Trie-based Output Space Itemset Sampling

Lamine Diop (lamine.diop@univ-tours.fr)
University of Tours https://orcid.org/0000-0003-4539-2549
Cheikh Talibouya Diop
University Gaston Berger of Saint-Louis
Arnaud Giacometti
University of Tours
Arnaud Soulet
University of Tours

Research Article

Keywords: Pattern mining, Pattern sampling, Itemset, Trie data structure

Posted Date: June 16th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1285827/v2

Under consideration for publication in Knowledge and Information Systems

Trie-based Output Space Itemset Sampling

Lamine Diop[1,2], Cheikh Talibouya Diop[1],

Arnaud Giacometti[2] and Arnaud Soulet[2]

[1]University of Gaston Berger of Saint-Louis, Senegal
Email: {diop.lamine3, cheikh-talibouya.diop}@ugb.edu.sn
[2]University of Tours, France
Email: {firstname.lastname}@univ-tours.fr

Abstract. Itemset mining methods are techniques to discover relevant patterns in transactional databases. The first approach, called constrained-based pattern mining, is based on exhaustive pattern mining techniques which consist in returning all itemsets that satisfy a given constraint. The main issues that hinder their efficiency are the pattern explosion and the difficulty for a user to set the threshold value. To solve this problem, methods that return the most interesting patterns, called top-k, are also proposed, but they tend to lack diversity, a challenging issue for interactive pattern mining. However, interactive pattern mining is based on fast methods that respond effectively to user demand. To overcome all these problems, output pattern sampling is proposed to draw quickly a set of interesting patterns while guaranteeing a good diversity. Pattern sampling techniques are probabilistic methods that aim to draw a set of interesting patterns where each pattern is drawn with a probability proportional to a given interestingness measure. Nowadays, there are several measures that a user can test when interacting with the same database. In that case, the system should last a few times to consider the new utility measures while guaranteeing an exact draw. So, the cost in time of utility change can be a real problem for output pattern sampling techniques in large databases. In addition, with the current sampling methods, it is necessary to store all data in memory and this storage is prohibitive for large data. To solve these problems, this paper deals with how to structure the data for output pattern sampling under length-based utility measures in large transactional databases. So, we revisit the trie structure initially proposed by D. Knuth to enrich it and then no longer have the need (i) to access the data to sample because the patterns will be directly taken from the enriched trie, (ii) to reprocess the entire dataset when utility changes. The computation of the value of a length-based utility measure is based on the lengths of all the patterns that are present in the database. So, we define a new structure of trie called trie of occurrences, built by our first algorithm TPSpace (Trie-based Pattern Space), which materializes all the occurrences of the patterns in the database.

The data compression comes from a factorization of the information via the prefixes of the patterns. The particularly remarkable result is that, by definition, the trie of occurrences is the same for any length-based utility measure provided that the same values are kept for the minimum and maximum length constraints. We then describe TPSampling (Trie-based Pattern Sampling) which performs the sampling by drawing patterns according to a length-based utility measure from the trie of occurrences. This paper is completed by the complexity analysis in memory and in time of the method and experiments on benchmark datasets. TPSampling is competitive with the two-step approach to sample following a given interestingness measure but, as expected, it is more particularly advantageous if several utility measures are used thanks to the generic preprocessing. TPSampling is 10^5 times faster than Two-Step for reprocessing in utility change.

Keywords: Pattern mining, Pattern sampling, Itemset, Trie data structure

1. Introduction

Pattern mining [1] is an active research area that aims at discovering interesting and not trivial information in large databases. Itemset mining methods are techniques to discover relevant patterns in a transactional database. During the last decade, the researchers in this field addressed the pattern explosion challenge resulting from the joint effect of the volume of data and the combinatorial nature of the mining methods. In fact, it is very difficult to control the size of the set of frequent patterns given a minimum threshold for instance. On the one hand, if the minimum threshold is very small, the set of returned patterns overwhelmed the end-user. On the other hand, if the minimum threshold is very large, the set of patterns risks being empty. To solve this problem, many approaches are proposed like Top-k pattern mining [2] which returns the k most frequent patterns, but with a lake of diversity. The recent proposal approach is based on output pattern sampling [3]. Output pattern sampling consists in generating a sample of patterns among the patterns that would have been extracted from the complete dataset. It is a non-exhaustive method to extract relevant patterns and ensures good interaction and great diversity while offering strong statistical guarantees thanks to its random nature. Its usefulness has been widely demonstrated in recent years in many areas like feature classification [4], outlier detection [5] or interactive discovery [6, 7, 8]. It has also been applied to several pattern languages such as graphs [3], itemsets [9, 4, 10] and sequences [11, 12].

However, at the time of writing this paper, none of the current sampling methods does mind managing the size in memory of the data or the reprocessing when the user changes the given utility. In the field of pattern mining, it is also very important to consider the size of the database which can be an obstacle when the data must be loaded into memory. Indeed, with the output sampling methods of the state-of-the-art, it is a necessity that the entire database is stored in memory except using a decentralized approach [10] where the data are natively decentralized in different sites. In our case, we assume that all transactions are available in a single and unique machine where the user needs to run a sampling algorithm. In some datasets with very large sizes, one can use a compact data structure in order to have a compressed representation of the database as it was already done by the exhaustive extraction of interesting patterns [13]. To the best of our knowledge, none of the output pattern sampling methods in

the literature [3, 4, 9, 11] has yet been applied to compact representations of databases. However, most of the proposed methods need to store the entire database in memory. It is also very important to facilitate reprocessing when a user decides to change a utility measure. So, an efficient method should not weight each transaction at each change. The processing phase is often consuming when the size of the database becomes large. In [14] the authors tackle the problem of utility change when they work with many utilities like frequency, area, or decay. But, the proposed solution also depends on the number of transactions in the database. Then, it has the same complexity of reprocessing as the Two-Step pattern sampling in [4]. So, besides the need of reducing the storage cost, this paper addresses techniques to speed up the reprocessing times for pattern sampling in large databases when the user changes the interestingness measure.

In this paper, we present a new method of output pattern sampling that is based on a compressed data structure. Our main objective is therefore to propose a generic output pattern sampling algorithm that extracts patterns proportionally to a length-based utility measure [10] from a trie of occurrences.

Our main contributions are as follows:

- We introduce a new structure called *trie of occurrences* then we propose TPSpace (Trie-based Pattern Space), an algorithm for its construction. It is a trie where each node has a set of weight information for the exact draw of a pattern. In our case, we weight each node according to the lengths of occurrences in the sub-trie of which it is the root.
- We propose TPSampling (Trie-based Pattern Sampling), a generic algorithm for sampling patterns from a *trie of occurrences* according to a probability proportional to a length-based utility measure. The genericity of TPSampling comes from the fact that it can consider any interestingness measure based on the length.
- We theoretically and experimentally evaluate the complexity of TPSampling. In particular, we show that TPSampling makes an exact draw according to a length-based utility measure chosen by the user and directly from the trie. In addition, we evaluate the complexity in memory storage of the trie of occurrences on different transactional databases including synthetic datasets and then we evaluate its speed according to different length-based utility measures.

The remainder of this paper is organized as follows: Section 2 situates our work in the state-of-the-art of pattern sampling methods and highlights the main motivations behind this paper. Section 3 begins with some basics to fully understand our approach to drawing patterns from a trie. It ends by highlighting the challenges that we need to solve to achieve our goal. Section 4 presents our first contribution on tries which consists in showing how to construct a trie of occurrences. Section 5 describes our generic algorithm for sampling patterns proportionally to a length-based utility measure from an enriched trie. Section 6 theoretically analyses our method by detailing the time of building a trie of occurrences and that of drawing a pattern by TPSampling. Section 7 presents the experimental results of our approach by comparing them with those of the two-step method algorithm [4] combining length-based utility measures [10]. Finally, Section 8 concludes this paper and gives some perspectives.

2. Related works

This section presents the related works in pattern sampling and data structures for pattern mining.

2.1. Pattern sampling techniques

Since the first proposition of pattern sampling method [3] in 2009, numerous algorithms are proposed for output pattern sampling [15, 4, 16, 17, 18, 19, 20, 21, 9, 22, 23, 11, 10, 24]. The types of these methods can be grouped into four main classes: *random work (MCMC)*, *SAT framework*, *multi-steps* and *reservoir sampling*. In the following paragraphs, we will briefly present each family of methods while discussing their efficiency in terms of storage in memory and their ability to rapidly change a utility measure.

