Sampling Frequent and Diversified Patterns Through Compression

Abstract

Exhaustive methods of pattern extraction in a database pose real obstacles to speed and output control: a large number of (potentially redundant) patterns are extracted. Pattern extraction methods through sampling, which allow controlling the size of the output while ensuring fast response times, provide a solution to these two problems. However these methods may still extract redundant patterns if a huge number is needed. For a preference learning task, we need to extract a set of patterns with low redundancy. Furthermore, some may extract infrequent patterns and can take a long time with some databases. To ensure a more diversified output, we propose integrating compression methods into sampling to select the most representative patterns from the sampled transactions and to guide the sampling process. We demonstrate that our approach improves the state of the art in terms of diversity of output patterns.

CCS Concepts

• **Information systems → Information extraction**.

Keywords

Pattern mining, Sampling, Diversity, Compression

ACM Reference Format:

. 2025. Sampling Frequent and Diversified Patterns Through Compression. In *Proceedings of ACM SAC Conference (SAC'25).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 Introduction

Pattern mining is a fundamental data mining task, extracting patterns to be either interpreted directly by domain experts, or to be used as descriptors in downstream tasks, such as classification or clustering. Exhaustive methods of pattern mining suffer from scalability and unmanageable output size. In this paper, we adopt *output pattern sampling*, another approach that has been proposed recently: instead of enumerating all patterns, patterns are sampled one by one, according to a probability distribution that is proportional to a given measure of interest. In [1] (Chapter 8), it is argued that one should ask for a small (easily interpretable) and non-redundant (with high diversity) set of patterns of interest. This is precisely what sampling approaches to pattern mining aim to do. The promised benefits include [11]: 1) *flexibility* in that potentially a broad range of quality measures can be used; 2) *anytime* pattern exploration, where a growing representative set of patterns can be generated and inspected at any time; 3) *diversity* in that the generated sets of patterns are independently sampled from different regions of the solution space. The usefulness of sampling has been widely demonstrated in recent years in many areas like feature classification [5] and interactive discovery [10, 13]. It has also been applied to several pattern languages such as graphs [12], itemsets [6, 11], and sequences [9].

We consider transactional (binary) datasets and propose an approach for sampling patterns based on the Minimum Description Length (MDL) principle [16, 18, 19], while ensuring convergence/scalability and a good diversity. This approach relies on Krimp [19] which allows one to select patterns compressing the dataset the most, where compression is used for model (i.e. set of patterns) selection rather than for reducing the number of bits. Our approach consists of two solving steps. In the first step, we exploit information from compression to sample a set of transactions. In the second step, we first mine a set of candidate patterns from the sampled transactions and then select from these candidates a subset of patterns using Krimp. We show the interest of our approach on the UCI datasets.

The paper is organized as follows. Section 2 introduces definitions and notation. Section 3 reviews the related work. Section 4 presents our contribution. Section 5 provides an empirical validation that the proposed approach improves the diversity of the sampled patterns while reducing the execution time. Section 6 gives directions of future work.

2 Preliminaries

2.1 Itemsets and Sampling

Let $\mathcal{I}$ be a set of *n items*. A *pattern* x is a non-empty subset of $\mathcal{I}$. The *pattern language* relative to $\mathcal{I}$ is the set of all $2^{|\mathcal{I}|}$ patterns using items from $\mathcal{I}$,denoted $\mathcal{L}_{\mathcal{I}}$. A transactional dataset $\mathcal{D}$ is a set of transactions, where each *transaction* $t \subseteq \mathcal{I}$. Table 1 gives an example of transactional dataset with four transactions. A transaction $t \in \mathcal{D}$ is said to *contain* pattern x if $x \subseteq t$. The *cover* of x in $\mathcal{D}$ is the set of transactions in which it appears: $cov_{\mathcal{D}}(x) = \{t \in \mathcal{D} \mid x \subseteq t\}$. The number of transactions in $\mathcal{D}$ containing pattern x is called the (absolute) *support* of pattern x, denoted $sup(\mathcal{D}, x) = |cov_{\mathcal{D}}(x)|$. The *area* of a pattern is defined by $area(\mathcal{D}, x) = sup(\mathcal{D}, x) \times |x|$. A pattern x is said to be *frequent* in $\mathcal{D}$ if $sup(\mathcal{D}, x) \geq \theta$, where θ is a user-defined threshold. Given $T \subseteq \mathcal{D}$, $\boldsymbol{i}(T)$ is a set of items that are common to all transactions in T: $\boldsymbol{i}(T) = \{i \in \mathcal{I} \mid \forall t \in T, i \in t\}$. The *closure* of a pattern x is the set of items that are contained in all transactions of $cov_{\mathcal{D}}(x)$: $clos(\mathcal{D}, x) = \bigcap_{t \in cov_{\mathcal{D}}(x)} t$. A pattern x is said to be *closed* if $clos(\mathcal{D}, x) = x$.

An important problem in local pattern mining is that of redundancy among patterns. One way to measure this redundancy between two patterns is by measuring the similarity between their respective cover. The similarity between two patterns can be measured using any measure between the two transaction sets, for example the Jaccard index. Given two patterns x and y, the Jaccard

SAC'25, March 31 –April 4, 2025, Sicily, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/xx.xxx/xxx_x

Table 1: Example of transactional dataset $\mathcal{D}$.

