarize the set using some form of clustering [16] or probabilistic techniques [15, 18]. The two-step method is inefficient as it computes the entire frequent pattern set, which can be infeasible in many situations. Furthermore, most of these summarization methods have been applied only in the context of itemset patterns.

In this research, we propose MUSK[1], a novel frequent graph pattern summarization approach based on Markov Chain Monte Carlo simulation. It accepts a parameter k from the user and outputs k frequent maximal patterns sampled uniformly from the set of all frequent maximal patterns. It performs a random walk on the frequent (sub)graph partial order and outputs every distinct maximal pattern it visits. The transition probability of the Markov chain is set so that it is ergodic. Furthermore, the stationary probability distribution, of the nodes (in the partial order graph) corresponding to the maximal frequent patterns, is uniform (i.e., all maximal patterns have roughly the same probability of being generated). The salient features of MUSK are as follows:

1. MUSK avoids enumeration of every frequent (or maximal) pattern, and thus is suitable for obtaining representative frequent patterns where complete

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

[1] MUSK is an anagram of the bold letters in the phrase **U**niform **S**ampling of **k** **M**aximal patterns.)

pattern mining algorithms are infeasible or very expensive.

2. Finding representative patterns by clustering a large frequent pattern set can be very costly for complex patterns like graph [3]. As pointed out earlier, enumerating all frequent (maximal) patterns may be infeasible. Further, clustering them may require computing $O(n^2)$ similarities which can also be practically infeasible. MUSK, on the other hand, obtains the uniform sample of maximal patterns in one step.

3. MUSK adopts a random walk instead of traditional breadth first or depth first exploration of the partial order graph; the output set consists of k maximal patterns that is guaranteed to be a uniform sample of the set of all frequent maximal patterns.

4. MUSK is generic with respect to the type of patterns, since it is equally applicable to the simpler patterns like sets, sequences, and trees, that are subsumed by the more general graph patterns.

2 Related Work

Many recent methods have been proposed for graph mining; these include [8, 9, 11, 20]. The focus of these methods is to mine all frequent subgraph patterns, rather than finding orthogonal or representative patterns. There are several works guided towards finding a subset of frequent patterns that are most informative, compressed, discriminative and non-redundant [1, 2, 16, 17]. Out of those, some [1, 2] tied strongly with set theory, as they were proposed to obtain condensed representation of itemset patterns, and as such are not applicable for mining graph patterns. Many others adopt a two-step approach that first enumerate the entire set of frequent patterns and then apply summarization techniques like clustering [17], profiling [15, 18], and so on. However, the focus of these above methods were also on itemsets. More importantly, the post-processing approach is not applicable for the cases when the mining algorithm itself does not finish in a reasonable time. Furthermore, summarization algorithms that are based on the clustering or profiling can require the computation of $\binom{n}{2}$ similarity scores, which can be very costly when the output consists of millions of frequent patterns. In summary, if the objective is to obtain a set of representative patterns, extracting the entire frequent pattern set is extremely inefficient, if not infeasible.

In the graph domain, methods for mining the closed [21] and maximal [7] frequent graphs have been proposed. Even though these two approaches generate a smaller set of patterns, the number of patterns in both cases can still be very large. Moreover, many patterns in the resulting sets can be very similar, hence, they may not be appropriate as a summary or representative pattern set. Two recent works [3, 19] have specifically focused on summarizing graph patterns. To overcome the problem where the mining process may not finish, these works do not guarantee completeness. Yan et. al. [19] proposed the *leap search* approach to mine interesting graph patterns where the interestingness is generally defined by an objective function. The leap search mechanism uses *structural proximity* to leap over significant portions of the search space (i.e., patterns that are similar to those already found are skipped). It also uses the concept of *frequency descending mining* to obtain a good lower bound on the objective function value, which further helps in pruning the search space drastically. ORIGAMI [3] is a randomized algorithm, that randomly traverses the frequent graph partial order to obtain a set of frequent maximal subgraphs; the output set from this step is further filtered in a second step so that the patterns in the final output set are very dissimilar from each other and also are representative.

Both these approaches avoid complete enumeration, but provide a summary set of patterns. However, they do have limitations. Yan's leap search approach is mainly useful for graphs with class labels because the objective function computation is not very suited without such labels. ORIGAMI, on the other hand depends on randomization to obtain a sample of maximal frequent subgraphs; our study reveals that the sampling of maximal frequent graphs in ORIGAMI is far from uniform, which may result in poor quality representatives. For instance, we ran ORIGAMI on the DTP (CM) dataset (see Sec.6) with 10% support, which yielded 301 maximal frequent subgraphs. To study the sampling uniformity, we ran it for 120,400 iterations, where each iteration randomly samples one maximal pattern. In this experiment, one pattern was selected a maximum of 28,902 times, while there were 3 other maximal patterns that were never generated, i.e., the randomized algorithm that ORIGAMI uses has strongly disproportionate generation probability, which clearly affects the quality of representativeness.

3 Problem Formulation

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

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 an *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 $\nexists H \supseteq G$, such that $H \subseteq \mathcal{D}$.

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 two maximal frequent graphs, as 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.