Random walk sampling. The first class of methods uses random walks in the search space to sample interesting patterns. In this class, most of the methods [3, 16, 15, 18] are based on Markov Chain Monte Carlo (MCMC) methods. The idea is to build a Markov chain simulating a distribution of a probability law. In [3], the authors proposed the first method of this class to sample frequent sub-graphs in a graph database, and then [16] proposed another method for interactive discovery. One of the limits of these methods is the slowness of their convergence.
Efficiency in memory storage: It is important to note that all these algorithms run locally in a single machine. So, they fully benefit from the available memory by storing the entire data.
Flexibility on utility change: The methods of this class need to restart from scratch when the user changes the utility measure. Indeed, the partial order graph of the interesting sub-graphs must be readjusted to make new random walks.

SAT-based sampling. The second class of output sampling methods is based on the SAT logical formalism. It has been implemented for the output sampling of pattern set [9]. In this article, the authors have presented both a generic algorithm called GFlexic and a specialized algorithm called EFlexic. For this class of methods, the basic idea is to use constraint programming and logical solvers to perform pattern mining. On a practical level, GFlexic has only been tested on small transactional databases (at most 4,000 transactions and 300 items), which is explained by the very great genericity of the method. One of the limits of these methods is that they are not exact (the draw of a pattern is approximatively exact).
Efficiency in memory storage: Like the random walk methods, all these algorithms run locally in a single machine and need to use the available memory for storing the entire data.
Flexibility on utility change: These methods need also to restart from scratch when the user changes the utility measure. Indeed, they have to divide the space of the valuations to satisfy the newly obtained logical formula.

Multi-step sampling. The multi-step approaches [4, 17, 22, 11, 12, 10, 14] walk in two phases. The first phase is dedicated to the preprocessing which consists

to weight each instance (transaction or sequence) of the database proportionally to the sum of the interest of the distinct patterns it contains. The multi-step sampling methods presented in [4, 17, 22, 11, 12] deal with local database in a single machine. However, in [10, 14], a multi-step algorithm is designed for transactional distributed databases and was applied in DBpedia and Wikidata. This algorithm centralizes the length of the transactions and uses two primitives to construct the sampled pattern. It is then important to note that even if we readapted the proposed algorithm in [10, 14] for local data, it could in no way reduce the used size in memory.
Efficiency in memory storage: These methods also need to store the entire data in main memory except that in [10, 14] in the case of native distributed databases.
Flexibility on utility change: When the user changes the utility measure, the two-step methods do a reprocessing phase which consists of weighting each transaction of the database according to the new utility measure. In [14], the authors show that the reprocessing phase is experimentally fast because proportional to the number of transactions in the database.

Reservoir pattern sampling. Reservoir pattern sampling methods [24] are techniques to discover relevant patterns from streaming data. This approach consists in incrementally maintaining a data sample representative of the data stream benefiting from reservoir sampling [25, 26]. In [24], the authors propose a generic algorithm called ResPat which samples relevant itemsets from data streams according to the dumped support. They sow that ResPat can be very fast according to the dumping function because it does one pass over the data to sample a set of patterns.
Efficiency in memory storage: One of the most important challenges resolved by reservoir sampling methods is the memory storage issue. ResPat works with one transaction at a time, then the only one which is present in memory. So, the methods of this class are very parsimonious in memory storage cost.
Flexibility on utility change: These methods need also to restart from scratch when the user changes the utility measure because they do not store the data in memory.

2.2. Data structures for pattern mining

To solve the problem of pattern mining, many data structures have been proposed such as "FP-Tree"[13] or "Trie"[27]. In transactional databases, there are lots of repetitions because several transactions can contain the same information. These repetitions make sense in the field of pattern mining because they allow us to find interesting rules, but we have to know how to represent them. For example, in a transactional database, if 90% of transactions that contain the items $\{e_1, e_2, e_3, e_4\}$ also contain the items $\{e_5, e_6\}$ then we might as well group them together so that they share the same prefix. This considerably reduces the size of the database in memory. With "FP-Tree" or "Trie", it becomes possible to represent transactions containing the same item in one path. This is because they share the same prefix. But the difference between these two structures is that, unlike "trie" which only links a node with its children, "FP-Tree" establishes links between nodes of different branches to quickly compute the frequency of the patterns. Since we do not want to compute this latter, we suggest using "trie"[27] to have a compact representation of the database in memory. The

"Trie" data structure is widely used in the field of text mining [28]. Other authors have used the trie structure for frequent pattern mining with Apriori [29]. Through the previous works, we note that trie is much more parsimonious in storage cost on transactional databases than on text databases. This is because the depth of the tree depends exclusively on the size of the longest chain. In the case of text mining, characters can repeat themselves multiple times in the same string where they follow an order, which is not the case with transactions. These are considered as sets of items, and therefore no transaction contains duplicated items.

In this paper, we propose an original multi-step pattern sampling, the first approach based on compact structure. We will see that the goal of using the compact structure is not only for *memory problem* but for the reprocessing when the user changes the utility measure too: *the flexibility on utility change*.

3. Preliminaries and problem statement

In this section, we present first some necessary basic notions and definitions for the understanding of the subject. We end it with a formalization of the problem that we address in this paper.

3.1. Basic definitions

Let $\mathcal{I} = \{e_1, \cdots, e_N\}$ be a finite set of literals called items. We assume that there is an arbitrary total order $>_\mathcal{I}$ between items : $e_1 >_\mathcal{I} \cdots >_\mathcal{I} e_N$. An itemset (or pattern), denoted by $\varphi = \{e_{i_1}, \cdots, e_{i_n}\}$ (or simply $\varphi = e_{i_1} \cdots e_{i_n}$), with $n \leq N$, is a none empty subset of $\mathcal{I}$, $\varphi \subseteq \mathcal{I}$. The set of all patterns that we can generate from $\mathcal{I}$ is called the pattern language denoted by $\mathcal{L} = 2^{|\mathcal{I}|} \setminus \emptyset$. The length of a pattern $\varphi \in \mathcal{L}$ denoted by $|\varphi|$ is the number of items it contains (its cardinality). A transactional database $\mathcal{T}$ is a multi-set of itemsets (called transactions) where each of them has a unique identifier $j \in \mathbb{N}$. We denote by $t_j = e_1 \cdots e_n$ a transaction identified by j of length $|t| = n$ defined in $\mathcal{I}$, and $\mathcal{L}(\mathcal{T})$ the set of all patterns that appear in $\mathcal{T}$. For example, Table 1 is a transactional database built from the set of four items $\mathcal{I} = \{A, B, C, D\}$. In the rest of this paper, we use this dataset to give some illustrations.

Originally, the goal of a pattern sampling technique is to access the pattern space $\mathcal{L}$ by an efficient sampling procedure simulating a distribution π: $\mathcal{L} \to [0; 1]$ which is defined with respect to some interestingness measure $m : \pi(\cdot) = m(\cdot)/Z$ where Z is a normalization constant, the sum of the interest of all pattern $\varphi \in \mathcal{L}$ in the dataset $\mathcal{T}$, defined by $Z = \sum_{\varphi \in \mathcal{L}} m(\varphi, \mathcal{T})$. The sampling of k patterns of the language $\mathcal{L}$ according to a distribution proportional to an interestingness measure m in the database $\mathcal{T}$ can be formulated as follows:

$$\mathcal{S}amp\ell e_k(\mathcal{L}, \mathcal{T}, m) = \bigcup_{i=1}^{k} \{\varphi_i \sim m(\mathcal{L}, \mathcal{T})\}$$

where $\varphi \sim m(\mathcal{L}, \mathcal{T})$ means that the pattern φ is drawn with a probability proportional to its interest m. Formally, $\varphi \sim m(\mathcal{L}, \mathcal{T}) \Leftrightarrow \pi(\varphi) = m(\varphi, \mathcal{T})/Z$.

In another word, the main objective of the output sampling methods is to have a sample of patterns representative of the set of patterns that can be extracted

Table 1. A transactional database $\mathcal{T}$

tid	Itemsets
t_1	A B
t_2	A C
t_3	B C
t_4	A B C D

from the database. For example, if a pattern φ_1 has an interest twice as higher than that of a pattern φ_2 according to the interestingness measure chosen by the user, then φ_1 is twice as likely to be in the sample as the pattern φ_2. In the literature, there are several interestingness measures, but the most used of them is the frequency measure.