Trans.	Items
t_1	A B
t_2	A C
t_3	B C
t_4	A B C D

Table 2: Pattern language of the dataset from Tab.1

x	A	B	C	D	AB	AC	AD	BC
sup	3	3	3	1	2	2	1	2
x	BD	CD	ABC	ABD	ACD	BCD	$ABCD$	
sup	1	1	1	1	1	1	1	

index is the relative size of the overlap of their covers: $Jacc(x, y) = \frac{|cov_{\mathcal{D}}(x) \cap cov_{\mathcal{D}}(y)|}{|cov_{\mathcal{D}}(x) \cup cov_{\mathcal{D}}(y)|}$. In this paper, we consider the Jaccard similarity coefficient as a measure of diversity between pairs of patterns. This index varies between 0 and 1, with a value close to 0 indicating high diversity. Originally, the goal of a pattern sampling technique is to access the pattern space $\mathcal{L}_I$ through an efficient sampling procedure simulating a distribution $\pi : \mathcal{L}_I \rightarrow [0, 1]$ which is defined with respect to some measure of interest $m : \pi(.) = m(.)/Z$, where Z is a normalization constant, the sum of the interest of all patterns $x \in \mathcal{L}_I$ in the dataset $\mathcal{D}$, defined by: $Z = \sum_{x \in \mathcal{L}_I} m(x)$. Formally, the sampling of k patterns of the language $\mathcal{L}_I$ according to a distribution proportional to an interestingness measure m in a database $\mathcal{D}$ can be formulated as follows [9]:

$$\mathcal{S}ample_k(\mathcal{L}_I, \mathcal{D}, m) = \bigcup_{i=1}^{k} \{x_i \sim m(\mathcal{L}_I)\} \tag{1}$$

where k represents the number of samples to extract, and $x \sim m(\mathcal{L}_I)$ means that the pattern x is drawn with a probability proportional to its interest m: $x \sim m(\mathcal{L}_I) \Leftrightarrow \pi(x) = m(x)/Z$. In the literature, there are several interestingness measures, the most popular being the frequency measure.

Example 1. *Table 2 gives all patterns extracted from $\mathcal{D}$ of Table 1 and their corresponding support. The normalization constant Z is the sum of the pattern supports: $Z = \sum_{x \in \mathcal{L}_I} sup(x) = 24$. Within the dataset $\mathcal{D}$, we have $sup(BC) = 2$ and $sup(AD) = 1$. This means that $\pi(BC, \mathcal{L}_I) = 2/24$ and $\pi(AD, \mathcal{L}_I) = 1/24$. So, the probability to draw BC is twice that of AD in the dataset $\mathcal{D}$.*

The computation of the normalization constant suggests an exhaustive enumeration of all patterns in $\mathcal{D}$. To avoid the expensive step of exhaustively generating patterns before their sampling, several sampling methods use random walks in the search space to simulate a distribution from a probability law and draw patterns proportionally to this distribution [5, 12]. Some important methods in this category are Markov Chain Monte Carlo (MCMC) techniques [4]. In MCMC methods, iterations are not independent and blindly sampled but every iteration/step of Monte Carlo is dependent on its previous iteration/step. This feature is referred to as the Markov property [8]. The idea is to construct a Markov chain simulating a distribution from a probability law where each state E_i (representing a set of transactions) depends only on the state E_{i-1} and not on others. The multi-step approaches first sample a subset of transactions from the database, and from this subset draws a pattern by direct sampling.

2.2 Data compression and MDL principle in pattern mining

The *Minimum Description Length* (MDL) principle [15] is a technique from the domain of information theory that allows to select the model, from a family of models, that best describes some data. The MDL principle states that the best model $\mathcal{X}$ for describing some data $\mathcal{D}$ is the one that minimizes the description length $L(\mathcal{D}, \mathcal{X}) = L(\mathcal{X}) + L(\mathcal{D}|\mathcal{X})$, where $L(\mathcal{X})$ is the length of the model and $L(\mathcal{D}|\mathcal{X})$ the length of the data encoded with the model.

When applying the MDL principle to pattern mining, the models considered consist of patterns capturing structure and regularities in the data. Depending on the type of data and the application at hand, one must decide what kind of patterns are relevant, i.e. a pattern language must be chosen. The model $\mathcal{X}$ then consists of all possible subsets of patterns from the chosen pattern language. The task is to find a set of patterns $\mathcal{X}$, denoted a *Code Table* (CT), that best describes the database $\mathcal{D}$.

Different approaches using the MDL principle [16, 17, 19] associate each pattern $x \in \mathcal{X}$ with a code C_x in such a way that all transactions $t \in \mathcal{D}$ are described by a set of codes $C(t)\{C_1, C_2, \ldots\}$. A correspondence table referred to as *Code Table* is then used to associate each pattern x with a code C_x and the list of transactions in which x is used. Encoding the data using the model (the $L(\mathcal{D}|\mathcal{X})$ part) can be fairly straightforward, simply replacing occurrences of the patterns in the data by their associated codes. Knowing how many times each pattern x is used in the description of the data $\mathcal{D}$ with model $\mathcal{X}$, denoted as $usage_{\mathcal{X},\mathcal{D}}(x) = |\{t \in D : C_x \in C(t)\}|$, x can be assigned a code of length $L(x) = -log_2(\frac{usage_{\mathcal{X},\mathcal{D}}(x)}{\sum_{y \in \mathcal{X}} usage_{\mathcal{X},\mathcal{D}}(y)})$. With this code table, each transaction possibly gets smaller. If we divide its new size by the original size, obtained using the standard code table ST, i.e. the set of items $\mathcal{I}$, we get a compression ratio:

$$CR(t, \mathcal{X}) = \frac{L(t, \mathcal{X})}{L(t, ST)} \tag{2}$$

Krimp [19] is a pattern set mining algorithm using the MDL principle to select a "characteristic" set of patterns from a transactional database. It takes as input a dataset and a set of previously extracted patterns (such as frequent or closed patterns) and aims to find the subset of patterns that best compresses the dataset. To obtain this description, Krimp starts by initializing the code table with singleton patterns (containing a single item). The patterns being added to Code Table are chosen from a candidate set. The candidate set is a subset of frequent (closed) patterns that are ordered w.r.t. their frequencies, lengths and lexicographically. In the Krimp algorithm, at each iteration a non-singleton pattern is considered as a candidate to Code Table. It is added to Code Table if it allows for a smaller encoding length, otherwise it is removed from the code table and candidate set. The algorithm stops when the candidate set is empty.