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 [20]) 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 (shown boxed) correspond to $\mathcal{M}$. The bottom element in the partial order is the empty graph (which is frequent by default). Algorithms for enumerating all

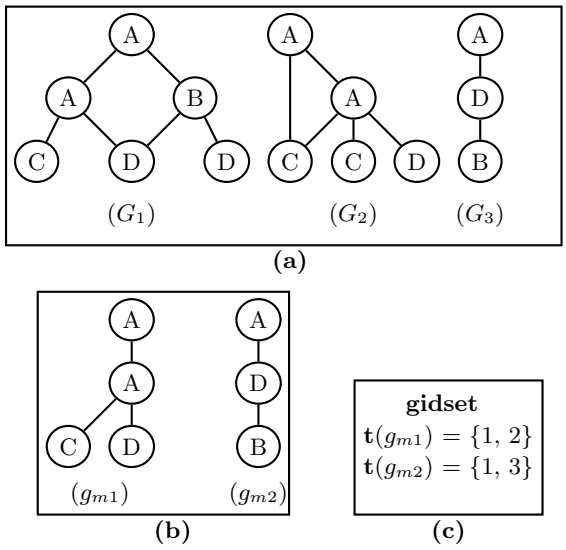


Figure 1: (a) Graph database with 3 graphs (b) Maximal frequent graphs (a) with support 2 (c) **gidset** of the maximal graphs g_{m1} and g_{m2}

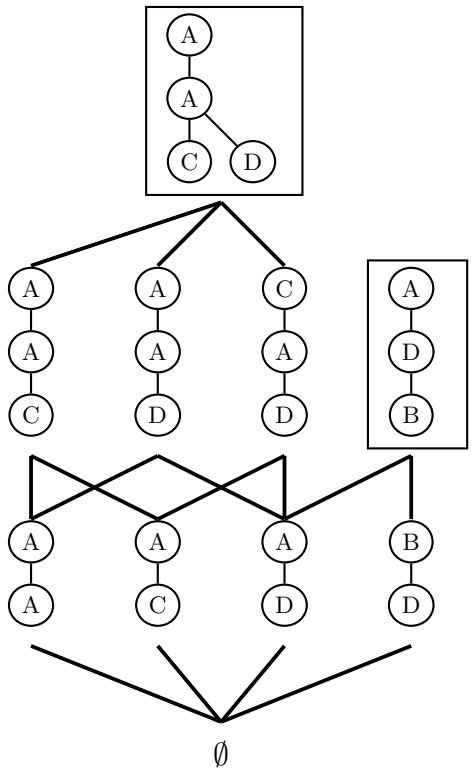


Figure 2: Partial Order Graph of Frequent Subgraphs

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.

Fig. 2(a) show the POG for the example data in Fig. 1.

Uniform Generation: Consider a problem instance that has many feasible solutions. If Σ is a finite alphabet in which we agree to encode both the problem instances and the solutions, a relation $R \in \Sigma^* \times \Sigma^*$ can be interpreted as assigning to each problem instance $x \in \Sigma^*$, a set of solutions $\{y \in \Sigma^* : (x, y) \in R\}$. *Uniform Generation* is the algorithmic process where we generate uniformly at random, a word $y \in \Sigma^*$ satisfying $(x, y) \in R$. If we consider the problem of generating a maximal frequent graph uniformly for a given problem instance which consists of an input graph database and a user defined minimum support, the relation R would be defined as: $R = \{(x, y) : x \in \Sigma^*$ is an encoding of $\mathcal{D}$ and $\pi^{\min}$, and $y \in \Sigma^*$ is an encoding of a maximal frequent graph $(y \in \mathcal{M})\}$.

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:

$$\pi = \pi P \tag{3.1}$$

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:

$$\pi(u)P(u, v) = \pi(v)P(v, u), \forall u, v \in S \tag{3.2}$$

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.

Problem Definition and Discussion: Given a database $\mathcal{D}$ of graphs (or any other kind of patterns), a minimum support threshold $\pi^{\min}$, and the number of desired output patterns k, the objective of Musk is to provide k maximal frequent patterns where each of these k patterns is sampled uniformly from the pool of all maximal frequent patterns. We emphasize the two main objectives:

1. Obtain a set of k maximal patterns without enumerating all frequent (or maximal) patterns.

2. Each maximal pattern should be drawn uniformly from an independent and identical (**iid**) distribution.

Objective 1 demands for only k maximal patterns. One can run any traditional maximal graph mining algorithm, like SPIN [7] and abort the mining process once k maximal patterns are obtained. However, the result-set ranks poorly in terms of representativeness, as the output covers only a localized space of the entire maximal frequent pattern set. Furthermore, the patterns may be very similar to each other. Musk achieves the objective 1 by adopting a randomized algorithm that generates maximal frequent patterns using a random walk on the POG. Objective 2 requires a uniform sample of the maximal patterns, but it is difficult to achieve when combined with objective 1. Two main problems arise: (1) We do not know the size of the sample space as we are not enumerating all the maximal patterns. (2) The sample space of candidate frequent patterns can potentially be very large, so it is statistically difficult to achieve uniformity due to large variance.

The first of the above two problems, in fact, draws the connection between uniform sampling and approximate counting (without explicit enumeration). It is known that if an efficient (approximate) counter is available, an (almost) uniform generation can be achieved efficiently and vice versa [13]. However, the counting of maximal frequent patterns is not easy; in fact, Yang [22] has shown that even for the most primitive pattern class, the itemsets, the problem belongs to the complexity class of #P-Complete, which consists of the hardest counting problems. The second problem is related to the simulation of *rare events*, a well known problem in statistical physics. Since, the number of maximal graphs are much smaller than the number of frequent graphs, and the number of actual frequent graphs can also be much smaller than the candidate subgraphs, sampling a maximal graph is essentially a rare event. An obvious algorithm for uniform generation of a maximal pattern would be to generate a candidate graph pattern uniformly and test whether the graph is maximal. If the test fails, repeat the experiment. This is clearly infeasible. Firstly, it is difficult to obtain a candidate graph uniformly from the pool of all possible candidate graphs without actually enumerating those. Secondly,

the success probability (probability of selecting a maximal pattern) is very small, which may require many trials to obtain one maximal pattern. Problems of above nature have been successfully handled by a family of algorithms, named Monte Carlo Markov Chain (MCMC). MUSK also adopts the same approach to achieve objective 2. The detail theoretical analysis of MUSK is provided in the next section.

