Randomly Sampling Maximal Itemsets

Sandy Moens
University of Antwerp, Belgium
sandy.moens@ua.ac.be

Bart Goethals
University of Antwerp, Belgium
bart.goethals@ua.ac.be

ABSTRACT

Pattern mining techniques generally enumerate lots of uninteresting and redundant patterns. To obtain less redundant collections, techniques exist that give condensed representations of these collections. However, the proposed techniques often rely on complete enumeration of the pattern space, which can be prohibitive in terms of time and memory. Sampling can be used to filter the output space of patterns *without* explicit enumeration. We propose a framework for random sampling of maximal itemsets from transactional databases. The presented framework can use any monotonically decreasing measure as interestingness criteria for this purpose. Moreover, we use an approximation measure to guide the search for maximal sets to different parts of the output space. We show in our experiments that the method can rapidly generate small collections of patterns with good quality. The sampling framework has been implemented in the interactive visual data mining tool called MIME[1], as such enabling users to quickly sample a collection of patterns and analyze the results.

Keywords

Maximal Itemsets, Random Walks, Output Space Sampling

1. INTRODUCTION

During the last decade an increase in data being generated and stored has taken place. An important reason for this trend is the relatively low cost of storing large amounts of data. The collected data contains a lot of useful information, which is evident because it is drawn from the real world. Therefore, extracting knowledge from the stored data has shown to be very important and interesting. The major problem, however, is that the really interesting and useful information is typically hidden in the vast volume of data.

Pattern mining techniques are able to extract *potentially* interesting knowledge from data by considering frequently occurring events as interesting. Many techniques are invented based on this idea in combination with the principle of monotonicity [2, 21, 14]. The core functionality of these methods is to give a complete set of patterns that fulfill a given quality constraint. However, the problems with these approaches are that a lot of redundancy is found, and, more importantly, many of the mined patterns are actually not interesting. In fact, the enduser inherently owns the subjective criteria to distinguish between truly interesting information, indeed, every user has it's own background knowledge of the data at hand.

Methods to reduce the amount of patterns and redundancy resulted in smaller collections of patterns, e.g. closed sets [19], maximal sets [5], non-derivable itemsets [8], etc. Unfortunately they could only partially alleviate the problem of pattern explosion and redundancy. A second downside of these approaches is that they generally rely on the enumeration of large parts of the pattern space. Only in later stages the less interesting patterns are pruned from the result. One has to realize, however, that enumerating the pattern space can actually in itself already be infeasible. To decrease redundancy in pattern collections even more, pattern set mining algorithms became increasingly important [1, 20, 17]. The goal of **pattern set mining** techniques is to find a small collection of patterns that are interesting *together*, rather than on their own. For instance, together they describe 90% of the original dataset.

On the other hand the iterative and interactive nature of data mining was already known from the start. One reason, thereto, is that typically pattern mining techniques deal with user-defined thresholds that are hard to set, making it into an iterative process to find a suitable parameter setting. Another reason is that interestingness is in essence subjective to a specific user. What is interesting for one person is by definition not always interesting for another person. Therefore, trying to define objective measures that capture the users' interest is perhaps impossible. In fact, in an ideal setting a user is interactively taking part in the **pattern exploration** process, combining her proper knowledge with objective functions that can be computed in an instant.

We differentiate the following building block for exploratory data analysis: interactivity, instant results and adaptability. First of all, exploration can only be done using a good computer-human interaction in a graphical interface. Secondly, instant results are necessary to allow a user to get immediate feedback on each request. This way she can continue the exploration task, without losing her focus. At last, we believe that results should be easily interpretable

[1]MIME is available at http://adrem.ua.ac.be/mime

and adaptable, giving the user control over the final pattern collection. Using good visualisation techniques a user should be able to understand the results better, faster and more easily [15].

The focus of our work is obtaining small collections of **maximal patterns** with low frequency that describe the data without much overlap and without explicit enumeration of the pattern space. Maximal patterns can for example be used in recommendation systems, where long patterns describe trends. For instance, a list of movies that many people like, can be used as recommendations for other users that share a large overlap of movies with the former list. In this paper we develop a sampling method that samples maximal itemsets based on monotone measures and reports near instant results. Moreover, we implemented our methods in MIME, a visual framework for data analysis developed by Goethals et al. [13]. The integration of the sampling method in a graphical framework enables users to easily explore, analyze and adapt the patterns that are reported by our sampling method. Therefore a user can quickly try out different settings and explore the data in search of her pattern collection of interest.

The following section gives an overview of the related work. In section 3 our sampling method is described and section 4 describes the framework in combination with MIME. Section 5 reports the experimental results of our method on various datasets. The last section concludes this work and discusses future work.