Definition 1 (Frequency of a pattern). Given a dataset $\mathcal{T}$ and its language $\mathcal{L}$. The frequency of the pattern $\varphi \in \mathcal{L}$ in $\mathcal{T}$ is defined as follows:

$$freq(\varphi, \mathcal{T}) = |\{t_i \in \mathcal{T} : \varphi \subseteq t_i\}|.$$

Let $m(\cdot) = freq(\cdot)$ for example, if φ_1 and φ_2 are two patterns of the language $\mathcal{L}$ having respective frequencies $freq(\varphi_1, \mathcal{T})$ and $freq(\varphi_2, \mathcal{T})$ such that $freq(\varphi_1, \mathcal{T}) = 2 \times freq(\varphi_2, \mathcal{T})$, then the probability to draw φ_1 according to the frequency is twice that of φ_2.

Example 1. Figure 1 gives all patterns defined in $\mathcal{T}$ and their corresponding frequency. The normalization constant Z is the sum of the pattern frequency: $Z = \sum_{\varphi \in \mathcal{L}(\mathcal{T})} freq(\varphi, \mathcal{T}) = 24$.
Within the dataset $\mathcal{T}$, we have $freq(AB, \mathcal{T}) = |\{t_1, t_4\}| = 2$ and $freq(AD, \mathcal{T}) = |\{t_4\}| = 1$. This means that $\pi(AB, \mathcal{L}(\mathcal{T})) = 2/24$ and $\pi(AD, \mathcal{L}(\mathcal{T})) = 1/24$. So, the probability to draw AB is twice that of AD in the dataset $\mathcal{T}$.

By definition, if there are in the language $\mathcal{L}$ at least two patterns having probabilities not null, then the operator $\mathcal{S}ample_k$, with $k > 0$, is non-deterministic. In other words, two draws with the same interestingness measure in the same database may not return the same k patterns. Let's also note that drawing $k = k_1 + k_2$ patterns following an interestingness measure m in the language $\mathcal{L}$ of a database $\mathcal{T}$ with $\mathcal{S}ample_k(\mathcal{L}, \mathcal{T}, m)$, comes down to draw k_1 patterns in $\mathcal{L}$ according to m with $\mathcal{S}ample_{k_1}(\mathcal{L}, \mathcal{D}, m)$, then k_2 patterns of $\mathcal{L}$ according to m with $\mathcal{S}ample_{k_2}(\mathcal{L}, \mathcal{D}, m)$. However, this does not mean that $\mathcal{S}ample_k(\mathcal{L}, \mathcal{D}, m)$ and $\mathcal{S}ample_{k_1}(\mathcal{L}, \mathcal{D}, m) \cup \mathcal{S}ample_{k_2}(\mathcal{L}, \mathcal{D}, m)$ return the same sample of patterns.

It is also common to give a utility to an itemset and to combine the frequency of an itemset with its utility. In this paper, we deal with length-based utility measures [10].

Definition 2 (Length-based utility measures). A utility u defined from $\mathcal{L}$ to $\mathbb{R}$ is called a length-based utility if there exists a function f_u from $\mathbb{N}$ to $\mathbb{R}$ such that $u(\varphi) = f_u(|\varphi|)$ for each $\varphi \in \mathcal{L}$. Given the set $\mathcal{U}$ of length-based utilities, $\mathcal{M}$ is the set of interestingness measures m_u such that for every pattern φ and database $\mathcal{T}$, $m_u(\varphi, \mathcal{T}) = freq(\varphi, \mathcal{T}) \times u(\varphi)$ with $u \in \mathcal{U}$.

For example, with the frequency, the utility function $u_{freq}(\varphi) = 1$ for all pattern in $\mathcal{L}$. If we consider the utility function $u_{area}(\varphi) = |\varphi|$, we obtain the area

φ	A	B	C	D	AB	AC	AD	BC
freq	3	3	3	1	2	2	1	2
m_{area}	3×1	3×1	3×1	1×1	2×2	2×2	1×2	2×2
m_{decay}	0.3	0.3	0.3	0.1	0.02	0.02	0.01	0.02

φ	BD	CD	ABC	ABD	ACD	BCD	ABCD
freq	1	1	1	1	1	1	1
m_{area}	1×2	1×2	1×3	1×3	1×3	1×3	1×4
m_{decay}	1×0.1^2	1×0.1^2	1×0.1^3	1×0.1^3	1×0.1^3	1×0.1^3	1×0.1^4

Fig. 1. Pattern language of $\mathcal{T}$: $\mathcal{L}(\mathcal{T})$, $\alpha = 0.1$ for m_{decay}

$\mathcal{T}$

tid	Itemsets
1	A B
2	A C
3	B C
4	A B C D

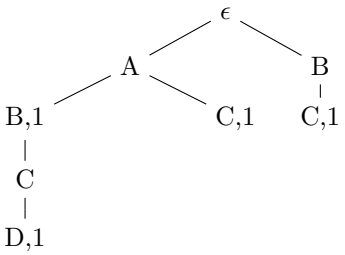

Fig. 2. Representation of a database $\mathcal{T}$ as a trie

measure: $freq(\varphi, \mathcal{T}) \times |\varphi|$. If $u_{decay}(\varphi) = \alpha^{|\varphi|}$, we get an exponential decay in $\alpha \in]0, 1]$. More generally, we consider the class of interestingness measures of the form $freq(\varphi, \mathcal{T}) \times u(\varphi)$ where u exclusively depends on the length of itemsets [10]. It is also possible to combine many utility functions to form another one.

3.2. Key ideas, challenges and problem statement

We first focus on some interesting key ideas and challenges to situate our work before reformulating the questions we should properly answer in order to achieve our goal. Finally, we reformulate the problem that we want to solve in this paper.

Key idea. As we have pointed out, drawing a pattern is one of the most important steps in pattern sampling, especially in the case of user-centered mining. In this paper, we propose to use a trie structure to build the pattern space while guaranteeing good flexibility on reprocessing in utility change.

Definition 3 (Trie [27]). A trie is a data structure in the form of a rooted tree such that for any node, its descendants have the common prefix.

Example 2. We represent the database $\mathcal{T}$ as a trie in Figure 2. To do this, each node of the trie is associated with a value which is the number of transactions of which it is the last element according to the order of insertion of the items, the number of times its label terminates a transaction of the dataset.

It is important to note that many representations of the database $\mathcal{T}$ as a trie are possible according to the insertion order of the items (decreasing order of their frequencies, ascending order, lexicographic order, etc.).

Challenges in trie-based pattern sampling. Sampling pattern in trie structure is far from easy. In addition to guaranteeing an exact draw of a pattern (with a probability proportional to its weight), we need also to cleverly consider to new length-based utility to avoid time-consuming reprocessing. In that case, the main challenges that we need to solve are as follows:

– efficiently build and weight the trie that corresponds to the database,
– draw a pattern directly from the trie according to a probability proportional to its interest in the database.

Problem statement. These main challenges can be finally solved by answering the following questions that we formulate here.
Let $\mathcal{T}$ be a transactional database, $u, u' \in \mathcal{U}$ two length-based utilities, and μ and M two positive integers such that $\mu < M$.

1. What additional information should be added to a (classical) trie to be able to directly sample patterns without needing to use the underlying transactional database?
2. How to draw a pattern φ from the language $\mathcal{L}_{[\mu..M]}$ proportionally to $m_u(\varphi, \mathcal{T})$ directly from the weighted trie?
3. How to compute $\mathcal{S}ampl\ell e_{k_1}(\mathcal{L}_{[\mu..M]}, \mathcal{T}, m_u)$ and $\mathcal{S}ampl\ell e_{k_2}(\mathcal{L}_{[\mu..M]}, \mathcal{T}, m_{u'})$, with $0 < k_1 \leq k_2$, without rebuilding the trie of occurrences?

Now, we are going to solve carefully question (1) in Section 4, and questions (2) and (3) in Section 5.

4. TPSpace: Trie-based Pattern Space

According to Figure 2, two occurrences of the same pattern may occur in different paths of a trie, for example AC within the path $A \to B \to C$ and AC within the path $A \to C$. But they can also be merged, in the case of occurrences of the pattern AB (appearing in transactions of identifiers 1 and 4) which are represented in a single path of the trie. According to this analysis, the approach that we will present in this paper is based on two fundamental notions which are *pattern occurrence* (or occurrence for short) and *language of occurrences*.