4 Uniform Generation of Maximal Patterns

MUSK follows an MCMC approach that simulates a finite Markov chain over the POG (each node is a state), and converges to the desired distribution, i.e., the one that assigns uniform probability to all the maximal frequent patterns. Thus the k maximal patterns returned by MUSK are chosen from an **iid** distribution. Since we have no specific interest regarding the distribution of the remaining frequent (non-maximal) patterns, MUSK is different from the traditional Metropolis-Hastings (MH) [12] algorithm. In MH, the desired stationary distribution vector (π) is known entirely and the transition probability $P(i,j)$ is computed using acceptance-rejection to satisfy Eq. (3.2). On the other hand, MUSK judiciously assigns the value of $P(i,j)$ to satisfy (3.2), and the value of $P(i,j)$ is computed locally while visiting the node i. More importantly, the choice of the value of $P(i,j)$ ensures a uniform stationary distribution with respect to the maximal frequent patterns.

4.1 State Space of the Random Walk

The frequent pattern partial order graph (POG) works as the state space of the Markov Chain on which MUSK runs its simulation. Unlike traditional graph mining algorithms, like gSpan [20], or DMTL [4], it walks on the full edge-set of the POG. Put another way, candidate generation in MUSK allows all possible one-edge extensions without restricting them to be on the right-most path [4, 20]. Note that MUSK constructs 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 (see below for details). If the current node is a new maximal pattern that was not visited before, MUSK returns the corresponding maximal pattern to the user. The local construction of POG is important as it avoids the construction of the entire POG, which would require finding all the frequent patterns.

4.2 Computing Transition Probability Matrix

A straightforward random walk on the POG does not achieve the desired uniform distribution, since different maximal patterns are of different sizes, and consequently the number of neighbors (in the POG) adjacent to a maximal pattern can vary a lot. Thus, a maximal pattern is oversampled if the corresponding node in the POG has a high degree compared to another maximal pattern with lower degree. In fact, the stationary distribution of a node is directly proportional to the degree of that node. Below, we prove a more general statement that uses random walk on weighted graphs.

For a graph $G(V,E)$, with $|V| = n$, let $w(a,b) \in \mathbb{R} \geq 0$, denote the weight on edge $(a,b) \in E$; we assume that $w(a,b) = w(b,a)$. Consider a random walk on G, the transition probability matrix P is given as:

$$P(u,v) = \begin{cases} \frac{w(u,v)}{\sum_{x \in adj(u)} w(u,x)} & \text{if } v \in adj(u) \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

For the case of regular random walk, the weight associated with all the edges is assumed to be $w(a,b) = 1$. The following Lemma holds:

LEMMA 4.1. *In an ergodic random walk with an associated weighted connected (undirected) graph, the stationary distribution of a vertex is directly proportional to the sum of the edge weight incident to that vertex.*

PROOF: Since the random walk is ergodic, it reaches a stationary distribution. Assume that the row-vector π is the stationary distribution vector and P is the transition probability matrix. Now define the following:

$$s(v) = \sum_{u \in adj(v)} w(u,v), \forall v \in [1 \ldots n] \tag{4.4}$$

$$W = \sum_{v \in 1 \ldots n} s(v) \tag{4.5}$$

We can see that $s(v)$ is the sum of the edge weights associated with a node v, and for a given graph W is a constant. We now prove that the stationary distribution given by $\pi(u) = \frac{s(u)}{W}$, for all $u \in [1 \ldots n]$, satisfies Eq. (3.1).

For any $x \in V$, the left hand side (LHS) of Eq. (3.1) is given as $LHS = \pi(x) = \frac{s(x)}{W}$. Now consider the right hand side (RHS) of Eq. (3.1) for x. $RHS = \pi P = \sum_{v \in adj(x)} \frac{s(v)}{W} * \frac{w(v,x)}{\sum_{y \in adj(v)} w(v,y)} = \sum_{v \in adj(x)} \frac{s(v)}{W} * \frac{w(v,x)}{\sum_{y \in adj(v)} w(y,v)} = \sum_{v \in adj(x)} \frac{s(v)}{W} * \frac{w(v,x)}{s(v)} = \sum_{v \in adj(x)} \frac{w(v,x)}{W} = \frac{s(x)}{W}$.

Here we used the fact that $w(v,y) = w(y,v)$. Since an ergodic Markov chain has a unique stationary

distribution, we are done. Thus $\pi(u) = \frac{s(u)}{W}$ gives the stationary distribution for node u, and it is directly proportional to the sum of the edge weights incident to u, i.e., $s(u)$. ■

To achieve uniform distribution for all the maximal frequent patterns, MUSK does not follow the regular random walk that weighs each edge in the POG equally. Instead, MUSK samples each maximal pattern node in the POG with a probability that is inversely proportional to its degree. It also ensures that the random walk is reversible. Assuming c is a constant and d_x is the degree of node $x \in S$, where S is the state space of the random walk, MUSK assigns weight on each node of the POG as follows:

(4.6)
$$w(u,v) = \begin{cases} 1 & \text{if } u \text{ and } v \text{ both are non-maximal} \\ \frac{c}{d_v} & \text{if } u \text{ is non-maximal and } v \text{ is maximal} \\ \frac{c}{d_u} & \text{if } u \text{ is maximal and } v \text{ is non-maximal} \\ 0 & \text{otherwise} \end{cases}$$

The value of the constant c can be chosen as average degree of a node in the POG; c is not critical for the correctness of the algorithm, but it affects the convergence rate to the stationary distribution. With a higher value of c, maximal patterns are visited with a higher probability, so the algorithm becomes efficient, but because of the slow convergence, the sampling uniformity may not be seen until we make many iterations.