3 Related Work

Sampling algorithms can circumvent the negative complexity results of exhaustive approaches. By high sampling accuracy, they might also give rise to a diverse result set, i.e., one originating from different regions in the solution space. The following approaches are the most well-known and recent ones.

Gibbs [2] is based on using subjective measures of interest [3], to be updated with information resulting from already mined patterns to ensure that future patterns add new information. This subjective measure of interest then directs a Gibbs sampling process to sample patterns, which is expected to reduce running times.

CFTP [6], samples patterns directly, using an objective quality measure. As such, this method is intended to give results very quickly that might, or might not, show diversity. It uses a two-step random sampling procedure tailored for a restricted set of itemset mining tasks. Patterns' probability of being sampled is related to their score but ignores patterns sampled before.

Flexics [11] uses a collection of random XOR constraints to partition the overall solution space into several non-overlapping bins, and then draws patterns from these bins. The bins are small enough to be efficiently mined using either a CP approach or a dedicated itemset miner. The XOR constraints enforce diversity in the data, which in theory could give rise to the diversity of patterns.

Bosc et al. [7] propose using Monte Carlo Tree Search (MCTS) and upper confidence bounds to direct the search towards interesting regions in the lattice given the already explored space. While MCTS is necessarily randomized, it iteratively extract up to k subgroup descriptions (similarly to our work) through a guided random search with a time budget. Notably, this approach requires a target attribute and a target value to focus on while our approach allows for unsupervised mining.

Except for Flexics, there is no straightforward way of adapting the above approaches to sampling closed sets. In contrast, with our sampling approach closed patterns can be generated efficiently according to some controlled target distribution. While the worst-case time complexity of the Markov chain step of CFTP can grow exponentially in the database size, our approach allows an efficient computation of a set of transactions using information derived from compression. To the best of our knowledge, this is the first time that information from compression is exploited for sampling.

4 A Compression-Oriented Sampler

We present our contribution: the Lead by Compression Sampler LCS approach. Similarly to CFTP, we proceed in two steps. However, contrary to CFTP [6], in the first step, we use the compression ratio of transactions as their weights to sample a set of c transactions. Furthermore, in the second step, instead of drawing a pattern proportional to an interestingness measure, the frequency, we first mine a set of candidate patterns and then select from these candidates a subset of patterns using Krimp. The intuition behind LCS is that compression reduces redundancy implicitly, by selecting a set of patterns that best compresses the underlying database. Two syntactically very similar patterns are unlikely to be chosen together since their contribution to compression is probably redundant.

Require: A database $\mathcal{D}$, a weight vector w, target number of transactions c
Ensure: A set S of c transactions

$w_{max} \leftarrow max(w)$
$S \leftarrow \emptyset$
while $|S| < c$ **do**
  $t \sim U(D)$
  if $t \notin S$ and $p \sim U(0,1) < \frac{w_t}{w_{max}}$ **then**
    $S \leftarrow S \cup \{t\}$
  end if
end while
return S

Figure 1: Algorithm to sample a subset of transactions

4.1 Transactions sampling

What makes CFTP (Coupling From The Past) noteworthy is its capacity to define the weight function for drawing a set of transactions for the second step using a tuple of c transactions without requiring the computation of all $\binom{|\mathcal{D}|}{c}$ tuples. However, its slow convergence to the desired target distributions makes the algorithm's performance prohibitive even for small values of c (not exceeding 4). This is particularly more pronounced on sparse datasets.

To address this problem, we propose to sample a set of c transactions using their compression ratio (see Formula 2) which serves as a guide for the sampling process. Initially, each transaction t is assigned a weight $w(t)$ of 1, and these weights are updated according to the current code table computed by Krimp at each iteration of sampling (see Algorithm 2). We adopt the methodology described in [14] based on the stochastic acceptance of a randomly selected individual without replacement. Our algorithm consists of the following steps (see Algorithm 1): First, select randomly one of the transactions (say, t). The selection is done with uniform probability $(1/|\mathcal{D}|)$, which does not depend on the transaction's weight w_t. Second, with probability $\frac{w_t}{w_{max}}$, where $w_{max} = \max\{w_i\}_{i=1}^{|\mathcal{D}|}$ is the maximum weight over all transactions, the selection is accepted. We repeat until we reach the desired sample size.

The authors demonstrated that drawing one sample typically requires constant time. They posit that the closer the ratio $\frac{w_{max}}{w_{mean}}$ is to 1, the faster the algorithm will be. In our case, this ratio should increase, though we will show in our experiments that this increase does not harm performance. Using this method ensures fast sampling regardless of the size or density of the database.

4.2 Pattern extraction

Once c transactions have been sampled, instead of drawing a pattern proportionally to a measure of interest, as in CFTP, we extract all closed patterns that are present in at least θ sampled transactions, where $\theta = \max(1, f \cdot c)$ with frequency threshold $f \in]0,1[$. This set of closed patterns is then added to the candidate set for Krimp. This allows identifying in the data a collection of patterns that yields a good compression. Additionally the use of closed patterns prevents generating many more candidates than necessary and allows reducing the redundancy between candidates. Because compression costs following the addition of candidate patterns is often prohibitively

expensive, using closed patterns allows to efficiently and accurately bound these costs.

4.3 Compression with Krimp

Compression here is exploited as a tool to compare models. We use Krimp to filter candidate patterns after sampling. With a set of candidate patterns denoted as C, Krimp will heuristically select patterns that form the minimal-length code table $\mathbf{X} \subseteq C$. The intuition behind using Krimp stems from the fact that finding the Minimum Description Length (MDL) is equivalent to identifying a set of patterns with large area values, and these areas should not overlap significantly, nor have a high Jaccard index. We will demonstrate through experiments that our approach achieves better results than those reported in the literature in terms of diversity. It is worth noting that our approach takes full advantage of the compression step to guide sampling to the least compressed segments of the database. Using compression is very beneficial because the new weights are computed simultaneously during compression. No additional work is therefore required to compute them. Moreover, the computed model will iteratively be enriched with new extracted information. As mentioned earlier, we use the compression ratio to compute the new weights as follows: $w(t) = min(1, CR(t, \mathbf{X}))$ avoiding weights above 1, which could misdirect the sampling. Having a compression ratio greater than 1 indicates that certain patterns are used less frequently than before compression. A weight above 1 would give more weight to transactions with items already used in other patterns from the code table.