2. RELATED WORK

Numerous methods have been proposed to mine interesting patterns from transactional databases [1, 2, 5, 8, 21]. Some methods try to enumerate all itemsets satisfying a frequency threshold, while others try to obtain a more condensed representation, e.g. maximal itemsets. One problem with these methods is generally the overwhelming number of patterns obtained, making analysis in fact harder. Another problem is that often these techniques have to enumerate all subsets of the output patterns, making them prohibitive in practice. Not many techniques exist to sample the output space of frequent patterns *without* the enumeration step.

In work by Zhu et al. [22], the authors introduce the concept of Pattern Fusion to obtain a small collection of maximal frequent itemsets. This new mining strategy finds maximal itemsets by agglomerating small core patterns into large patterns, as such taking larger jumps in the lattice tree compared to typical sequential algorithms, e.g. Apriori [2] and Eclat [21]. The main idea behind this approach is that large maximal itemsets contain a lot of core patterns with similar characteristics. As a consequence randomly sampling from a pool of core patterns then merging them, results into large maximal itemsets.

Also for itemsets, Boley et al. [6] developed a general sampling framework for generating small collections of patterns from a user-defined distribution. In this work a user constructs a distribution over the complete pattern space by defining singleton preference weights and a multiplicative or additive evaluation scheme. Weights are then used in a Metropolis-Hastings sampling framework with linear time and space requirements. Their method does not materialize the complete pattern space.

The rest of this overview on sampling patterns deals with graph databases. Two methods for randomly sampling maximal subgraphs where proposed by Chaoji et al. [9] and Hasan and Zaki [3].

Chaoji et al. [9] use the same technique as our approach to sample maximal subgraphs. They utilize a two-step approach for finding orthogonal maximal frequent subgraphs. In a first step, a random walk is used to randomly sample maximal graphs. The second step then prunes the output to a smaller set of orthogonal patterns. In their work the random walk only traverses the lattice tree downwards. I.e. for a given graph, a random node can only be extended and nodes are never deleted from the current graph. The proposed method does not guarantee uniformity of the output collection of patterns in step 1, rather a biased traversal is used to simulate uniformity. In contrast, our work presents a framework for sampling maximal patterns using an evaluation and an approximation measure. The latter is used to guide the search to different parts of the lattice.

Hasan and Zaki [3] restrict themselves to uniformly sampling k-maximal subgraphs for a given threshold. They use Metropolis-Hastings and construct a probability matrix that enforces uniformity. Therefore, in each step all neighboring nodes (i.e. subgraphs) have to be checked for maximality. If the neighboring node is maximal, the probability of going to that state is proportional to the inverse of its degree.

Hasan and Zaki [4] proposed a method for sampling frequent subgraphs. A random walk over the partially ordered graph of frequent patterns is used to sample according to a prior interestingness distribution. Their work is similar to ours, in the sense that they also use a branch-and-bound technique to walk over the patterns. The differences are twofold: the output space is different and the underlying random walk is also different. Hasan and Zaki use Metropolis-Hastings as chain simulation method to sample according to the desired distribution. In our method a simple random walk over the remaining subsets is used.

The graphical pattern mining framework MIME [13] has been used as a base for implementing and testing our sampling technique. MIME is equipped with an intuitive user-interface and many interestingness measures to analyze the mined patterns. Moreover, it lets users quickly run various data mining algorithms using different parameter settings.

3. RANDOM MAXIMAL ITEMSETS

3.1 Preliminaries

We denote by $\mathcal{D}$ a dataset containing transactions t_j. Each transaction t_j is a subset over a finite set of items $\mathcal{I} = \{i_1, \dots, i_n\}$. Individual items are also called singletons and will be used interchangeably in the rest of the paper. An itemset X is also a subset over the set of items $\mathcal{I}$. The cover of an itemset is the set of transactions that contain the itemset, i.e. $cov(X) = \{t_j | X \subseteq t_j\}$. The support of an itemset is defined as the size of its cover, i.e. the number of transactions that contain the itemset. A quality function q is an monotonically decreasing function that gives the interestingness of an itemset in the corresponding dataset $q : X \to \mathbb{R}$ (e.g. support). p is a monotonic function that is used as approximation for the quality function.

σ is a minimum quality threshold such that iff $q(X) >= \sigma$ then X is interesting. The negative border are the itemsets that are not interesting wrt σ and q, but whose subsets are all interesting. The positive border, is the set of itemsets

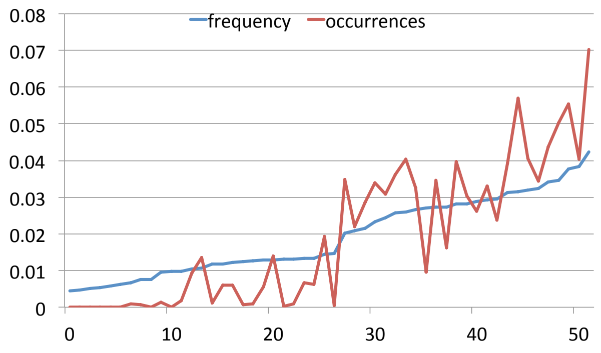


Figure 1: Singleton frequency vs singleton occurrences in maximal sets for mammals datasets with minimum support of 10%

that are interesting, but whose supersets are not. The positive border is also called the set of maximal itemsets.