Given $w(u,v)$ defined above, the transition probability matrix is obtained via Eq. 4.3; all the row-sums add to 1, as required. Each entry of P is computed locally while visiting a node. For example, if at some time t, the random walk is at node u, MUSK computes the corresponding row of P on the fly by first obtaining the corresponding edge weights. To do so, it needs to test the status of a neighboring node in terms of maximality. If the node u is maximal, then all its neighbors are frequent but non-maximal. On the other hand, if u is non-maximal, its sub-neighbors are non-maximal, but some of its super-neighbors may be maximal; hence, MUSK tests all super-neighbors of u for maximality.

LEMMA 4.2. *An ergodic random walk on the weighted graph where the transition matrix P is defined via the weight function in* (4.6) *is reversible.*

PROOF: Firstly, the edge weights are symmetric; if both u and v are non-maximal, we have $w(u,v) = w(v,u) = 1$; if both are maximal $w(u,v) = w(v,u) = 0$; if u is maximal and v is non-maximal, or vice-versa, $w(u,v) = w(v,u) = \frac{c}{d_m}$, where d_m is the degree of the maximal node ($m \in \{u,v\}$).

From the proof of Lemma 4.1, for any $u, v \in S$, $\pi(u) = \frac{s(u)}{W}$ and $\pi(u)P(u,v) = \frac{s(u)}{W} * \frac{w(u,v)}{\sum_{x \in adj(u)} w(u,x)} = \frac{s(u)}{W} * \frac{w(u,v)}{s(u)} = \frac{w(u,v)}{W}$. Similarly, $\pi(v)P(v,u) = \frac{w(v,u)}{W}$. Since, $w(u,v) = w(v,u)$, we have $\pi(u)P(u,v) = \pi(v)P(v,u)$. So, the random walk is reversible. ■

LEMMA 4.3. *The random walk defined by* MUSK *converges to a stationary distribution.*

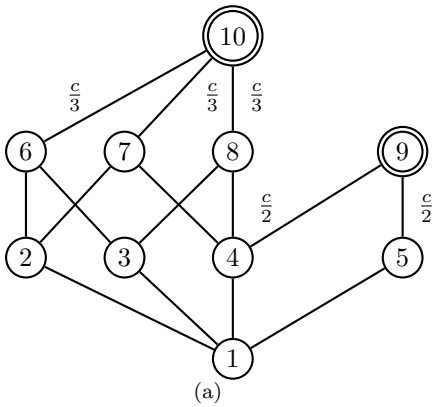
PROOF: To achieve an stationary distribution, a random walk needs to be finite, irreducible, and aperiodic [10]. First, POG is finite since the number of frequent pattern is finite. Second, for any two nodes u and v in POG, there exists a positive probability to reach from one to other, since every pattern 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 multi-stage graph (each stage contains patterns of the same size), so one can make it a bipartite graph by accumulating the vertices from alternate stages 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 [2] [13]. Thus the claim is proved. ■.

THEOREM 4.1. *Maximal pattern output by* MUSK *are generated with the same probability.*

PROOF: By Lemma 4.3 MUSK converges to a stationary distribution. By Lemma 4.2, the associated edge weights are symmetric and the detailed balance equations are also satisfied. The total edge weight incident to a maximal node u in the POG is given as $\sum_{v \in adj(u)} \frac{c}{d_u} = d_u * \frac{c}{d_u} = c$. According to Lemma 4.1, the stationary distribution of the maximal patterns is directly proportional to the above sum of weights, i.e., c. So, MUSK return each maximal pattern with a uniform probability. ■.

4.3 MUSK: A Complete Example Fig. 3(a) shows the POG in Fig. 2 annotated with edge weights, and each distinct pattern has been given a unique id. The maximal pattern nodes are shown doubly circled. The weight on each edge incident on maximal node 10 is $\frac{c}{3}$, since node 10 has degree 3. The edges without any weight are assumed to have weight 1. Fig. 3(b) shows the transition probability matrix P, which is computed by setting the value of constant c to be 3. Fig. 3(c)

[2] In that case, Eq. (4.3) should be changed as below:
$$P(u,v) = \begin{cases} \frac{1}{2} & \text{if } u = v \\ \frac{w(u,v)}{2 * \sum_{x \in adj(u)} w(u,x)} & \text{if } v \in adj(u) \\ 0 & \text{otherwise} \end{cases}$$
However, we skip this addition in the Eq. (4.3) for the benefit of simplicity. Further, the addition of a self-loop does not have any effect on the uniform generation claim.



(a)

Transition Probability Matrix P ($c = 3$)

	1	2	3	4	5	6	7	8	9	10
1	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0	0
2	$\frac{1}{3}$	0	0	0	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0
3	$\frac{1}{3}$	0	0	0	0	$\frac{1}{3}$	0	$\frac{1}{3}$	0	0
4	$\frac{2}{9}$	0	0	0	0	0	$\frac{2}{9}$	$\frac{2}{9}$	$\frac{1}{3}$	0
5	$\frac{2}{5}$	0	0	0	0	0	0	0	$\frac{3}{5}$	0
6	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0	0	0	$\frac{1}{3}$
7	0	$\frac{1}{3}$	0	$\frac{1}{3}$	0	0	0	0	0	$\frac{1}{3}$
8	0	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0	0	$\frac{1}{3}$
9	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	0
10	0	0	0	0	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0	0

(b)

Stationary Distribution π for MUSK:
(.125, .094, .094, .140, .078, .094, .094, .094, **.094**, **.094**)
π for Regular (Unweighted) Random Walk:
(.133, .101, .100, .133, .066, .100, .100, .100, **.066**, **.101**)

(c)

Figure 3: (a) POG of frequent subgraphs shown in Fig. 2. (b) Transition probability matrix P with $c = 3$. (c) Stationary distribution π for MUSK and regular random walk.

contrasts the stationary distribution achieved by MUSK versus that by a regular (unweighted; where all weights are 1) random walk. For MUSK we can see that the stationary probabilities for the maximal nodes 9 and 10 are equal, i.e. $\pi(9) = \pi(10) = 0.094$. For the unweighted random walk the stationary probabilities are: $\pi(9) = 0.066$, and $\pi(10) = 0.101$, which are not uniform; the probability is directly proportional to their degrees (a ratio of 2:3 respectively).