4.4 Our approach in a nutshell

Figure 2 gives the pseudo-code of our approach. The sampling process proceeds iteratively for some reasonable number of iterations n. The algorithm maintains two internal data structures: the weight vector w associated to transactions of $\mathcal{D}$ and the code table CT. Initially, the code table is initialized by the standard code table and the weight of each transaction is set to 1. At each iteration, the algorithm samples a set of c transactions by exploiting the current weight vector w. The set of all closed patterns with a frequency of at least θ is then extracted using the LCM algorithm (line 8). The extracted patterns are added to those already in CT (line 9). Then they are used by Krimp to compute the new code table, which yields a good compression, and the weight vector w is updated accordingly (line 10).

Require: database $\mathcal{D}$, number of transactions c, number of samplings k, frequency threshold f
Ensure: A diverse set of patterns
1: $CT \leftarrow$ StandardCodeTable()
2: $\theta = \max(1, f \cdot c)$
3: **for all** $t \in \mathcal{D}$ **do**
4: $w(t) \leftarrow 1$
5: **end for**
6: **for** $i = 1 \ldots k$ **do**
7: $S \leftarrow$ sampleTrans($\mathcal{D}, w, c$)
8: $newCand \leftarrow$ minePatterns(S, θ)
9: $cand \leftarrow CT \cup newCand$
10: $CT, w \leftarrow$ Krimp ($\mathcal{D}, cand$)
11: **end for**
12: **return** CT

Figure 2: Lead by Compression Sampler (LCS)

5 Experiments

We assess the performance of our LCS method by analyzing the diversity of the sampled patterns and the execution time. We use nine databases from the FIMI repository[1], the UCI Machine Learning repository[2], the CP4IM repository[3], and a real case study based on biological yeast. These databases exhibit diverse characteristics, varying in size and database density (see Table 3). We first discuss the parameter settings of our algorithm, then we compare it to other methods. For our comparison, we selected the following approaches: Gibbs [2], CFTP [6], and Flexics [11]. Whenever sampling is involved, we report average results over 10 runs. We also compare to Krimp in terms of runtime and compression rate.

The Krimp algorithm is implemented in C++. Our algorithm is implemented in Python[4]. Implementations of Flexics, CFTP and Gibbs were made available to us by the authors. Flexics is implemented in Scala, while CFTP and Gibbs are implemented in Java. The experiments were conducted on a 12th Gen Intel® Core™ i7-12700H processor with the Linux operating system and 4 GB of RAM. Krimp is run in parallel with four cores for compression. The timeout is set to one hour for analysis purposes but a decent time for sampling should be less than a minute. To quantify the diversity of a set of solutions, we use the pairwise Jaccard as quality measure. Since probability density functions can be prone to small changes in the distribution resulting in visually rather distinct shapes, we instead use *cumulative density functions* (CDF) over pairwise Jaccard values. Given a set of patterns $\mathcal{S} = \{X_1, \ldots, X_n\}$, the exact function plotted is the following:

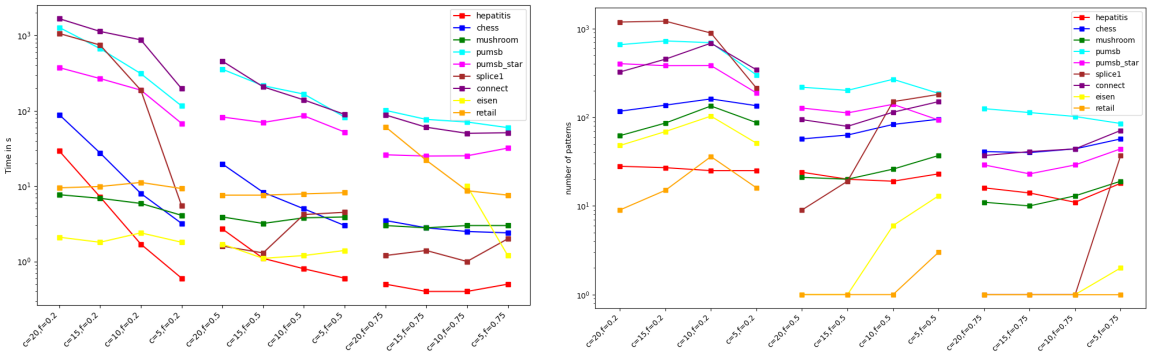
$$CDF(\tau) = \frac{2 \cdot |\{(i,j) \mid Jac(X_i, X_j) \leq \tau, 1 \leq i < j \leq k\}|}{k(k-1)} \quad (3)$$

Factor $\frac{2}{k(k-1)}$ is a normalization factor (to have a result between 0 and 1). For any given Jaccard value τ, $CDF(\tau)$ indicates which percentage of pairs of patterns have Jaccard values below τ. This also means that curves further to the left are better (since a low Jaccard indicates less redundancy), as are curves further to the top.