3.2 Uniform Sampling

In this section we give an overview of two algorithms for uniform sampling of maximal sets.

A first, and naïve, way is to first generate the complete set of maximal sets, then performing sampling on the result. This way each maximal set has an equal, uniform probability of being drawn. The downside of this method, however, is that generating all maximal itemsets is both time and memory consuming. I.e. we have to enumerate almost the complete set of frequent patterns to obtain the collection of maximal frequent patterns. Furthermore, we have to rely on subset checking or other techniques [7] to quickly check if a potential maximal set has a superset in the result already.

A second approach is proposed by Al Hasan and Zaki on graphs [3] and can be adopted to itemsets. For this method we use Metropolis-Hastings to simulate a random walk over the complete pattern space. In each step, the current node represents the current itemset. The connected nodes (i.e. possible transition states) are the subsets with length that is one smaller and the supersets with length that is one larger than the current set. To obtain uniform samples we have to create a doubly stochastic probability matrix P which is used to propose new transitions. An easy way to do so is using the following probabilities [3]:

$$P(u,v) = \begin{cases} \frac{1}{Z} & \text{if } u,v \text{ non max} \\ \frac{1/d_v}{Z} & \text{if } u \text{ non-max, } v \text{ max} \\ \frac{1/d_u}{Z} & \text{if } u \text{ max } v \text{ non-max} \\ 0 & \textit{otherwise}, \end{cases} \quad (1)$$

with d_x the degree of x and, Z a normalization factor over all neighboring nodes. Note that using these weights, we always have to check each neighboring superset for maximality.

Sampling according to the described MH-chain results in a uniform sample of maximal patterns. The chain, however, has to be restarted from a random state for each sample and has to be run for a large number of iterations until a sample is returned. Generally, MCMC methods take a long number of iterations before converging to the stationary distribution, this is called the mixing time. When a chain converges fast to the desired distribution, the chain is called rapidly mixing, otherwise it is slowly mixing.

Algorithm 1 Random Sampling

```
Require: set of singletons $\mathcal{I}$, approximation function $p$
         quality function $q$, quality threshold $\sigma$
Returns: random maximal itemset $R$

1: $R \leftarrow \{\}$
2: $C \leftarrow \{c_j \in I : q(c_j) \geq \sigma\}$
3: while $|C| > 0$ do
4:     $M \leftarrow (c_i, p(R \cup c_i))$ for all $c_i \in C$
5:     $x \leftarrow$ sample from $M$
6:     $R \leftarrow \{R \cup x\}$
7:     $C \leftarrow \{c_j \in C : q(R \cup c_j) \geq \sigma\}$
8: return $R$
```

3.3 Random Sampling

Randomly generating maximal sets can be done in a faster and easier way [9]. The only downside is that uniformity of the method is not guaranteed.

We randomly generate maximal itemsets by performing a simple random walk over all singleton items $\{i_1, \ldots, i_n\}$. We walk over the space of frequent patterns by starting at the empty set and gruadually appending new singletons. Initially each item is given a probability proportional to it's approximation score $p(i_j)$, favoring highly ranked items in the result. Then, for each iteration of the walk we prune the items that result in non-interesting itemsets according to q. The remaining items (also called extensions) are evaluated with respect to the current set and the approximation measure p, i.e. we evaluate $p(X \cup i_k)$ for all remaining i_k. This process continues until no more items can be added such that the set remains interesting. If no new items can be added a maximal itemset has been found.

The basic intuition behind this sampling approach is that higher valued items occur more frequently in the set of interesting maximal sets. This is under the assumption that the approximation function simulates the original quality function. Figure 1 shows an example for the mammals dataset when using frequency as quality and approximation measure. The minimum support threshold used is 10%. The blue line shows the relative support of singletons wrt the cumulative support of all singletons. The red line gives the number of occurrences of an item in the set of maximal frequent itemsets. The relation between both curves is not exact, but we clearly see that high support items occur more often in the set of maximal patterns than low support items.

The sampling procedure is outlined in Algorithm 1. Lines 1 and 2 are the initialization part where singletons that do not meet the quality threshold are discarded. In line 4 and 5 we perform biased sampling using p as bias for all remaining extensions. Lines 6 and 7 make sure that all non interesting supersets are discarded from the set of new samples. This simple random walk over the pattern space can be repeated any number of times.

3.4 Downgrading

Our method tries to sample small collections of maximal patterns that describe different parts of the data. As such, we want to find patterns that have a small overlap in the data described by them. An easy way to do so is using a weighting (or discounting) scheme in the sampling procedure: by giving less weight to patterns that are already covered by previously found patterns, we force the method to come up with different results [16]. Essentially, we construct a distribution over a set of events that dynamically changes over time.