4.4 Convergence rate of random walk

We saw above that MUSK converges to a unique stationary distribution that yields uniformly sampled maximal patterns. We discuss about the rate at which the initial distribution converges. The convergence rate of a random walk has been studied extensively in spectral graph theory [5], 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, where the total number of states can be exponentially large. An algorithm that is *rapidly mixing* is considered efficient.

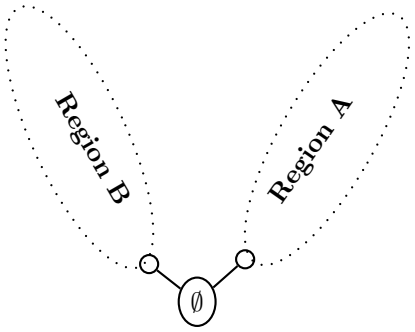


Figure 4: POG with slow mixing of random walk

If π^0 is any arbitrary initial distribution (a row-vector), then the distribution at time t is given as $\pi^t = \pi^0 P^t$; where, P is the transition matrix. The convergence is generally computed by the *relative pointwise distance* of P^t to the stationary distribution π, computed as $\Delta(t) = max_{x,y} \frac{|P^t(y,x) - \pi(x)|}{\pi(x)}$. This term can be bounded by the eigenvalue of the graph Laplacian. If $\lambda_i, i \in [1 \dots n]$ are the eigenvalues, $\lambda = \max_{i \neq 1} |1 - \lambda_i|$, ϵ is a small constant, W is as defined in Eq. (4.5), and $t \geq \frac{1}{1-\lambda} \log \frac{W}{\epsilon \min_x d_x}$, then we have $\Delta(t) \leq \epsilon$ [5].

Unfortunately, the value of λ, W, or $\min_x d_x$ are dependent on the input data and are not available to us before starting the random walk. However, the value of λ can be bounded by the graph diameter [5]. If a rough estimate of the diameter of POG can be obtained via pre-processing, an estimate of the mixing rate can be found. In general, random walk on a graph with smaller diameter has faster mixing rate.

The diameter of the POG can be estimated by finding a pair of maximal graph patterns with minimal overlap. If u and v are two such maximal nodes, and w is the maximum common subgraph between u and v, then the diameter of POG is equal to $size(u) + size(v) - size(w)$. In Fig. 4 we show a hypothetical scenario where the mixing rate of the random walk is not good. Region A and Region B are connected by only the null pattern. So, once the random walk is in region A, it takes a long time for it to go to region B and vice versa. However, for most real life graph datasets, the maximal patterns do have substantial overlap, and thus the convergence is generally very good. In fact, in all our experiments, MUSK outputs a maximal pattern the first time it is visited, since we assume that the random walk has converged by that time. The results show that indeed we do obtain almost uniform generation.

5 MUSK: Implementation Details

The major task in MUSK is to compute the transition probability matrix P locally while performing the ran-

dom walk. To do so, it first obtains all the neighbors of the pattern g. Then it calculates the outgoing probability vector for g by using the Eq. (4.3), which requires it to distinguish the neighbors that are maximal (if any).

5.1 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 precompute 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.

5.2 Support Counting

Only the frequent patterns are part of the POG. So, while Musk 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 Ullmanns's subgraph isomorphism algorithm [14] 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. In the next two paragraphs 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 $\mathbf{t}(q)$.

5.3 Maximality checking

Maximality check for neighbors of the current pattern p is important to fill the transition probability matrix P. The sub-neighbors of p are not maximal, but some of the frequent super-neighbors can potentially be maximal. A pattern is maximal if all its super-neighbors are infrequent. So, Musk attempts to extend each of the super-neighbors (say q) of p. If any of the resulting extensions is frequent, q is not maximal and the process returns immediately. If all the extensions are found to be infrequent, then q is noted as maximal. This information is used in Eq. (4.6).

Musk $(\mathcal{D}, \pi^{\min}, c, \mathcal{M}, k)$:
1. Start with any frequent pattern p as the current node
2. $\mathcal{N}_u$ = all immediate frequent super-patterns of p
3 $\mathcal{N}_d$ = all immediate frequent sub-patterns of p
4. Let $\boldsymbol{w} = (1, 1, \ldots 1)$ be a weight vector of size $|\mathcal{N}|$ where, $\mathcal{N} = \mathcal{N}_u \cup \mathcal{N}_d$
5. **for** each $q \in \mathcal{N}_u$
6. **if is_max**$(q) = true$
7. $d =$ **compute_degree**(q)
8. $\boldsymbol{w}_q = \frac{c}{d}$
9. Normalize w so that $\sum w_i = 1$
10. Save $\mathcal{N}_u, \mathcal{N}_d, w$ in **neighborhood**(p)
11. Choose the next pattern p_{next} with probability proportional to $\boldsymbol{w}$
12. **is_max**(p_{next}) **= true**
13. insert p_{next} in $\mathcal{M}$
14. if $|\mathcal{M}| = k$ **exit**
15. $p = p_{next}$, **goto** line 2

Figure 5: Musk Algorithm

5.4 Musk Pseudo-code

The pseudo-code of Musk is given in Fig. 5. It accepts four input parameters, a graph database $\mathcal{D}$, minimum support threshold $\pi^{\min}$, the random walk constant parameter c (see Eq. (4.6)) and a value k for the desired number of maximal frequent patterns. If no value for c is used, Musk simulates a few iterations to set c as

the average degree of the visited nodes. As we noted earlier, the value of c does not affect the algorithm's correctness, but only the convergence. Generally, k is expected to be much smaller compared to the number of maximal patterns in the dataset.