5.1 Parameter analysis of LCS

Our approach has three parameters: the number of transactions c to sample, the number of iterations k and the threshold f to mine closed frequent patterns in the transactions sample. Each parameter is set with predefined values, $c \in \{5, 10, 15, 20\}$, $k = 25$ and $f \in \{0.2, 0.5, 0.75\}$. To asses our choice of extracting all closed patterns present in at least two transactions from the transactions sample, we compare different combinations of parameter values f and c. Figure 3a shows the running time (in seconds) for each dataset and each pair of values of c and f. Each color represents a

[1] http://fimi.uantwerpen.be/data/
[2] https://archive.ics.uci.edu/datasets
[3] https://dtai.cs.kuleuven.be/CP4IM/datasets/
[4] The source code is available at https://anonymous.4open.science/r/LCS-papier-1B57



(a) Comparing CPU-times (b) Comparing the number of patterns extracted

Figure 3: Analysing LCS for various combinations of values for f and c.

Table 3: Information about the databases

name	#Trans.	# Item	Avg I/T	density %
hepatitis	137	68	34	50.00
mushroom	8124	119	23	19.20
splice	3190	287	60	20.90
chess	3196	75	37	49.30
eisen	2464	9839	37	0.30
pumsb	49046	2113	74	3.50
pumsb*	49046	2088	51	2.40
connect	67557	129	43	33.00
retail	88162	16470	10	0.06

different dataset. As we can see, for $f = 0.75$ LCS takes less time on almost all of the datasets to sample patterns. The running time increases slightly with c, except for the two very sparse datasets Eisen and Retail, where the running time increases drastically. Indeed, a high value for c and f requires frequently occurring patterns which makes it difficult in sparse databases. Lowering f increases the number of patterns extracted (see Figure 3b), thus increasing the running time of all the steps of the algorithm.

Figure 4 shows the evolution of the cumulative functions of the Jaccard index for different combinations of f and c on the Chess, Splice and Pumsb datasets. With $f = 0.75$, we see that the number of patterns extracted by LCS with different configurations is the lowest and the CDF curves are shifted further to the right indicating higher pairwise Jaccard values, while for $f = 0.2$ LCS always extracts a very large number of patterns for which the pairwise Jaccard is very low (i.e. CDF curves are found in the top-left corner). This is due to the fact that patterns sampled with higher values of f are much more redundant as they are more likely to cover similar transactions and with extremely sparses datasets, like splice, it can lead to getting few redundants patterns. The best results for CDFs are generally obtained when the threshold resulting from f and c is small. For $c = 5$, we can see that pairwise Jaccard values are always very low, as seen since the CDF curves stay higher than those of other values of c. They rise more steeply, though, and eventually cross the curves of $c = 5$. When increasing the values of c, the threshold values derived from f may lead to a timeout. This is the case for the Eisen dataset where we reach a timeout for $c > 15$ and $f = 0.75$. So, keeping a low value for f is better but if $\theta = 1$ it will add at most c useless closed patterns with an absolute support of 1. We therefore decided to set $\theta = 2$, thus avoiding having to manage an additional parameter for our method.

Figures 5-6 analyze the impact of the number of iterations on LCS w.r.t. different performance criteria on the Connect dataset, but similar observations can be made on other datasets. We can see that at the beginning the compression ratio greatly increase, then almost stagnates after a certain number of iterations between 25 and 30. We can also observe that the best performance is achieved with large values of c. The impact of the number of iterations on the running time is almost linear for any configuration. When c increases, time grows exponentially. For $k = 25$, with $c = 5$ it takes 100 seconds and with $c = 20$ it goes up to 3000 seconds. Regarding the pairwise Jaccard values, the diversity is always good and gets better with each iteration. Based on these findings, we set $k = 25$, $c = 10$ and $f = 0.2$ as our default parameters.

5.2 Comparison to existing Sampling methods

Flexics and Gibbs require different parameter settings. Flexics needs a minimum frequency threshold to mine patterns and an error tolerance $\kappa \in [0, 1]$. We run Flexics with values of $\kappa \in \{0.1, 0.5, 0.9\}$. We have selected, for every dataset, three frequency thresholds to have different numbers of frequent closed itemsets Th ($|Th| \leq 15k$, $30k \leq |Th| \leq 10^6$, and $|Th| > 10^6$). We used LCM as a baseline to determine suitable thresholds. We denote a combination of Flexics and minimum frequency threshold setting by concatenation. For example, "Flexics-$t = 0.2$" means a configuration of Flexics with the minimum frequency threshold at 20%. We report results corresponding to the best setting for parameter κ (i.e. $\kappa = 0.9$). CFTP is evaluated with different values of $c \in \{3, 4\}$. Gibbs sampling requires to tune an iteration number g. We followed the same experimental protocol proposed by [2] and fixed the number of iterations g to 1 000, and 10 000. "Gibbs-$g = 1\,000$ means that we allow Gibbs to perform 1 000 steps. For LCS we use the default parameters stated earlier but also show combinations ($c = 5, f = 0.4$), ($c = 15, f = 0.1$), and ($c = 20, f = 0.1$) for reference purposes.