In our method we build a probability map based on possible extensions for the current set. To guide the pattern search process, we can downgrade the probabilities of the extensions (i.e. singletons) that have already been reported in previous sets, using weighting schemes. One downgrading scheme is proposed together with two others that have been adopted from Lavrač et al. [16]:

- **multiplicative weights** [16] decrease exponentially by the number of times a singleton has already been covered. Given a parameter $0 \leq \gamma \leq 1$, we bias singleton weights using the formula $w(i_j, k) = \gamma^k$, where k is the number of times a singleton i_j occurs already in the result.

- **additive weights** [16] can be implemented using the weight assignment $w(i_j, k) = \frac{1}{k+1}$. Using the additive weighting scheme the decrease is less drastical, but also less effective in settings where we want to sample only a small collection of patterns.

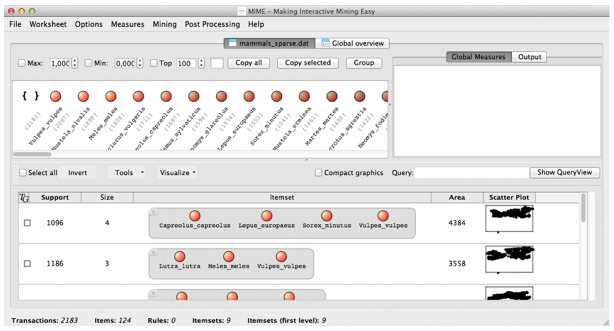
- **adaptive weights** is an adaptive scheme favoring singletons that have not been sampled for a long time. The formula is as follows: $w(i_j, k) = \left(1 - \frac{k}{tot}\right)$, where *tot* is the total number of patterns discounted so far.

Downgrading should be implemented with a precaution however. We still have to guarantee maximality of the patterns being found. As an example suppose multiplicative weights are used with $\gamma = 0$. In this case any item that has been found in just one sample will be given a weight of 0. Using the constructed probability map the 0-weighted items would never be sampled. After a while, only 0-weight elements will be left in the set of possible extensions. An effective way to deal with this problem is to start sampling uniformly in cases where only 0-weighted extensions are left. Indeed, now there is no real favoring of one the items that is left. One could come up with several other methods to deal with this problem aswell.

3.5 Completeness

Our sampling method can use any function as quality and approximation function, with the only restriction that the function has to be monotonic. This monotonic property allows to prune non interesting itemsets during each iteration [2], without losing guarantee that the random walk generates a maximal pattern. In fact, our method can mine *any* of the existing maximal itemset for the given quality threshold. This is easy to see, in the end a maximal itemset is build up by a number of subsets. During the random walk a chain of subsets is sampled, each with probability $P(X'_i)$, where X'_i represents a subset of size i. Now if a maximal set can be generated by a set of distinct random walks R_{max}, then the probability of sampling a maximal itemset of length n is given by:

$$\begin{aligned} P(X_{max}) &= \frac{\sum_{r \in R_{max}} P(r)}{Z} \\ &= \frac{\sum_{r \in R_{max}} P(X'_1)P(X'_2)...P(X'_n)}{Z} \\ &= \frac{\sum_{r \in R_{max}} P(e_1)P(e_2|e_1)...P(e_n|e_1...e_{n-1})}{Z}, \end{aligned}$$



Support	Size	Itemset	Area	Scatter Plot
1096	4	Capreolus_capreolus Lepus_europaeus Sorex_minutus Vulpes_vulpes	4384	
1186	3	Lutra_lutra Meles_meles Vulpes_vulpes	3558	

Transactions: *2183* Items: *124* Rules: *0* Itemsets: *9* Itemsets (first level): *9*

Figure 2: Interface of MIME

with Z a normalization factor over all maximal sets S_{max}. The first equality gives the probability of sampling a maximal set as the sum of probabilities of random walks that lead to this set. The second equality elaborates on individual chains and gives it's probability as the product of the intermediate steps. The last equation links probabilities of intermediate sets to the previously sampled subsets.

Since we know the probability of sampling a maximal itemset, we can give an estimate for the number of samples necessary to sample each distinct maximal itemsets using the generalized version of the Coupon Collector's problem [10]:

$$E(C_m) = \sum_{q=0}^{m-1} (-1)^{(m-1-q)} \sum_{|J|=q} \frac{1}{1 - P_J} \quad \text{with } m = |S_{max}|$$

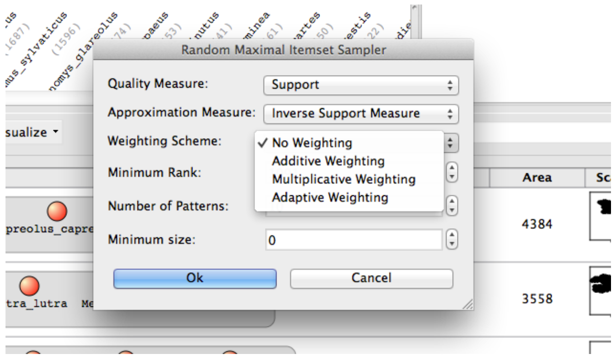
$$P_J = \sum_{X \in S_{max}} P(X).$$

We would like to conclude this section with a small remark. To be completely correct, the approximation function does not have to be monotone. Rather it is possible to use non-monotonic approximation measures, however it is then impossible to see the influence in the final sample.