In the rest of this section, we present first the necessary elements to define a trie of occurrences. Then, we present the algorithm named TPSpace which designs a weighted trie of occurrences. We end it with an illustrative example for building a trie of occurrences.

Definition 4 (Occurrence and language fo occurrences). Let $\mathcal{T}$ be a transactional database and $\mathcal{L}_{[\mu..M]}$ its pattern language under minimum μ and maximum M length constraints. If there is a transaction t of identifier i in $\mathcal{T}$ that contains the pattern $\varphi \in \mathcal{L}_{[\mu..M]}$, then we denote by φ_i the occurrence of φ in the transaction t_i. The set of occurrences in $\mathcal{T}$ under length constraint μ and M forms a language of occurrences denoted by $\mathcal{L}^o_{[\mu..M]}$. Formally, $\mathcal{L}^o_{[\mu..M]} = \{\varphi_i : (\exists(\varphi, t_i) \in \mathcal{L}_{[\mu..M]} \times \mathcal{T})(\varphi \subseteq t_i)\}$. The length of an occurrence φ_i of a pattern $\varphi \in \mathcal{L}$ is equal to the length of φ.

Note that $\mathcal{L}^o$ is a set of occurrences while $\mathcal{L}$ is a set of patterns, and, unlike a pattern, an occurrence belongs to one and only one transaction. The frequency

of a pattern in $\mathcal{L}$ is the cardinality of the set of its occurrences in $\mathcal{L}^o$. In other words, it is the number of transactions in $\mathcal{T}$ containing one of its occurrences.

Example 3. If we consider the patterns of length 2, then we note that AB_1 and AB_4 are 2 occurrences of the pattern AB in the database $\mathcal{T}$ because AB is contained in transactions of identifier 1 and 4. The frequency of AB is therefore equal to 2. Similarly, AC_2 and AC_4 are 2 occurrences of the pattern AC. BC_3 and BC_4 are 2 occurrences of BC. Patterns AD, BD, and CD have single occurrences in transaction t_4. In this example, to draw a pattern of length 2 proportionally to its frequency in $\mathcal{T}$, it suffices to uniformly draw an occurrence in the set $\mathcal{L}^o_{[2..2]}(\mathcal{T}) = \{AB_1, AB_4, AC_2, AC_4, BC_3, BC_4, AD_4, BD_4, CD_4\}$. Thus, the probability of drawing the pattern AB in the set $\mathcal{S}$ is equal to $\pi(AB, \mathcal{L}_{[2..2]}(\mathcal{T})) = \pi(AB_1, \mathcal{L}^o_{[2..2]}(\mathcal{T})) + \pi(AB_4, \mathcal{L}^o_{[2..2]}(\mathcal{T})) = \frac{2}{9}$.

Property 1. Let $\mathcal{T}$ be a transactional database, $\mathcal{L}$ and $\mathcal{L}^o$ the pattern language and occurrence language of $\mathcal{T}$ respectively, and ℓ a positive integer. The probability to draw a pattern φ from $\mathcal{L}$ is computed from the occurrence language as follows:

$$\pi(\varphi, \mathcal{L}_{[\ell..\ell]}(\mathcal{T})) = \sum_i \pi(\varphi_i, \mathcal{L}^o_{[\ell..\ell]}(\mathcal{T})).$$

By convention, $\pi(\varphi_i, \mathcal{L}^o_{[\ell..\ell]}(\mathcal{T})) = 0$ if $\varphi_i \notin \mathcal{L}^o_{[\ell..\ell]}(\mathcal{T})$.

Proof. The proof of Property 1 is trivial and stem from the fact that $\mathcal{L}^o_{[\mu..M]} = \{\varphi_i : (\exists(\varphi, t_i) \in \mathcal{L}_{[\mu..M]} \times \mathcal{T})(\varphi \subseteq t_i)\}$ by definition. □

From Property 1, we know how to draw a pattern of length ℓ proportionally to its frequency among the set of patterns of the same length using a uniform drawing of occurrences. In this case, if we know how to draw a length ℓ proportionally to the sum of the utilities of patterns of length ℓ, so we know how to sample a pattern proportionally to its interest in the database.

4.1. Definition of a trie of occurrences

In Figure 2, we can obtain another trie that represents the data by starting with the reverse order for instance. This shows that a trie of a transactional dataset changes shape according to the total order relation considered on the items. Thus, we start by defining the total order relations used in this paper before talking about the identifier and the content of a node.

Definition 5 (Total order relation between items). Let $\mathcal{I}$ be a set of items or literals on which the transactional dataset $\mathcal{T}$ is defined.

–A total order relation on literals is said a lexicographic-based order and denoted by $>^{lexico}_{\mathcal{I}}$, if it orders the literals of $\mathcal{I}$ according to lexicographic order.
–A total order relation on literals is said a frequency-based order and denoted by $>^{freq}_{\mathcal{I}}$ if it orders the elements of $\mathcal{I}$ according to the descending order of their frequencies in $\mathcal{T}$ and according to the lexicographic order in case of equal frequency.

In the following we denote $>_{\mathcal{I}}$ a total order relation between the items of a

dataset. Note that there are other types of total order relations in the literature that can be applied to literals. However, we will limit ourselves to the two total order relations previously defined as examples, even if our approach works with any total order relation on literals.

Example 4. Considering the dataset $\mathcal{T}$, we have $freq(A, \mathcal{T}) = 3$, $freq(B, \mathcal{T}) = 3$, $freq(C, \mathcal{T}) = 3$ and $freq(D, \mathcal{T}) = 1$. In this case, we can already say that $A >_{\mathcal{I}}^{freq} D$, $B >_{\mathcal{I}}^{freq} D$ and $C >_{\mathcal{I}}^{freq} D$ because $freq(A, \mathcal{T}) = freq(B, \mathcal{T}) = freq(C, \mathcal{T}) > freq(D, \mathcal{T})$. Now, if we consider the lexicographic order between the items, we have $A >_{\mathcal{I}}^{lexico} B >_{\mathcal{I}}^{lexico} C$. So, continuing with the order relation $>_{\mathcal{I}}^{freq}$, we have $A >_{\mathcal{I}}^{freq} B >_{\mathcal{I}}^{freq} C >_{\mathcal{I}}^{freq} D$.

We are now going to define the notion of an identifier of a node in a trie. It is a concept that will allow us to enrich the trie of occurrences from a transactional dataset.

Definition 6 (Node identifier). Given a set of items $\mathcal{I} = \{e_1, \cdots, e_n\}$ and a symbol $\epsilon \notin \mathcal{I}$, a trie τ defined on $\mathcal{I}$ is a tree where every node $\eta \in \tau$ except the root contains a label denoted by $\eta.label$ belonging to $\mathcal{I}$, $\eta.label \in \mathcal{I}$, and where the root r of the trie contains the label ϵ, $r.label = \epsilon$. Thus, any node $\eta \in \tau$ can be identified by the sequence of node labels on the path from the root of τ to the node η. If $P = \epsilon e_{i_1} \dots e_{i_k}$ is this sequence, we write it down more simply $P = e_{i_1} \dots e_{i_k}$ for a node that is not the root, and we write $\widetilde{P} = e_{i_k}$ the label of the identified node η, $\widetilde{P} = \eta.label$.

Example 5. Considering the trie of the dataset $\mathcal{T}$ in Figure 2, the node containing the label B resulting from the transaction t_4 is identified by $P_1 = AB$. This same node represents the item B coming from t_1. Besides, the item B of the transaction t_3 is represented by the identifier node $P_2 = B$.

In the following, we will often use the concept of sub-trie of a trie defined below:

Definition 7 (Sub-trie). Let τ be a trie and P the identifier of a node. We denote by τ_P the sub-trie of τ whose root is the node identifier P.

We now define the concatenation operator $\circ$ as follows:

Definition 8 (Concatenation operator $\circ$). Let φ and φ' be two itemsets defined in $\mathcal{I}$ and ordered according to the total order relation $>_{\mathcal{I}}$, $\varphi \circ \varphi' = \varphi \cup \{e' \in \varphi' : (\forall e \in \varphi)(e >_{\mathcal{I}} e')\}$.