Line 1 obtains any frequent pattern, p. Line 2 finds all possible super-neighbors of the pattern p. Line 3 finds all possible sub-neighbors of p. For all the neighbors, their support against the database is also obtained. If any candidate neighbor is infrequent, it is removed and the rest of the neighbors are saved in the neighbor list $\mathcal{N}$.

To compute the transition probability, MUSK runs maximality test (line 6) on each pattern in $\mathcal{N}_u$. Once the status (maximal or frequent) of all the neighbors is known, the weight vector is computed (line 8) and normalized (line 9). Using the probability distribution we choose the next step of the random walk. Along the walk, if we visit any maximal node, we save it in the result set, $\mathcal{M}$. MUSK terminates once k distinct maximal patterns are obtained.

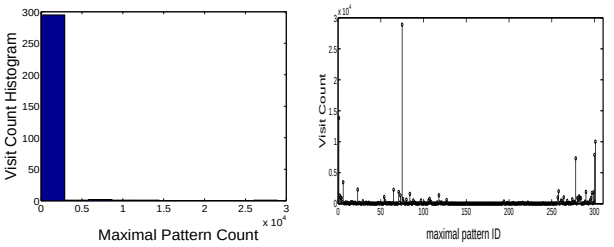
6 Experimental Results

There are three main objectives for the empirical evaluation. Firstly, we want to experimentally validate our claim that MUSK samples each maximal pattern uniformly. Secondly, we want to show that the set of maximal patterns obtained by MUSK is much better in terms of non-redundancy in comparison to the same number of patterns obtained by traditional maximal pattern mining algorithms. Finally, we want to evaluate the performance of MUSK in terms of execution time.

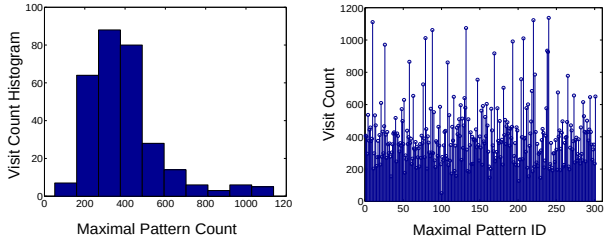
6.1 Uniform Pattern Generation

We used 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.

We first compare the uniform generation capability of MUSK with that of ORIGAMI [4]. ORIGAMI is a two-stage algorithm that obtains a set of maximal frequent graph patterns and summarizes them via a local-optimal algorithm in the post-processing step. In this experiment, our focus is only at the first stage of ORIGAMI, where a randomized algorithm is used to obtain a set of maximal frequent patterns. Both MUSK and ORIGAMI uses random walks to obtain one maximal pattern in each iteration. We allow both the algorithms to run for sufficiently large number of iterations, and analyze the number of times each pattern is visited. The intuition is that in the case of uniform generation



(a) ORIGAMI histogram (b) ORIGAMI individual counts



(c) MUSK histogram (d) MUSK individual counts

MUSK		ORIGAMI		Ideal	
Median	Std	Median	Std	Median	Std
367	185.7	16	2048	400	19.97

(e) Comparison between MUSK, ORIGAMI and Ideal case

Figure 6: Uniform Generation on DTP dataset

each maximal pattern will be visited (generated) nearly equal number of times. We set the support value $\pi^{\min}$ sufficiently high so that the number of maximal pattern is small and a large number of iterations can be performed in a reasonable amount of time.

Fig. 6 shows the results with $\pi^{\min} = 10\%$; the exact number of maximal patterns is 301 and we ran both MUSK and ORIGAMI for 120,400 iterations, so that in the case of ideal uniform generation, each maximal pattern would be generated 400 times. For ORIGAMI, the distribution is very skewed as seen in Fig. 6(a), which shows the frequency histogram of visit counts. That is, the x-axis shows the number of times a pattern is visited, and the y-axis shows how many patterns fall in that bin. We can see clearly that many patterns are visited only a few times, but there are some patterns that are visited many many times. Fig. 6(b) shows the visit counts for each of the 301 maximal patterns. For example, ORIGAMI generates one maximal pattern 28902 (out of 120K iteration) times, whereas 3 (out of 301) patterns were never generated. The median of the visit counts is 16 (see Table in Fig. 6(e)), so half of the patterns have in fact been generated less than 16 times! Keep in mind that the ideal uniform generator visits each pattern 400 times. On the other hand, as seen in the histogram in Fig. 6(c), and the individual per-pattern counts in Fig. 6(d), MUSK yields a much more uniform distribution. In fact, MUSK had visited all the 301 patterns at least once by 5553 iterations. The

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

minimum, median and the maximum visit counts were 51, 367, and 1137 respectively. Note that the median value is sufficiently close to the actual mean (400), in comparison to the case of ORIGAMI. Note also that MUSK has a much smaller standard deviation of 185.7 compared to 2048 for ORIGAMI, further justifying the superiority of MUSK.

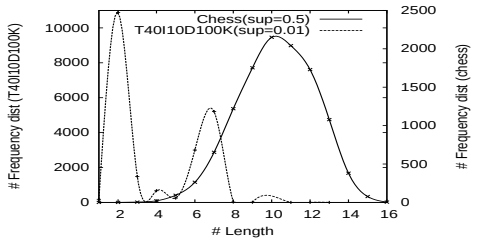
The question remains, how good is MUSK's uniform distribution in practice? Let us contrast MUSK with a perfect uniform sampler. If the dataset has m maximal patterns and we perform the uniform generation for $r \cdot m$ iterations, the number of times (k) an specific pattern will be picked is described 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 maximal 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 $r = 400$, and $m = 301$ as in our running example, we expect that the median visitation count is $r = 400$, with a standard deviation of $\sqrt{400(\frac{301-1}{301})} = 19.97$ (as noted in Table 6(e)). We can see that the distribution curve of MUSK is also approaching a normal curve; but the peak value is slightly less than 400 and it has a longer right tail. If we increase the number of iterations, more uniformity would be seen. On the other hand, for ORIGAMI, the distribution curve is completely distorted. Note here, for MUSK a perfect uniformness with a finite number of iterations also depends on the mixing rate of the random walk and the value of c.