4. PATTERN EXPLORATION

The goal of our sampling method is to quickly generate small collections of maximal patterns that can easily be explored. Exploration is typically done in a visual framework with easy user interaction. Therefore, we implemented our framework into MIME, a user friendly pattern mining and visualization tool developed by Goethals et al. [13]. The main philosophy of MIME is to give users pattern exploration capabilities by providing a snappy interface with quick response times. Thereto, different caching and threading techniques are used.

In MIME a user is presented with the database in vertical/item format as shown in Figure 2. This is indeed the most intuitive way of looking at the database, since generally the user knows what an item represent. Furthermore, a user is able create patterns by clicking and using drag-and-drop operations. Alternatively she can use a lot of internal pattern mining algorithms to create an initial collection of patterns. Then she can evaluate patterns and/or pattern collections using a variety of interestingness measures [11]. Plugging our sampling method into MIME results in synergic behavior on both sides: (1) since the tool demands quick response times, the addition of this sampling framework provides good means to quickly obtain a small collec-


Random Maximal Itemset Sampler

Quality Measure: Support

Approximation Measure: Inverse Support Measure

Weighting Scheme: ✓ No Weighting

Minimum Rank:

Number of Patterns:

Minimum size: 0

Figure 3: Sampling interface in MIME

tions of maximal patterns. (2) using the interactive interface of MIME we can easily try out different settings for the random maximal sampler and, more importantly, evaluate them immediately.

Figure 3 shows the graphical interface for sampling random maximal itemsets in MIME. Here a user is able to set all necessary information for sampling and also a minimum support size of the patterns. This is usefull when searching for long maximal itemsets with a size at least k. During the sampling process itemsets that are found pop-up one by one and the process can be stopped at any given time.

5. EXPERIMENTAL EVALUATION

5.1 Datasets

For our experiments, we used three datasets of different sizes. The main characteristics of the datasets are shown in Table 1.

The first dataset is *mammals* and is provided by T. Mitchell Jones[2] [18]. This dataset contains information on the habitat of animals in Europe. Each transaction corresponds to a 50 by 50 kilometers region and each item represents a mammal that lives somewhere in Europe. An item that is contained in a transaction corresponds to a mammal living in the specified region.

The other two datasets are publicly available from the FIMI repository[3]. The *pumsb* dataset provides census information about population and housing. The *accidents* dataset [12] contains anonymized information of traffic accidents on the Belgian roads.

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	σ (%)	Max Sets
mammals	2.183	121	10	198.231
pumsb	49.046	2.113	55	92.221
accidents	340.183	468	5	551.073

Table 1: Characteristics of datasets

5.2 Evaluation Metrics

We evaluate our sampled pattern collections using the following metrics:

- **Size**: the size is the number of singletons contained by a pattern. It is important for our sampling method, because we mainly want to obtain long patterns. Shorter patterns are more general, but also provide less interesting information when viewed from a correlation perspective.

- **Jaccard**: the Jaccard index is a similarity measure defined as the size of the intersection of two sets relative to the size of the union: i.e. $Jaccard(X,Y) = \frac{X \cap Y}{X \cup Y}$. For our evaluation, we use the average pairwise Jaccard dissimilarity $(1 - Jaccard)$, between all pairs of patterns in the sampled collection. This gives an intuition on the overlap between two patterns in the collection of samples.

- **Duplicates**: the number of duplicates is interesting in a randomized sampling method, since obviously this is something we want to minimize.

- **Approximation Error**: this set evaluation measure was proposed by Zhu et al. [22] and evaluates a pattern collection as a whole. It measures the average edit distance of a set of patterns to a set of cluster centers. In our setting, the set of clusters is the sampled collection of patterns and the set of patterns is the complete set of maximal itemsets for the same support threshold. Then, for each maximal set we assign it to the closest center (i.e. an itemset from the sample set) using the edit distance $Edit(X,Y) = |X \cup Y| - |X \cap Y|$. This results in a clustering of the maximal sets with respect to our sampled collection. Given a sample collection P of size m, the approximation error is now defined as $\triangle(C_P^{S_{max}}) = \Sigma_{i=1}^{m} r_i/m$, with r_i the maximal edit distance between a member of the cluster and it's center. Lower approximation errors result in descriptive capabilities of the sampled collection wrt the total set of maximal itemsets.

- **Total Area**: another set evaluation measure is the total area covered by the collection. This entails the real coverage by the patterns in the data and not the cumulated sum of the areas of individual areas. Higher total area results in higher descriptive capabilities of the pattern collection wrt the original data.