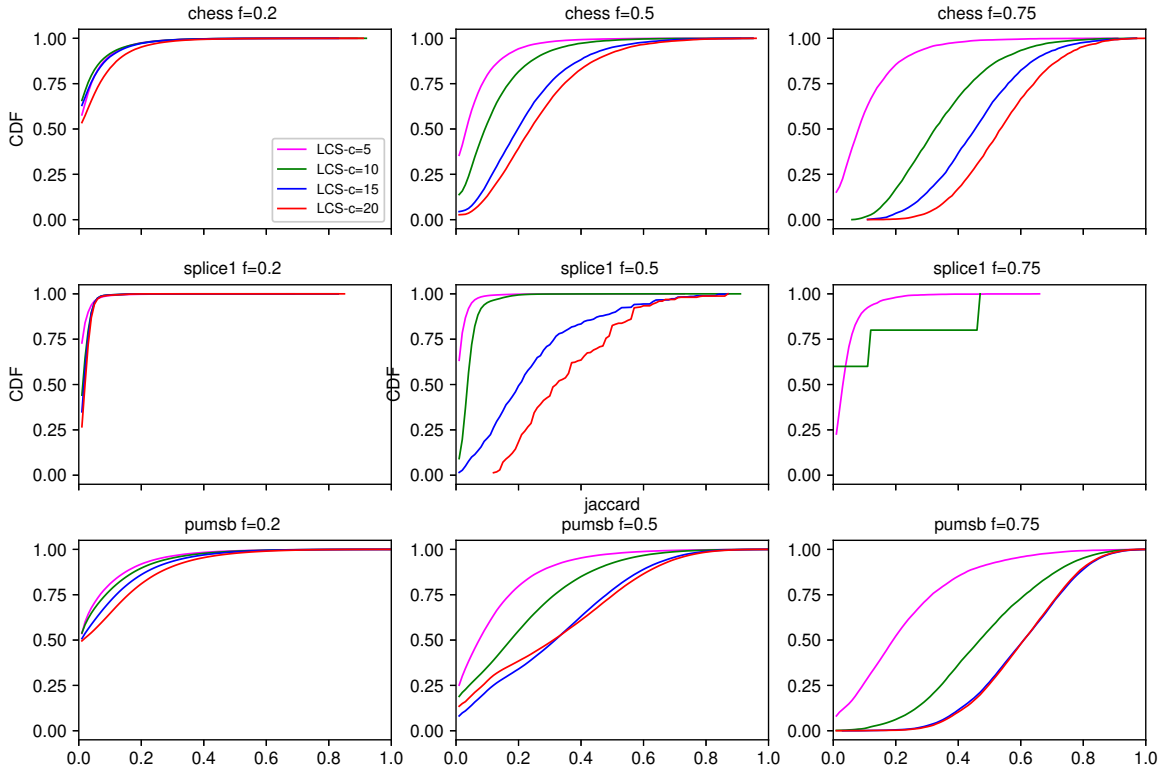


Figure 4: Assessing overall pairwise redundancy of LCS for various combinations of values for f and c on Chess, Pumsb and splice datasets. The x-axes are labeled with the pairwise Jaccard (multiplied by 100 for better readability), and the y-axes with proportions between 0 and 1.0. For any given Jaccard value, the curves indicate which percentage of pairs of patterns have Jaccard values below that value.

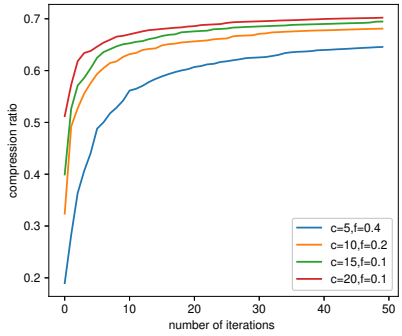


Figure 5: Compression ratio analysis of LCS regarding number of iterations k for some configuration on Connect dataset.

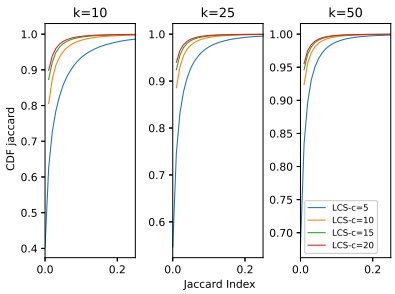


Figure 6: Cumulative distributions of the sampled patterns for various configurations of LCS on Connect.

To compare the performance of the different samplers, we asked Flexics, CFTP and Gibbs to sample the same number of patterns returned by the LCS configuration ($c = 10$, $f = 0.2$).

5.2.1 Quality of patterns diversification. Starting with diversity analysis of the result set, Figure 7 shows representative CDFs for LCS and other methods on the Chess, Connect, Pumsb, Pumsb_Star, Eisen and Retail datasets. As we can see, Gibbs curves stay below those of the other techniques. A deep analysis of the result set shows that Gibbs often samples the same area, thus leading to the same patterns. For dense datasets like Chess or Connect, almost half of the patterns extracted are the same. On Connect, up to 200 patterns are sampled multiples times. Consequently, this greatly impacts the diversity of the resulting set. For sparse datasets like Pumsb, Gibbs is quite good. For Retail,

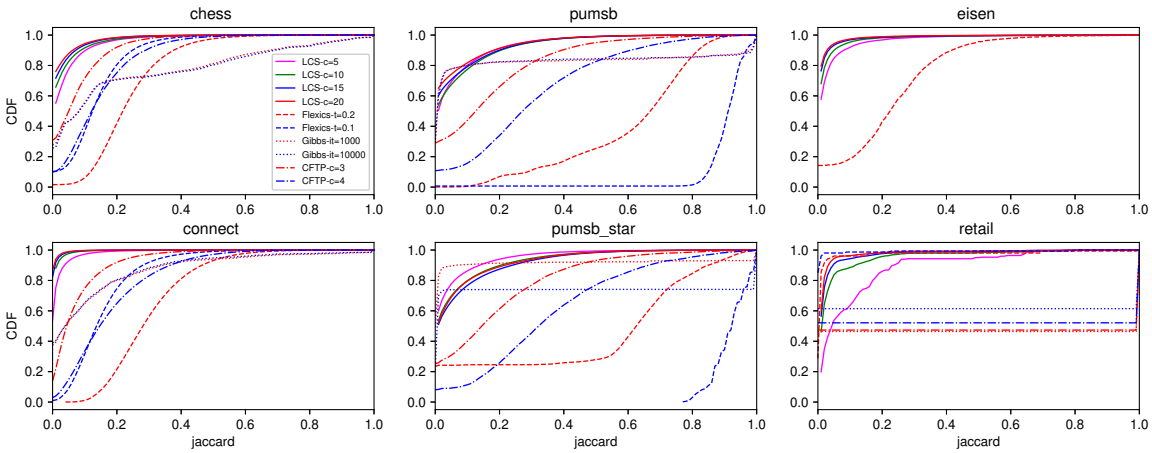


Figure 7: Cumulative distribution function of the Jaccard index for each configuration on six datasets (Chess, Connect, Pumsb, Pumsb_Star, Retail and Eisen).