If $>_{\mathcal{I}}$ is the lexicographic order, then we have $B \circ AC = BC$ and $A \circ BC = ABC$. This concatenation operator allows us to define the notion of prefix that we are going to use in the definition of a projected database.

Definition 9 (Prefix). Let t be a transaction defined in $\mathcal{I}$ and P an itemset defined in $\mathcal{I}$ and ordered according to the total order relation $>_{\mathcal{I}}$. P is a prefix of transaction t if there is an itemset $\varphi \subseteq \mathcal{I}$ such as $t = P \circ \varphi$.

In the dataset $\mathcal{T}$ in Figure 2, $P = AB$ is a prefix of the transaction $t_4 = ABCD$ but P is not a prefix of the transaction $t_3 = BC$.

According to Definition 9, transactions with a common prefix can be grouped. To identify where the occurrences of a pattern are represented in a trie, i.e. in

which sub-trie. We are now going to introduce the notion of a projected database. Our definition is an adaptation of the notion introduced by [30].

Definition 10 (Projected database). Let $>_{\mathcal{I}}$ be a total order relation on all the items $\mathcal{I}$, $\mathcal{T}$ a transactional database, and P the identifier of a node. A projected database of $\mathcal{T}$ on P, denoted by $\mathcal{T}_P$, is a transactional database which contains for any transaction t of $\mathcal{T}$ with prefix P a copy of this transaction without the items of t preceding $\widetilde{P}$. Formally, the projected database of $\mathcal{T}$ on P is defined by:

$\mathcal{T}_P = \{(i, \widetilde{P} \circ \varphi) \in \mathbb{N} \times \mathcal{L} : (i,t) \in \mathcal{T} \wedge t = P \circ \varphi\}$

Example 6. Let us consider the trie of the transactional database $\mathcal{T}$ in Fig. 2. Occurrences in the projected database $\mathcal{T}_{AB}$ are the occurrences stored in the sub-trie whose root is identified by the prefix AB: B_1, B_4, BC_4, BD_4, CD_4, BCD_4. Occurrences in the projected database $\mathcal{T}_{AC}$ are the occurrences stored in the sub-trie whose root is identified by the prefix AC: C_2.

Now, we will define precisely which are the pattern occurrences represented at the level of the projected database of $\mathcal{T}$ on the prefix P, by partitioning these sets of occurrences by their length.

Definition 11 (Computing weights Φ_ℓ^- and Φ_ℓ^+). Let $\mathcal{T}$ be a transactional database and P a prefix. The set of occurrences of the projected database on the prefix P and having a length equal to ℓ is defined by:

$$\phi_\ell(P, \mathcal{T}) = \{(i, \varphi) \in \mathbb{N} \times \mathcal{L} : (i, \varphi') \in \mathcal{T}_P \wedge \varphi \subseteq \varphi' \wedge |\varphi| = \ell\}$$

$\Phi_\ell(P, \mathcal{T})$ denotes the total number of occurrences of length equal to ℓ in the projected database $\mathcal{T}_P$: $\Phi_\ell(P, \mathcal{T}) = |\phi_\ell(P, \mathcal{T})|$. The set of occurrences $\phi_\ell(P, \mathcal{T})$ can be split into two parts:

– The set of occurrences of the database projected on the prefix P, having a length equals to ℓ and **containing** the item $\widetilde{P}$ is defined by: $\phi_\ell^+(P, \mathcal{T}) = \{(i, \varphi) \in \phi_\ell(P, \mathcal{T}) : \widetilde{P} \in \varphi\}$. Its cardinality is denoted by $\Phi_\ell^+(P, \mathcal{T}) = |\phi_\ell^+(P, \mathcal{T})|$.
– The set of occurrences of the database projected on the prefix P, having a length equals to ℓ and **not containing** the item $\widetilde{P}$ is defined by: $\phi_\ell^-(P, \mathcal{T}) = \{(i, \varphi) \in \phi_\ell(P, \mathcal{T}) : \widetilde{P} \notin \varphi\}$. Its cardinality is denoted by $\Phi_\ell^-(P, \mathcal{T}) = |\phi_\ell^-(P, \mathcal{T})|$.

As $\widetilde{P}$ is not defined if P is the empty prefix, we consider by convention that $\phi_\ell^-(\epsilon, \mathcal{T}) = \phi_\ell(\epsilon, \mathcal{T})$, while $\phi_\ell^+(\epsilon, \mathcal{T}) = \emptyset$.

Example 7. Let us consider the projected database $\mathcal{T}_A = \{(1, AB), (2, AC), (4, ABCD)\}$, we have $\phi_2^+(A, \mathcal{T}) = \{(1, AB), (2, AC), (4, AB), (4, AC), (4, AD)\}$ and $\phi_2^-(A, \mathcal{T}) = \{(4, BC), (4, BD), (4, CD)\}$ on the other hand. Then $\Phi_2^+(A, \mathcal{T}) = 5$ and $\Phi_2^-(A, \mathcal{T}) = 3$.

Subsequently, at the level of the node identified by P of a trie, we will store as weight the cardinalities of these sets, distinguishing the subsets of occurrences of length ℓ containing or not the item $\widetilde{P}$.

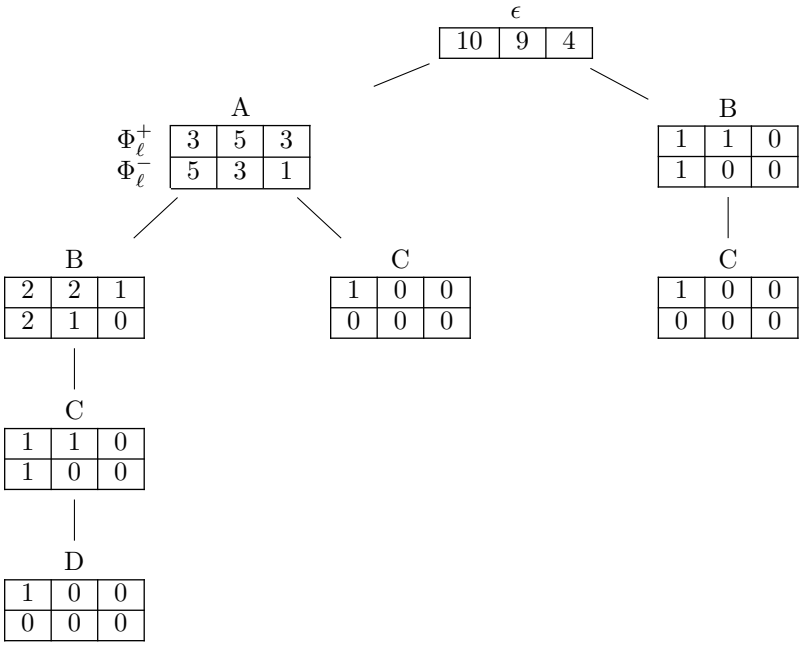

Fig. 3. Enriched trie of the trie provided by Figure 2

After having introduced the necessary notions and notations for the enrichment of a classic trie to a trie of occurrences, we will explicitly define the latter. In the remainder of this chapter, the notation τ denotes a trie of occurrences.

Definition 12 (trie of pattern occurrences). Given a transactional database, a trie of occurrences for $\mathcal{T}$, denoted by τ, is a tree where each node $\eta \in \tau$ contains:

–a label denoted by $\eta.label$ belonging to $\mathcal{I} \cup \{\epsilon\}$, ϵ being the label reserved for the root.

–a list of children denoted by $\eta.child$. Subsequently, we denote $|\eta.child|$ the number of children of node η and $\eta.child[i]$ the i-th child of η for $i \in [1..k]$ with $k = |\eta.child|$.

–an array of positive weights denoted by $\eta.\Phi^+$. If P is the identifier of the node η in τ, then $\eta.\Phi^+[\ell] = \Phi^+_\ell(P, \mathcal{T})$ for $\ell \in [\mu..M]$, μ and M being the minimum and maximum length constraints considered during the construction of the trie.

–an array of negative weights denoted by $\eta.\Phi^-$. If P is the identifier of the node η in τ, then $\eta.\Phi^-[\ell] = \Phi^-_\ell(P, \mathcal{T})$ for $\ell \in [\mu..M]$, μ and M being the minimum and maximum length constraints considered during the construction of the trie.

Example 8. Let us consider the transactional database of Figure 2, the minimum $\mu = 1$ and maximum $M = 3$ length constraints, we build the enriched trie according to the total order relation $>_\mathcal{I}$ in Figure 3.