5.3 Experimental Results

We test our approach to a regular top-k mining algorithm for finding maximal itemsets. We know that sequential algorithms walk over the pattern space such that similar itemsets are found close to each other. Using our sampling method, we find maximal itemsets that are spread over the complete set of maximal sets. All our methods have been implemented in Java and are available as standalone or in MIME [4].

5.3.1 *Pattern Set Quality*

We empirically tested sampling by instantiating our framework with different approximation measures and weighting schemes. As quality measure we used the support measure. For approximation measures we used a mixture of the frequency measure, the inverse frequency measure and an equality measure (i.e. all items are given equal weights).

[2]The mammals dataset can be obtained at http://www.european-mammals.org/

[3]The FIMI repository can be found at http://fimi.ua.ac.be/data/

[4]Implementations are available at: http://adrem.ua.ac.be/implementations

The weighting schemes used throughout the experiments are described in Section 3.4. Uniform sampling is simulated using the naïve approach as described in Section 3.2. At last, we also used consequtive chunks of maximal sets found by the Eclat algorithm with maximal itemsets pruning. As such, we try to simulate running Eclat with very short time budgets for finding maximal patterns.

For each of the methods we did 10-fold cross-validation, where collections of 10 and 20 itemsets are sampled from the pumsb dataset with a support threshold of 50%. To objectively measure the quality of the collections found, we compute the metrics defined in 5.2. Results of the experiments are shown in Table 2.

From Table 2 we can immediately see that chunking is not suitable for obtaining small sample collections to describe the data, i.e. the total area is very low compared to all other methods. Also, the Jaccard distance is very high, meaning that a lot patterns share a large overlap in the number of items. Uniform sampling is already better in terms of Jaccard distance and the approximation error, but lacks the descriptiveness of the original data. For all of the randomized methods the average Jaccard distance between any pair of samples is much lower than the chunking method and the total areas are generally much better than chunking and uniform sampling.

Interestingly, sampling with inverse frequency results in high approximation scores and, therefore, bad cluster centers. However, if we also look at the average size of the patterns found, we see they are the lowest for inverse frequency. In other words, the reason why the samples are bad cluster centers, is because they are too small compared to the average size of all maximal itemset. As such, the edit distance from any maximal set to its' cluster center will always be larger. The difference in size between frequency and inverse frequency can actually be explained intuitively. A set with high support has more extensions with high support. In turn these high support extensions have a higher probability of being sampled, resulting again in high support intermediate sets. This is in contrast to when the set already has a low support, then the extensions are also low on support, and, moreover, the lowest support extensions have highest probability of being sampled.

We analyze the weighting schemes again using Table 2, and we compare to the none weighted case for the same approximation measure. Generally speaking, the weighting schemes mildly influence the quality of the patterns for such small collections. Although no clear winner can be appointed, the additive scheme performs slightly better than the rest: the total area is always the highest and Jaccard the lowest for all measures.

In another experiment concerning the length of maximal itemsets, we sampled 100 times the number of maximal itemsets found in the mammals dataset when setting support to 437 (20%). The distribution of the maximal sets found wrt their length is given in Figure 4. The purple line gives the length distribution of the actual set of maximal patterns. This plot shows similar results for the sizes compared to Table 2. Inverse frequency favors sampling of short patterns, while frequency favors sampling of long patterns. The equal measure is somewhere in the middle, which makes sense because in each iteration high and low support items share the same sampling probability.

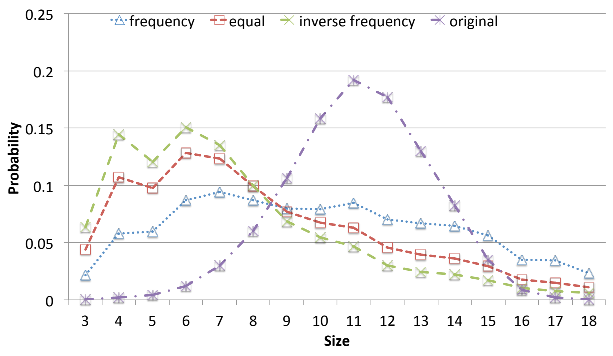


Figure 4: Distribution of length of maximal patterns

5.3.2 *Timing Results*

In the first timing experiment we sample small collections of patterns with low thresholds for the different datasets. We use frequency as quality/approximation measure during this experiment and set no weighting scheme, because weighting schemes do almost not influence the computation times. For the different tests we report the time needed to sample the collection of desired size as well as the number of non duplicate sets. The results are shown in Table 3.

Obviously, the time needed to sample X sets increases linearly by the number of sets to sample. Furthermore, we see that most of the experiments run in less than a few seconds, unless the threshold is set rather low (e.g. 1% for pumsb and accidents). On the other hand, from the experiments we see that sampling just 10 sets, takes less than a second. In an interactive setting this is very reasonable, since in fact we do not want to mine hundreds or thousands of patterns, rather we like to obtain small collections that we can analyze easily and fast. E.g. in MIME, if a user is not satisfied with the collection she can gradually ask for more patterns.