Gibbs samples differents patterns but from the same transactions thus leading to this linear CDF. Surprisingly, for Eisen, Gibbs could only sample from the same transaction, so there is no curve to see. CFTP assembles low-redundancy sets on the sparse datasets, due to sampled patterns having very low support, as we will discuss later on. Flexics leads to a decent distribution of Jaccard values, albeit worse than for the resulting sets of LCS. We can see that Flexics could achieve a good diversity at very low thresholds but it would take too much time to sample patterns for Chess or Eisen. However, as shown on Pumsb, lower threshold values may degrade the CDFs. This could come from the number of XOR constraints needed to reduce the size of cells. Nonetheless, Gibbs can rarely beat Flexics such as on Retail. Finally, LCS clearly achieves the best results for almost all of the datasets, even with low values of c.

5.2.2 Characteristics of the mined patterns. We investigate what kind of patterns are mined by each approach. We show in Figure 8 the scatter plots of the points obtained from two pattern descriptions, i.e., length and cover size for four datasets (Chess, Pumsb, Retail and Eisen). This plot tells us about the shape of patterns sampled by each approach. On dense datasets, Gibbs favors sampling short patterns widely distributed in terms of cover size, while CFTP favors sampling long patterns covering only a few transactions on sparse datasets. Our approach returns well-distributed patterns in terms of length and cover size for any value of c. Flexics is also able to return patterns with diverse lengths but less distributed in terms of cover size. On very sparse datasets Retail and Eisen, although the length of sampled patterns by Gibbs varies, they, unfortunately, cover only one or two transactions.

5.2.3 Performance evaluation. Figure 9 compares the running times of LCS ($c = 10, f = 0.2$) with Flexics, CFTP and Gibbs for different datasets and different settings. We also include the results of LCS with ($c = 20, f = 0.1$). For each dataset, the different samplers are requested to generate the same number of samples returned by the LCS configuration ($c = 10, f = 0.2$). More precisely, for each dataset (Hepatitis, Chess, Mushroom, Pumsb, Pumsb_star, splice, Eisen,

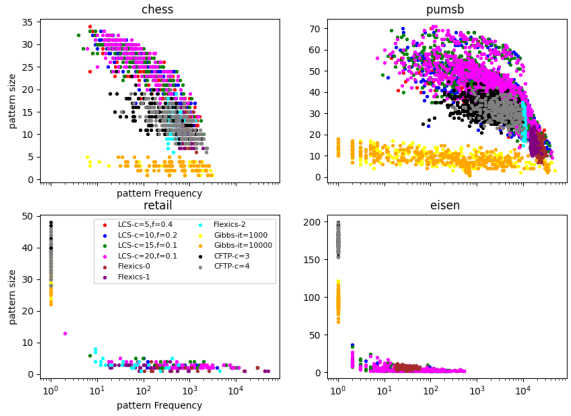


Figure 8: Scatter plots of the sizes of the patterns and their covers across various datasets (Chess, Pumsb, Retail and Eisen).

Retail and Connect) we respectively asked for 27, 180, 125, 609, 322, 547, 92, 28, and 590 patterns. These are the number of pattern get with $c = 10, n = 25, f = 0.2$. For CFTP, we only report the results of $c = 4$ since for high values of c (i.e. $c \geq 5$), CFTP almost reaches the timeout of one hour. For sparse datasets, even with $c = 4$ CFTP was not able to finish in one hour. The two configurations of Flexics correspond to the two frequency thresholds used to obtain the two sets Th: Flexics-0 for $|Th| \leq 15k$ and Flexics-2 for $|Th| > 10^6$.

We first clearly see that LCS ($c = 10, f = 0.2$) largely dominates Flexics, being more than an order of magnitude faster, illustrating its usefulness for sampling patterns in an anytime manner. The only exception are the two datasets Connect and Pumsb_star where Flexics is faster. Second, CFTP is almost always faster than LCS in the settings with c between 2 and 3. On these instances, indeed, the computation can be performed in less than 10 seconds on dense datasets. Third, we can observe that Gibbs provides runtime

Table 4: Comparison to Krimp in terms of compression ratio (CR) and time in seconds.

Dataset	LCS								Krimp		
	$c = 5$		$c = 10$		$c = 15$		$c = 20$				
	Time	CR	Time	CR	Time	CR	Time	CR	Config	Time	CR
hepatitis	0.6	31.4	1.7	37.3	8.4	39.2	39.5	41.0	clos-20%	32.4	54.6
chess	2.9	52.1	8.0	60.5	28.3	62.8	106.7	64.1	clos-20%	100.9	62.8
mushroom	3.9	49.4	5.9	59.7	8.5	63.8	11.3	64.7	clos-0.03%	1.2	75.4
pumsb	113.0	45.8	311.8	55.0	671.5	58.2	1394.4	60.0	clos-50%	2129.1	41.6
pumsb_star	63.9	42.0	188.3	51.7	335.3	54.2	517.7	55.9	clos-10%	3226.2	56.6
splice	5.6	5.7	188.1	19.4	781.8	24.1	1361.8	25.4	clos-1%	824.2	31.0
connect	186.7	61.2	875.9	66.4	1876.5	68.1	3589.4	69.3	clos-10%	1724.1	68.9
eisen	1.6	2.7	2.4	3.9	2.8	4.8	3.2	5.6	clos-0.1%	6.3	21.2
retail	9.4	1.5	11.2	2.0	12.0	2.1	13.5	2.3	clos-0.003%	0.2	7.2

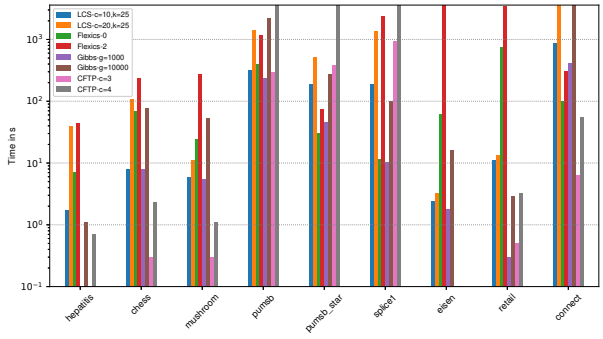


Figure 9: CPU-time analysis for different configurations of algorithms.

benefits over LCS for 1 000 iterations. However, with 10 000 iterations, LCS ($c = 10, f = 0.2$) often performs better, except on three datasets (Hepatitis, Splice-1 and Retail). Indeed, the complexity of the whole Gibbs sampling procedure increases when the number of iterations g becomes larger.