In this example, the set of labels for the children of the root r is $\{A, B\}$. The

number of patterns of length $\ell = 2$ in the trie τ is equal to $r.\Phi^-[2] = 9$. Let η be the identifier node $P = \epsilon A$. Then we have $\eta.\Phi^+[2] = 5$ and $\eta.\Phi^-[2] = 3$ to say that the sub-trie τ_A contains 5 pattern occurrences of length 2 with the item $\widetilde{P} = A$ (AD_4, AC_4, AB_4, AB_1, AC_2) and 3 occurrences of length 2 without the item $\widetilde{P} = A$ (CD_4, BD_4, BC_4).

4.2. TPSpace: Algorithm for building a trie of occurrences

Now, we show how to build a trie of occurrences from a transactional database to efficiently take into account any type of length-based utility measure. Note first that this construction is done iteratively by adding the transactions one by one to the trie. In our case, we need to compute the positive and negative contributions of each transaction t within each node of identifier P such that P is a prefix of t.

Property 2. Let $\mathcal{T} = \{t_1, \cdots, t_n\}$ be a transactional database. We denote by $\mathcal{T}_i$ the subset of transactions defined by $\mathcal{T}_i = \{t_k \in \mathcal{T} : 1 \leq k \leq i\}$. If P is a prefix of t_i, then we have:

$$\Phi^\star_\ell(P, \mathcal{T}_i) = \Phi^\star_\ell(P, \mathcal{T}_{i-1}) + \binom{|t_i| - |P|}{\ell - \epsilon}, \epsilon = \begin{cases} 1 & \text{if } \star = + \\ 0 & \text{if } \star = - \end{cases}$$

By convention, $\Phi^-_\ell(P, \mathcal{T}_0) = 0$ whatever the identifier P. If P is the root identifier, $P = \epsilon$, then $|P| = 0$ and in this case, we set that $\Phi^+_\ell(P, \mathcal{T}) = 0$.

Proof. From Definition 11, we know that on the one hand $\Phi^+_\ell(P, \mathcal{T}_i) = \Phi^+_\ell(P, \mathcal{T}_{i-1}) + \Phi^+_\ell(P, \{t_i\})$ and on the other hand $\Phi^-_\ell(P, \mathcal{T}_i) = \Phi^-_\ell(P, \mathcal{T}_{i-1}) + \Phi^-_\ell(P, \{t_i\})$. If P is a prefix of t_i, then according to Definition 9, there is an itemset φ' such that $t_i = P \circ \varphi'$. So, $\Phi^+_\ell(P, \{t_i\}) = |\{\widetilde{P} \circ \varphi : \varphi \subseteq \varphi' \wedge |\varphi| = \ell - 1\}|$. Then we have $\Phi^+_\ell(P, \{t_i\}) = \binom{|t_i|-|P|}{\ell-1}$. On the other hand, $\Phi^-_\ell(P, \{t_i\}) = |\{\varphi \subseteq \varphi' : |\varphi| = \ell\}|$. Then we also have $\Phi^-_\ell(P, \{t_i\}) = \binom{|t_i|-|P|}{\ell}$. Hence the result. □

When adding the transaction t_i in the trie τ, the terms $\binom{|t_i|-|P|}{\ell-1}$ and $\binom{|t_i|-|P|}{\ell}$ are called respectively the positive and the negative contribution of t_i to the occurrences of length ℓ of the node identified by the prefix P.

Example 9. Let us continue with Example 7 by computing the weights $\Phi^+_2(A, \mathcal{T})$ and $\Phi^-_2(A, \mathcal{T})$ but this time using Property 2. By definition, we have $\mathcal{T}_A = \{(1, AB), (2, AC), (4, ABCD)\}$.
According to Property 2, we have: $\Phi^+_2(A, \mathcal{T}_1) = 0 + \binom{|t_1|-1}{2-1} = \binom{2-1}{1} = \binom{1}{1} = 1$. Then, by adding t_2 we have $\Phi^+_2(A, \mathcal{T}_2) = \Phi^+_2(A, \mathcal{T}_1) + \binom{|t_2|-1}{2-1} = 1 + \binom{2-1}{1} = 1 + \binom{1}{1} = 1 + 1 = 2$. Adding transaction t_3 does not affect the weights of the identifier node $P = A$ because, in this case, P is not a prefix of t_3. So we have $\Phi^+_2(A, \mathcal{T}_3) = \Phi^+_2(A, \mathcal{T}_2) = 2$. Finally, by adding transaction t_4, we have $\Phi^+_2(A, \mathcal{T}_4) = \Phi^+_2(A, \mathcal{T}_3) + \binom{|t_4|-1}{2-1} = 2 + \binom{4-1}{1} = 2 + \binom{3}{1} = 2 + 3 = 5$.

We also have $\Phi^-_2(A, \mathcal{T}_1) = 0 + \binom{|t_1|-1}{2} = \binom{2-1}{2} = \binom{1}{2} = 0$. With adding transaction t_2 we have $\Phi^-_2(A, \mathcal{T}_2) = \Phi^-_2(A, \mathcal{T}_1) + \binom{|t_2|-1}{2} = 0 + \binom{2-1}{2} = \binom{1}{2} =$

0. Likewise, adding transaction t_3 does not affect the weights of the identifier node $P = A$. So we have $\Phi_2^-(A, \mathcal{T}_3) = \Phi_2^-(A, \mathcal{T}_2) = 0$. Finally, by adding the transaction t_4, we have $\Phi_2^-(A, \mathcal{T}_4) = \Phi_2^-(A, \mathcal{T}_3) + \binom{|t_4|-1}{2} = 0 + \binom{4-1}{2} = \binom{3}{2} = 3$.

We need now to introduce some basic functions for creating, adding, or finding a node when inserting the items of a transaction into a trie.

- Let *CreateNode* the function defined by $\eta \leftarrow CreateNode(e)$ where η is a node such as $\eta.label = e$, $\eta.child = \emptyset$ where $\emptyset$ represents here an empty list of nodes and $\eta.\Phi^+[\ell] = \eta.\Phi^-[\ell] = 0$ for $\ell \in [\mu..M]$.
- Let *SearchChild* the function defined by $\eta.child[i] \leftarrow SearchChild(e, \eta)$ if there is i such as $\eta.child[i].label = e$, *null* otherwise.
- Let *AddChild* be the function allowing to add a child to a node. More precisely, if η is a node such as $k = |\eta.child|$, we will consider that after execution of $AddChild(c, \eta)$, we have $|\eta.son| = k + 1$ and $\eta.child[k+1] = c$.

In the following, $t[j]$, with $j > 0$, is the j^{th} item of transaction t according the total order relation $>_{\mathcal{I}}$.

Algorithm 1 TPSpace

```
1:  Input: A transactional database 𝒯, the minimum μ and maximum M length constraints
2:  Output: A trie of occurrences τ
3:  τ ← CreateNode(ϵ)                                  ▷ Creation of the trie root node
4:  for t ∈ 𝒯 do
5:      for ℓ ← μ to M do
6:          τ.Φ⁻[ℓ] ← τ.Φ⁻[ℓ] + C(|t|, ℓ)
7:      η ← τ
8:      for j ← 1 to |t| do
9:          c ← SearchChild(t[j], η)
10:         if c = null then                           ▷ If c is not child of node η
11:             c ← CreateNode(t[j])                   ▷ so we create it
12:             AddChild(c, η)
13:         for ℓ ← μ to M do
14:             c.Φ⁺[ℓ] ← c.Φ⁺[ℓ] + C(|t|−j, ℓ−1)
15:             c.Φ⁻[ℓ] ← c.Φ⁻[ℓ] + C(|t|−j, ℓ)
16:         η ← c
17: return τ
```

Algorithm 1 describes the TPSpace method to create a trie of occurrences of an input database $\mathcal{T}$ according to a total order relation $>_{\mathcal{I}}$. We initialize the trie of occurrences (line 3) by creating an empty node with the *CreateNode* function. For each transaction t of the input database whose items follow the order relation $>_{\mathcal{I}}$, we start at the root then, using Property 2, we compute its total contribution in the trie according to the lengths which we add to the root (line 6). Then, for each item $t[j]$ of the transaction being inserted in the trie, if there is not a child node c labelled with the item $t[j]$ according to the *SearchChild* function (line 9), we create it using the function *CreateNode* (line 11) then we add it among the children of η with the function *AddChild* (line 12). Finally, we add the positive

and negative contributions of the transaction t to the node c (lines 13 to 15) using Property 2. We now go to node c (line 16) and the process starts again with the item at position $j+1$ in t. Finally, line 17 returns the trie of occurrences τ for the transactional database $\mathcal{T}$.