$\mathcal{D}$	σ(%)	Samples	Non-Dup	Time (s)
mammals	10 (0.5%)	10	10,0	0,007
		100	93,6	0,073
		1000	776,1	0,721
	21 (1%)	10	9,9	0,006
		100	96,6	0,055
		1000	864,5	0,551
pumsb	490 (1%)	10	10,0	0,46
		100	99,1	3,933
		1000	946,8	40,207
	26.975 (55%)	10	10,0	0,067
		100	95,6	0,265
		1000	816,1	2,545
accidents	3.401 (1%)	10	10,0	0,624
		100	100,0	4,905
		1000	999,5	48,133
	68.036 (20%)	10	10,0	0,226
		100	96,1	1,187
		1000	818,3	12,235

Table 3: Time results for sampling method using frequency as quality and approximation measure, and without weighting

In a second experiment we measure the time needed for our sampling method to obtain the complete set of maximal patterns. We would like to remind the reader that this is in fact not the main use of our sampling approach.

$\mathcal{D}$	σ (%)	Samples	Method	Weighting Scheme	Size	Jaccard	Ap. Err.	Duplicates	Total Area
pumsb	50	10	chunking	-	14,67	0,73	1,54	0,0	494.323
			uniform	-	12,83	0,28	1,38	0,0	1.164.491
			sampling$_{freq}$	-	13,77	0,26	1,39	0,0	1.424.332
				additive	13,00	0,20	1,50	0,0	1.507.440
				multiplicative	14,77	0,29	1,27	0,0	1.386.859
				adaptive	13,54	0,22	1,47	0,1	1.470.019
			sampling$_{invfreq}$	-	10,55	0,19	1,86	0,2	1.278.312
				additive	10,40	0,14	2,06	0,1	1.375.827
				multiplicative	11,40	0,21	1,86	0,0	1.269.745
				adaptive	10,92	0,18	1,87	0,2	1.314.444
			sampling$_{equal}$	-	12,91	0,22	1,53	0,0	1.410.602
				additive	11,02	0,16	1,87	0,1	1.469.940
				multiplicative	13,28	0,24	1,55	0,0	1.426.970
				adaptive	11,66	0,18	1,70	00	1.405.476
		20	chunking	-	12,45	0,65	1,63	0,0	495.525
			uniform	-	12,94	0,28	1,21	0,0	1.335.700
			sampling$_{freq}$	-	13,41	0,25	1,32	0,1	1.712.057
				additive	11,68	0,18	1,52	0,0	1.776.483
				multiplicative	13,22	0,25	1,30	0,1	1.683.907
				adaptive	12,85	0,21	1,30	0,1	1.728.926
			sampling$_{invfreq}$	-	11,11	0,20	1,65	0,2	1.595.981
				additive	10,02	0,15	1,79	0,1	1.766.286
				multiplicative	10,47	0,19	1,81	0,3	1.597.760
				adaptive	10,68	0,18	1,63	0,4	1.654.833
			sampling$_{equal}$	-	13,22	0,25	1,34	0,2	1.670.994
				additive	11,22	0,17	1,55	0,1	1.790.034
				multiplicative	13,33	0,24	1,31	0,2	1.703.623
				adaptive	12,09	0,19	1,42	0,1	1.727.109

Table 2: Overview of metrics computed on sampled collections from the pumsb dataset

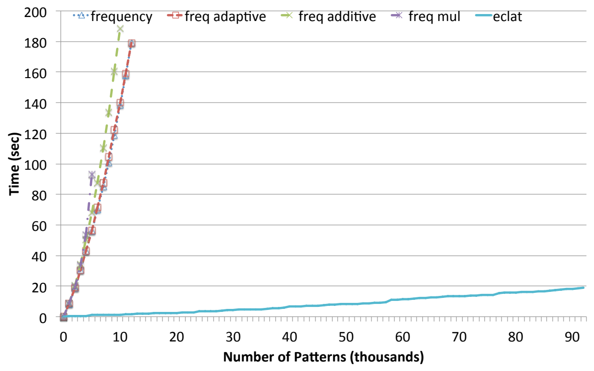


Figure 5: Detailed view of time results for finding distinct maximal sets using sampling

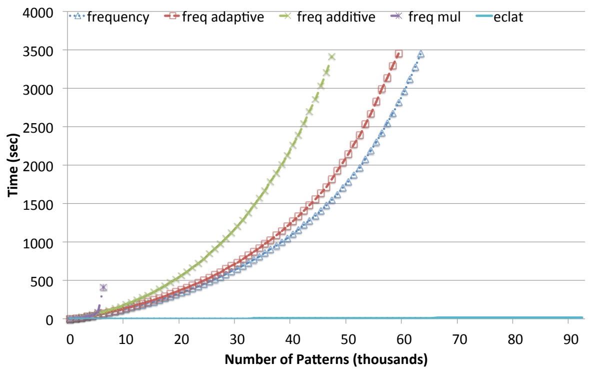


Figure 6: Time results for finding distinct maximal sets using sampling with a time budget of 1 hour