5.3 Comparison to Krimp

To check the quality of patterns extracted with LCS, we compare to Krimp, selecting some configurations that terminate within the time limit. A Krimp configuration is written as "clos-θ%" which means that the pattern candidates are either frequent or closed over the thresholds θ. For Hepatitis, clos-20% means that it corresponds to all the frequent closed patterns with a frequency= 20% of database size. Table 4 reports the compression ratio and the running time of each configuration of LCS, the running time and the compression ratio for Krimp. The time reported for Krimp corresponds to the time for mining candidate sets and for compressing these candidates. Note that our approach is not yet another compression method intended to achieve a better compression. Instead we aim to evaluate the ability of our method to obtain interesting models. A better compression ratio means that the extracted patterns should overlap less. We can see that on some sparse datasets, we get a better compression ratio. For datasets with very low thresholds, Krimp clearly achieves better results. This is due to the fact that Krimp has access to a large set of patterns, therefore it has a better chance of getting a higher compression ratio.

6 Conclusion

In this paper we introduced a new approach that uses the principles of pattern sampling and data compression in order to sample more diverse patterns, while ensuring scalability. Our approach exploits an exact method, LCM, to mine a set of closed patterns from a candidate set of transactions drawn uniformly using a weighted sampling method. A compression step is then performed to select from these candidates a subset of patterns using Krimp. The intuition behind our approach is that compression reduces redundancy implicitly, by selecting a set of patterns or a new description that best compresses the underlying database. This description is then used to update weights of the transactions for subsequent iterations of sampling. Experimental results on several UCI datasets showed that the diversity of sampled patterns is significantly enhanced while greatly reducing execution times. These results confirm the interest of our approach for obtaining diverse patterns using data compression.

Our work is primarily based on frequency, but can be adaptable to various measures of interest, even sets of measures, allowing for multi-criteria sampling. By incorporating user feedback, the method could guide sampling towards parts of the database more likely to interest the user.

References

[1] C. C. Aggarwal and J. Han. 2014. Frequent pattern mining. Springer.
[2] A. Bendimerad, J. Lijffijt, M. Plantevit, C. Robardet, and T. De Bie. 2020. Gibbs Sampling Subjectively Interesting Tiles. In IDA 2020. 80–92.
[3] T. De Bie. 2011. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. Data Min. Knowl. Discov. 23, 3 (2011), 407–446.
[4] M. Boley, T. Gärtner, and H. Grosskreutz. 2010. Formal Concept Sampling for Counting and Threshold-Free Local Pattern Mining. In SDM 2010. 177–188.
[5] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner. 2011. Direct local pattern sampling by efficient two-step random procedures. In KDD 2011. 582–590.
[6] M. Boley, S. Moens, and T. Gärtner. 2012. Linear space direct pattern sampling using coupling from the past. In Proceedings of KDD 2012. ACM, 69–77.
[7] G. Bosc, J-F Boulicaut, C. Raïssi, and M. Kaytoue. 2018. Anytime discovery of a diverse set of patterns with Monte Carlo tree search. Data Min. Knowl. Discov. 32, 3 (2018), 604–650.
[8] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. 2011. Handbook of Markov Chain Monte Carlo. Chapman and Hall/CRC.
[9] L. Diop. 2020. Échantillonnage sous contraintes de motifs Structurés. Ph. D. Dissertation. Gaston Berger University, Saint-Louis, Senegal.
[10] V. Dzyuba and M. van Leeuwen. 2017. Learning What Matters - Sampling Interesting Patterns. In PAKDD 2017, Proceedings, Part I. 534–546.
[11] V. Dzyuba, M. van Leeuwen, and L. De Raedt. 2017. Flexible constrained sampling with guarantees for pattern mining. Data Min. Knowl. Discov. 31, 5 (2017), 1266–1293.
[12] M. Al Hasan and M. J. Zaki. 2009. Output Space Sampling for Graph Patterns. Proc. VLDB Endow. 2, 1 (2009), 730–741.
[13] A. Hien, S. Loudni, N. Aribi, A. Ouali, and A. Zimmermann. 2023. Interactive Pattern Mining Using Discriminant Sub-patterns as Dynamic Features. In PAKDD 2023. 252–263.
[14] A. Lipowski and D. Lipowska. 2012. Roulette-wheel selection via stochastic acceptance. Physica A: Statistical Mechanics and its Applications 391, 6 (2012), 2193–2196.
[15] J. Rissanen. 1978. Paper: Modeling by Shortest Data Description. Automatica 14, 5 (1978), 465–471.
[16] A. Siebes and R. Kersten. 2011. A Structure Function for Transaction Data. In Proceedings of SIAM SDM 2011, Mesa, Arizona, USA. 558–569.
[17] A. Siebes, J. Vreeken, and M. van Leeuwen. 2006. Item Sets that Compress. In SDM 2006. 395–406.
[18] K. Smets and J. Vreeken. 2012. Slim: Directly Mining Descriptive Patterns. In Proceedings of SIAM SDM 2012, Anaheim, California, USA. 236–247.
[19] J. Vreeken and A. Van Leeuwen, M.and Siebes. 2011. Krimp: mining itemsets that compress. Data Mining and Knowledge Discovery 23, 1 (2011), 169–214.