Itemset Patterns: We also analyzed MUSK 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. It is easy to analyze uniformness for itemsets, because the number of neighbors of a node in the POG for itemsets is directly proportional to the size of the pattern, whereas for graph data, the connectedness of a pattern plays an important role in finding its neighbor count.

We took the following two datasets used in [6], the real dataset, chess, and the synthetic dataset, T40I10D100K (we refer to it as T40). It was shown in [6] that the maximal patterns in these two datasets follow very different distributions in terms of their sizes. For instance, chess has a symmetric distribution (close to normal), with an average and maximum pattern length of 10.25 and 16, respectively. T40 has an exponentially decaying distribution, where most of the maximal patterns have size 2 and then the number of patterns drops sharply for longer lengths, though there is a small peak at size 7. The longest maximal has a length of 13. Fig. 7(a) shows the dataset characteristics, and Fig. 7(b) plots the size distribution of the maximal patterns.

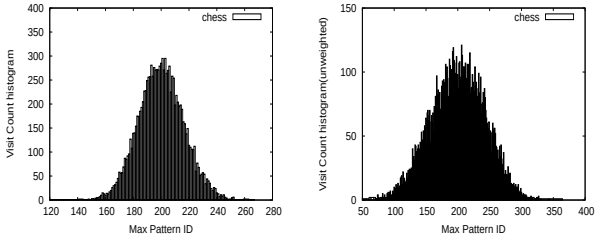
Dataset	Size	# Maximal	Avg Length	Longest Length
Chess	3196	11463	10.25	16
T40	100,000	21692	3.92	13

(a) Itemset Datasets



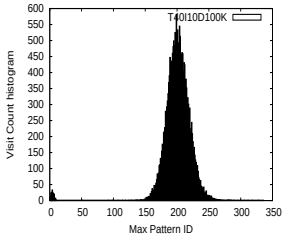
(b) Length Distribution of Maximal patterns

Figure 7: Itemset Datasets

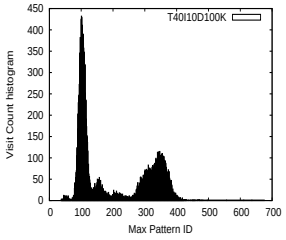


(a) Chess (MUSK)

(b) Chess (URW)



(c) T40 (MUSK)



(d) T40 (URW)

	MUSK		URW		Ideal	
Dataset	Median	Std	Median	Std	median	std
Chess	200	16.27	201	42.58	200	14.14
T40	201	24.74	125	112.06	200	14.14

(e) Comparison between MUSK, URW, and Ideal

Figure 8: Uniform Generation on Itemset dataset

We compare MUSK with the unweighted random walk (URW), where the maximal patterns are sampled with respect to their length (see Lemma 4.1). Fig. 8 shows the results. For chess, $r = 200$, and the number of maximal patterns is $m = 11463$. For chess both MUSK and URW obtain a normal distribution of visitation counts, with medians essentially coinciding with the ideal value of 200. However, MUSK has a much better uniformity with a standard deviation value of 16.27

(close to the ideal of 14.14) compared to 42.58 for URW (see Fig. 8(e)). This difference is reflected in the normal curves; MUSK has a more centralized distribution (thin), whereas URW has more wider distribution.

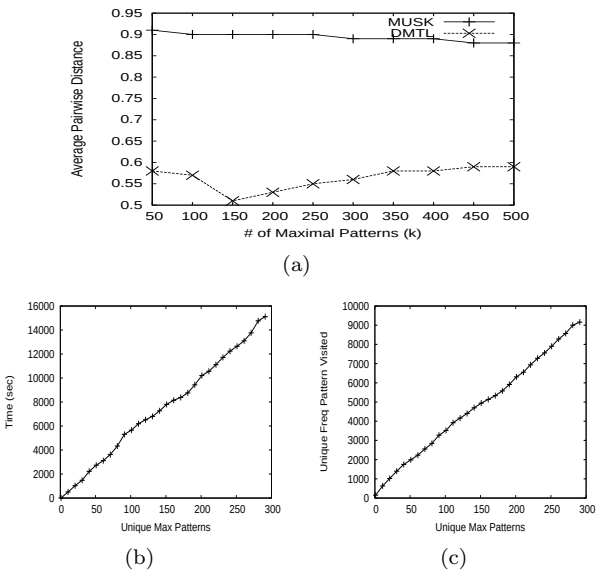
For T40, we have $r = 200$ and $m = 21692$. MUSK still achieves a bell-shape curve (except for 6 patterns that were visited between 3-10 times), whereas URW shows a bimodal distribution similar to the length distribution of the maximal patterns in this dataset. The median and standard deviation for URW are substantially different from the ideal, whereas MUSK's median and std-dev are close to ideal. The above experiments show the superiority of MUSK in terms of uniform pattern generation compared to ORIGAMI, which uses a biased random walk, as well as compared to the regular unweighted random walk.

6.2 Quality and Performance Experiments

Whereas the primary aim of MUSK is to sample uniformly from the set of maximal patterns, i.e., to find representative maximal patterns, a secondary objective is that these patterns should be different from each other. We can therefore evaluate the quality of MUSK's output by computing the average distance between pairs of output patterns. The distance between two graphs is given as follows: $\delta(G_i, G_j) = 1 - \frac{|mcs(G_i,G_j)|}{max(|G_i|,|G_j|)}$, where $mcs()$ returns the maximum common subgraph.