Figure 5 and 6 present time results for finding distinct patterns using different approximation measures. The dataset used for this experiment is the pumsb dataset with a support threshold of 55%. For Figure 5, we first mined the dataset for maximal itemsets using our proper Eclat implementation using bitsets. We used the optimizations described by Borgelt [7] to prune the output set for maximal itemsets. Then we sampled itemsets for 10 times the amount of time Eclat needs to mine all maximal sets. The results show that Eclat is much faster in this experiment, which is not suprising because the sampling methods have to create multiple distribution maps for each sample. In contrast, Eclat only has to make intersections of lists of transaction identifiers. Moreover, we only report the non-duplicate patterns in this timing experiment. We can also see that for the first 5K samples the sampling methods take approximately the same time. Later in the sampling process the discounting measures actually experience more difficulty in finding non-duplicate sets. The bigger picture is shown in Figure 6. For this chart we let the different methods run for 1 hour before stopping execution. Going to higher numbers of distinct patterns we observe an exponential increase in the running time. Indeed, more duplicates are found and our sampling technique has difficulties walking in non-traversed regions of the pattern space.

6. CONCLUSION

We studied the problem of randomly sampling maximal itemsets without explicit enumeration of the complete pattern space. For this purpose we employed a simple random walk that only allows additions of singletons to the current set untill a maximal set is found. The proposed framework uses two types of monotone interestingness functions: a quality function that prunes the search space for maximal itemsets and an approximation measure that guides the search to patterns with different characteristics. Empirical studies have shown that our method can generate a small collection of maximal patterns fast, while preserving good quality. This method has been integrated into MIME, a framework for visual pattern mining and data exploration.

A more thorough study on the effect of approximation measures is still to be done. At last we still have to explore other quality and approximation measures that can be used in our sampling framework.

Acknowlegdements

We would like to thank T. Mitchell-Jones for providing us a copy of the mammals dataset.

7. REFERENCES

[1] F. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *Proc. ACM SIGKDD*, pages 12–19. ACM, 2004.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. VLDB*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[3] M. Al Hasan and M. J. Zaki. Musk: Uniform sampling of k maximal patterns. In *SDM*, pages 650–661, 2009.

[4] M. Al Hasan and M. J. Zaki. Output space sampling for graph patterns. *Proc. VLDB Endow.*, pages 730–741, 2009.

[5] R. J. Bayardo, Jr. Efficiently mining long patterns from databases. *SIGMOD Rec.*, pages 85–93, 1998.

[6] M. Boley, S. Moens, and T. Gärtner. Linear space direct pattern sampling using coupling from the past. In *Proc. ACM SIGKDD*, pages 69–77. ACM, 2012.

[7] C. Borgelt. Efficient implementations of apriori and eclat. In *Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90*, 2003.

[8] T. Calders and B. Goethals. Non-derivable itemset mining. *Data Min. Knowl. Discov.*, pages 171–206, 2007.

[9] V. Chaoji, M. Al Hasan, S. Salem, J. Besson, and M. J. Zaki. Origami: A novel and effective approach for mining representative orthogonal graph patterns. *Stat. Anal. Data Min.*, pages 67–84, 2008.

[10] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, pages 207–229, 1992.

[11] L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Comput. Surv.*, 2006.

[12] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Transportation Research Board*, 2003.

[13] B. Goethals, S. Moens, and J. Vreeken. Mime: a framework for interactive visual pattern mining. In *Proc. ACM SIGKDD*, pages 757–760. ACM, 2011.

[14] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, pages 1–12, 2000.

[15] D. A. Keim. Information visualization and visual data mining. *IEEE Trans. on Vis. and Comp. Graph,*, pages 1–8, 2002.

[16] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup discovery with cn2-sd. *J. Mach. Learn. Res.*, pages 153–188, 2004.

[17] M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. *ACM Trans. Knowl. Discov. Data*, pages 16:1–16:42, 2012.

[18] A. Mitchell-Jones. *The Atlas of European Mammals.* Poyser Natural History. T & AD Poyser, 1999.

[19] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. Int. Conf. on Data. Th.*, pages 398–416. Springer-Verlag, 1999.

[20] J. Vreeken, M. Leeuwen, and A. Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, pages 169–214, 2011.

[21] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for discovery of association rules. *Data Min. Knowl. Discov.*, pages 343–373, 1997.

[22] F. Zhu, X. Yan, J. Han, P. S. Yu, and H. Cheng. Mining colossal frequent patterns by core pattern fusion. In *IEEE Int. Conf. on Data Eng.*, pages 706–715. IEEE, 2007.