Example 10. In this example, we show how to build a trie of occurrences with Algorithm 1. Let's consider the total order relation $>_{\mathcal{I}}^{freq}$ between the items, let us construct the trie of occurrences corresponding to the transactional database $\mathcal{T}$ given in Figure 2.

First, we compute the frequency of each item, and then we fix the order of the items. By scanning the database $\mathcal{T}$, we have $A >_{\mathcal{I}}^{freq} B >_{\mathcal{I}}^{freq} C >_{\mathcal{I}}^{freq} D$ according to the example 4. All the items of each transaction in the database must follow this order. Note that this order is important since each path of the trie also follows it.

Assuming now that we want to sample patterns of length between $\mu = 1$ and $M = 3$, we start the construction of the trie by creating the root with the function $CreateNode(\epsilon)$. We do not mention the exhaustive list of computations of the weights, but Figure 4 gives all intermediate results for the construction of the trie of occurrences τ corresponding to the database $\mathcal{T}$. Now, we show step by step how to insert transactions into the trie by computing the weight of each node each time an occurrence is counted in it.

1. $insert(t_1)$: We compute the weights of the root r: $r.\Phi^-[\ell] \leftarrow \binom{|AB|}{\ell}$, with $\ell \in [1..3]$ (lines 5 and 6). It gives the following results $r.\Phi^-[1] \leftarrow 2$, $r.\Phi^-[2] \leftarrow 1$ and $r.\Phi^-[3] \leftarrow 0$. The insertion of the transaction $t_1 = AB$, according to the total order relation $>_{\mathcal{I}}^{freq}$, adds to nodes in the trie. Since the root has no children yet, then the function $SearchChild(A, r)$ returns "null" (line 9). So that, just after the root, a node with the label A is created using the function $CreateNode$, $c \leftarrow CreateNode(A)$ (line 11) is added as first node of the root thanks to the function $AddChild(c, r)$ (line 12). In this node, we have $c.\Phi^+[1] \leftarrow \binom{|AB|-1}{0} = 1$, $c.\Phi^-[1] \leftarrow \binom{|AB|-1}{1} = 1$, $c.\Phi^+[2] \leftarrow \binom{|AB|-1}{2-1} = 1$ and the other weights are zero (lines 13 and 15). We go to the node c by pointing the variable η there, $\eta \leftarrow c$ (line 16). The node η has no child carrying the next item of the transaction because $SearchChild(B, \eta) = null$, then we create node $c \leftarrow CreateNode(B)$ (line 11) then we add it to the children of η with the function $AddChild(c, \eta)$ (line 12). In this node, we have $c.\Phi^+[1] \leftarrow \binom{|AB|-2}{1-1} = 1$ and the other weights are zero.
2. $insert(t_2)$: We add the number of occurrences of length equals to $\ell \in [1..3]$ to the weights of the root of τ (lines 5 and 6) : $r.\Phi^-[1] \leftarrow r.\Phi^-[1] + \binom{|AC|}{1} = 4$, $r.\Phi^-[2] \leftarrow r.\Phi^-[2] + \binom{|AC|}{2} = 2$ and $r.\Phi^-[3] \leftarrow r.\Phi^-[3] + \binom{|AC|}{3} = 0$. Then, we have $SearchChild(A, r) \neq null$ (line 9), so the weights of the node c are updated as follow: $c.\Phi^+[1] \leftarrow c.\Phi^+[1] + \binom{|AC|-1}{1-1} = 1 + 1 = 2$ et $c.\Phi^-[1] \leftarrow c.\Phi^-[1] + \binom{|AC|-1}{1} = 1 + 1 = 2$ (lines 13 and 15). For the next and last item of t_2, we have $\eta \leftarrow c$ (line 16) and $SearchChild(C, \eta) = null$ (line 9), then we create the child of the node η with the label C, $c \leftarrow CreateNode(C)$ (line 11). We add the node c to the children of η, $AddChild(c, \eta)$ (line 12), and we compute its weights (lines 13 to 15).
3. $insert(t_3)$: After adding the number of occurrences of length $\ell \in [1..3]$ of the

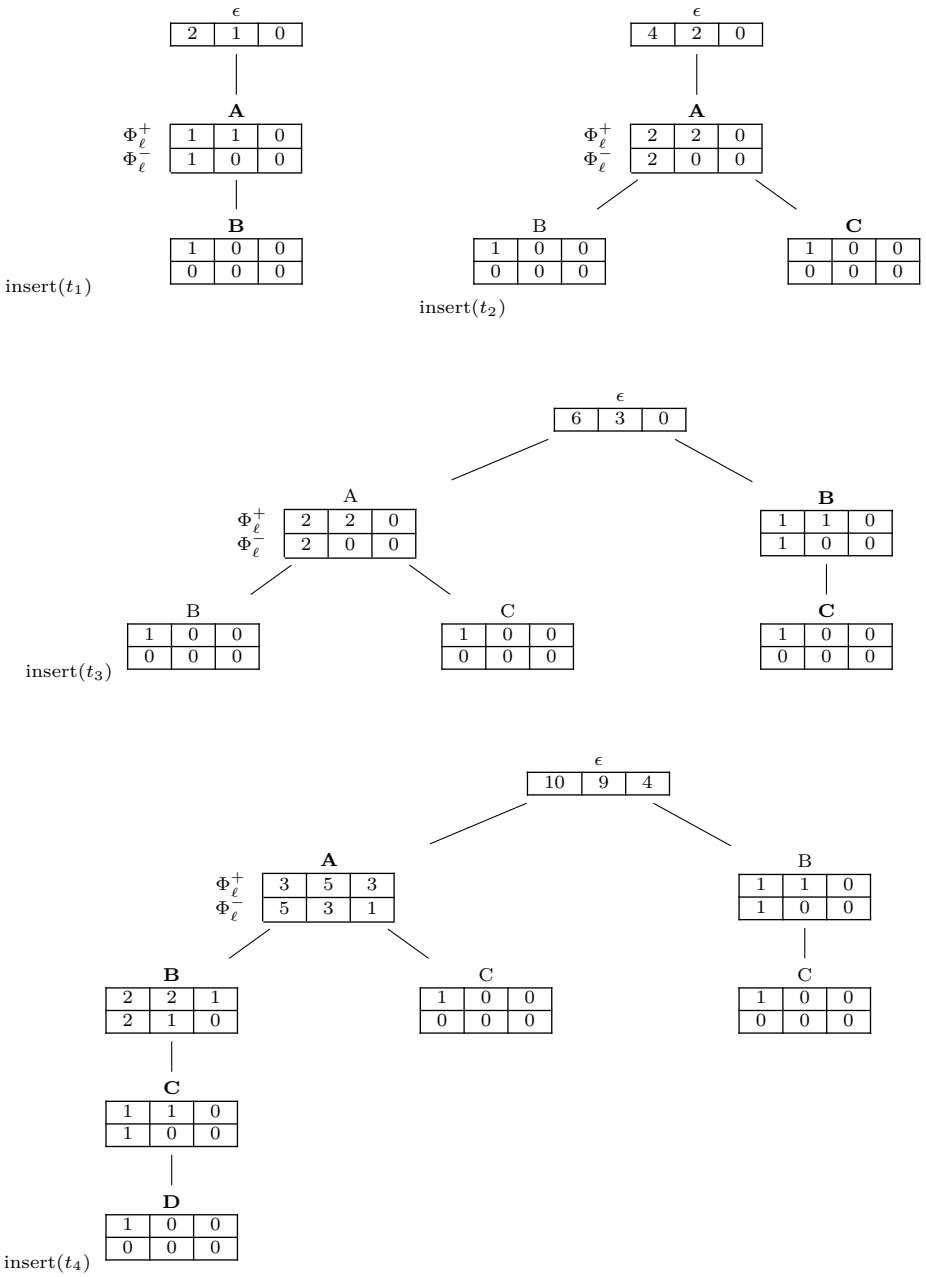

Fig. 4. Steps of building the trie of occurrence corresponding to $\mathcal{T}$ with $\ell \in [1..3]$

transaction t_3 to the weights of the root of τ (lines 5 and 6), we note that there is no child of the root having the label B because $SearchChild(B, r) = null$ (line 9). Now we create the node $c = CreateNode(B)$ (line 11) and we add it to the children of the root $AddChild(c, r)$ (line 12). Finally, we compute the weights of c (lines 13 to 15). Let $\eta \leftarrow c$ (line 16), we have $SearchChild(C, \eta) = null$ (line 9), then we create the node labelled with the item C, $c \leftarrow CreateNode(C)$ (line 11), and we add it to the children of η with the function $AddChild(c, \eta)$ (line 12). Finally, we compute the weights of the node c ($c.\Phi^+[1] \leftarrow 1$ and the other weights are 0) (lines 13 to 15).