We generated a dense synthetic dataset of 10,000 random labeled graphs, with an average of 44.15 vertices and 105.52 edges, and with labels randomly chosen between 1 and 10. With $\pi^{\min} = 1\%$, the dataset has 32,553,571 frequent patterns. Gaston [11], one the fastest graph mining algorithms, took around 8 hours to completely mine the dataset. SPIN [7], another maximal graph mining algorithm, broke after a few hours with a memory allocation error, so it was not possible to get an exact estimate of the number of maximal frequent graphs.

We compare the quality of the k representative patterns returned by MUSK with the set of k maximal patterns returned by DMTL [4]. The patterns are obtained by modifying DMTL in such a way that a maximality test is added to each frequent pattern it returns. Fig. 9(a) shows the average pairwise distance as we select k maximal patterns from MUSK and DMTL, with k varying from 50 to 500. We can see that MUSK has a much higher quality; the average distance is around 90% compared to under 60% for DMTL. Fig. 9(b)-(c) also show the time and number of distinct frequent nodes visited by MUSK when extracting $k = 300$ maximal patterns. For instance MUSK visits only about 9,000 frequent patterns, compared to the complete 325 million frequent patterns, to enumerate all 300 maximal



(a) (b) (c)

Figure 9: MUSK Quality & Time

patterns, in about 15,000s ($\approx$ 4 hours). In contrast DMTL returns in only 102s, but gives absolutely no guarantee about the uniformity of the sample; in fact, it returns maximal patterns that are much similar (due to the depth first exploration). Hence, with respect to the quality of the patterns, the timing of MUSK is indeed good. If one wants to obtain such highly non-redundant (average difference 90%) patterns by clustering the set $\mathcal{M}$, (s)he requires to obtain all pair-wise distances, which is very costly for graph. For example, just computing all pair-wise distances for the above 300 patterns took approximately 1 hour, and the cost grows quadratically.

Note that, each iteration of MUSK is costlier compared to each iteration of a traditional maximal pattern mining algorithm. Here, by an iteration we mean all the steps required to output a new maximal pattern. However, note that MUSK's objective is to take a uniform sample of the maximal patterns, where the number of maximal patterns is huge. If the number of maximal patterns is not large, it may be better to just use a complete maximal pattern mining algorithm. On the contrary, when a dataset has an enormous number of maximal patterns, MUSK is the best tool, as it outputs a uniform sample of the desired number of maximal patterns.

MUSK is particularly efficient when the ratio $\frac{|\mathcal{M}|}{|\mathcal{F}|}$ is not too small. For instance, in Fig. 3, we have 2 maximal patterns and at stationary distribution the probability of the random walk to be on a maximal node is 0.188(.094 + .094). So, roughly 20% time of the random walk is spent on the maximal nodes. MUSK is very efficient for these cases, since the mixing rate will be high as the maximal patterns would share

many common frequent sub-patterns. On the other hand, when $|\mathcal{M}| \ll |\mathcal{F}|$, then the proportion of time MUSK spends in a maximal pattern node is very small, and there will be small overlap between the maximal nodes, leading to slow sampling times. For these datasets, ORIGAMI is a better alternative than MUSK. Note that ORIGAMI starts with an empty pattern and always walks upward towards the maximal patterns by taking a random edge. So, the number of frequent patterns ORIGAMI visits to reach a maximal node is linear in terms of the size of the maximal pattern. Off course, there is no guaranty of uniform generation. It is also worth pointing that ORIGAMI walks on a directed POG graph, from more general to more specific patterns (bottom to top). On the other hand, MUSK walks on an undirected graph, following both up- and down-neighbors. Theoretical analysis of random walks on directed graph is more complex, so it remains an open question whether it is possible to provide a uniform sampling guaranty by adopting an ORIGAMI style random walk.

7 Conclusions

In this paper we presented MUSK, an MCMC based algorithm to obtain a uniform sample of size k consisting of maximal graph patterns. MUSK is useful for domains where complete pattern search is not feasible, and in cases where the number of possible maximal patterns is relatively large. We showed experimentally that it yields a nearly uniform and good quality sample, close to the ideal case.

8 Acknowledgement

We like to thank Paul Horn of University of California, San Diego for his useful suggestions regarding uniform sampling.

References

[1] F. Afrati, G. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *KDD Proceedings*. ACM, August 2004.

[2] T. Calders, C. Rigotti, and J.-F. Boulicaut. A Survey on Condensed Representation for Frequent Sets. *Lecture Notes in Computer Science*, 3848:64–80, February 2006.

[3] 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.

[4] 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.

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

[6] K. Gouda and M. Zaki. GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets. *Data Mining and Knowledge Discovery*, 11:1–20, 2005.

[7] J. Huan, W. Wang, J. Prins, and J. Yang. SPIN: Mining Maximal Frequent Subgraphs from Graph Databases. In *KDD Proceedings*. ACM, 2004.

[8] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[9] M. Kuramochi and G. Karypis. An Efficient Algorithm for Discovering Frequent Subgraphs. *IEEE Trans. on Knowledge and Data Engineering*, 16(9):1038–1051, 2004.

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

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

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

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

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

[15] C. Wang and S. Parthasarathy. Summarizing Itemset Patterns Using Probabilistic Models. In *KDD Proceedings*. ACM, August 2006.

[16] D. Xin, H. Cheng, X. Yan, and J. Han. Extracting Redundancy-Aware Top-k Patterns. In *KDD Proceedings*. ACM, August 2006.

[17] D. Xin, J. Han, X. Yan, and H. Cheng. Mining Compressed Frequent-Pattern Sets. In *VLDB Proceedings*. ACM, August 2005.

[18] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing Itemset Patterns: A Profile-Based Approach. In *KDD Proceedings*. ACM, August 2005.

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

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

[21] X. Yan and J. Han. CloseGraph: Mining Closed Frequent Graph Patterns. In *KDD Proceedings*. ACM, 2003.

[22] G. Yang. The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns. In *KDD Proceedings*. ACM, 2004.