4. $insert(t_4)$: After adding the number of occurrences of length $\ell \in [1..3]$ of the transaction t_4 to the weights of the root of the trie τ (lines 5 and 6), we note that $SearchChild(A, r) \neq null$ (line 9), we update the weights of the node c identified by $\epsilon \rightarrow A$ (lines 13 to 15): in one hand $c.\Phi^+[1] \leftarrow c.\Phi^+[1] + \binom{4-1}{1-1} = 2 + 1 = 3$, $c.\Phi^+[2] \leftarrow c.\Phi^+[2] + \binom{4-1}{2-1} = 2 + 3 = 5$ and $c.\Phi^+[3] \leftarrow c.\Phi^+[3] + \binom{4-1}{3-1} = 0 + 3 = 3$, in another hand $c.\Phi^-[1] \leftarrow c.\Phi^-[1] + \binom{4-1}{1} = 2 + 3 = 5$, $c.\Phi^-[2] \leftarrow c.\Phi^-[2] + \binom{4-1}{2} = 0+3 = 3$ et $c.\Phi^-[3] \leftarrow c.\Phi^-[3] + \binom{4-1}{3} = 0+1 = 1$. Now, we compute the weights of the node identified by $\epsilon \rightarrow A \rightarrow B$ by following the same reasoning applied to the previous node. Using functions $CreateNode$ (line 11) and $AddChild$ (line 12), we create the nodes identified by $\epsilon \rightarrow A \rightarrow B \rightarrow C$ and $\epsilon \rightarrow A \rightarrow B \rightarrow C \rightarrow D$ that do not yet exist in the trie according to the function $SearchChild$ (line 9). Finally, we add these last nodes in their place while computing their respective weights (lines 13 to 15).
5. Finally, we return the obtained trie τ (line 16).

Finally, we have a total of 10 occurrences having a length equal to 1, 9 occurrences have a length equal to 2 and 4 occurrences have a length equal to 3.

As we have built the trie of occurrences, if we keep the minimum and maximum length constraints already used during the construction, we will see in the next section that the user can apply any length-based utility measure $m \in \mathcal{M}$ without rebuilding the trie. This is because the trie weights only each length to make an exact draw. So, when the maximal length constraint M don't change, we just iterate on the top array of the trie that has $M - \mu + 1$ values to reprocess the database for a new utility. Let's recall that the maximal length constraint M should not be too large to avoid the long tail problem.

5. TPSampling: Trie-based Pattern Sampling

This section begins by introducing some basic notions relating to the trie of occurrences to better present our approach. Then, it presents the TPSampling algorithm for the exact draw of a pattern proportionally to a given interestingness measure.

5.1. Drawing approach

To draw a pattern of length ℓ proportionally to a length-based utility multiplied by its frequency in the database, we can uniformly draw an occurrence among the set of occurrences of length ℓ. To do this, we first need to draw an

Table 2. Ranking occurrences of length 2

φ_i	CD_4	BD_4	BC_4	AD_4	AC_4	AB_4	AB_1	AC_2	BC_3
$rank_2$	1	2	3	4	5	6	7	8	9

integer $\ell \in [\mu..M]$ proportionally to $\Phi_\ell(\epsilon, \mathcal{T}) \times f_u(\ell)$ with a length-based utility u. Second, we uniformly draw a length ℓ of an occurrence among the set of length ℓ of occurrences in the database, but directly from the trie.

The drawing of an occurrence of a given length is not trivial because we do not know a priori where it is in the trie. This means that it is necessary to cleverly scan the nodes of the trie to find the sought occurrence. Of course, there already exist numbering systems for tree traversal (like prefix or postfix orders) but in our case, the goal is not to number the nodes of the trie, but the occurrences of patterns represented by the trie. The intuition of the numbering system that we use can be summarized as follows:

- It is a recursive and postfix traversal (in depth and from left to right). At the level of the occurrences represented in a sub-trie, they are numbered from left to right, the children of a root of a sub-trie being ordered,
- At the level of a sub-trie, from the root identified by a prefix P, we give a lower rank to occurrences without the label $\widetilde{P}$ than others containing $\widetilde{P}$.

Definition 13 (Ranking occurrences by length). If φ_j is an occurrence of length ℓ from a database $\mathcal{T}$ and τ is a trie constructed from $\mathcal{T}$, we denote $rank_\ell(\varphi_j, \tau) = rank_\ell(\varphi_j, \tau_\epsilon)$ the rank of this occurrence relative to the trie τ. This rank can be defined recursively as follows:

–If $\varphi_j \in \phi^-_\ell(P, \mathcal{T})$, and more precisely if $\varphi_j \in \phi_\ell(P \circ e_i, \mathcal{T})$ where $e_i = \tau_P.child[i]$, then $rank_\ell(\varphi_j, \tau_P) = \sum_{k=1}^{i-1} \Phi_\ell(P \circ e_k, \tau) + rank_\ell(\varphi_j, \tau_{P \circ e_i})$.

–If $\varphi_j \in \phi^+_\ell(P, \mathcal{T})$, then $\varphi_j = \widetilde{P} \circ \varphi'_j$. And in this case, if $\varphi'_j \in \phi_{\ell-1}(P \circ e_i, \mathcal{T})$ where $e_i = \tau_P.child[i]$, then $rank_\ell(\varphi_j, \tau_P) = \Phi^-_\ell(P, \tau) + \sum_{k=1}^{i-1} \Phi_{\ell-1}(P \circ e_k, \tau) + rank_{\ell-1}(\varphi'_j, \tau_{P \circ e_i})$.

–Finally, if φ_j is an occurrence of length 1, we define $rank_1(\varphi_j, \tau_P)$ by: $rank_1(\varphi_j, \tau_P) = rank_{>_\mathcal{I}}(\varphi_j, \phi_1(P, \mathcal{T}))$ where $>_\mathcal{I}$ defined on items is extended to occurrences of length 1 as follows: $(i, e) >_\mathcal{I} (j, e')$ if $e' >_\mathcal{I} e$ or $e = e'$ and $i > j$.

Example 11. Considering only the patterns of length 2 and the total order relation $>^{freq}_\mathcal{I}$, the list in Table 2 gives the rank of each occurrence in the trie of Figure 3:
$\phi_2(\epsilon, \mathcal{T}) = \{CD_4, BD_4, BC_4, AD_4, AC_4, AB_4, AB_1, AC_2, BC_3\}$. We know that $CD_4 \in \phi^-_2(A, \mathcal{T})$, then $rank_2(CD_4, \tau_\epsilon) = rank_2(CD_4, \tau_A) = rank_2(CD_4, \tau_{AB}) = rank_2(CD_4, \tau_{ABC}) = \Phi^-_2(ABC, \tau) + rank_1(D, \tau_{ABC}) = 0 + rank_{>^{freq}_\mathcal{I}}(D, \{D\}) = 1$.
Let us compute $rank_2(AB_4, \tau_\epsilon)$. We know that $AB_4 \in \phi^+_2(A, \mathcal{T})$, then $rank_2(AB_4, \tau_\epsilon) = \Phi^-_2(A, \tau) + 0 + rank_1(B_4, \tau_{AB}) = 3 + rank_{>^{freq}_\mathcal{I}}(B_4, \{B_1, B_4, C_4, D_4\})$. Or $(4, D) >^{freq}_\mathcal{I} (4, C) >^{freq}_\mathcal{I} (4, B) >^{freq}_\mathcal{I} (1, B)$, then $rank_{>^{freq}_\mathcal{I}}(B_4, \{B_1, B_4, C_4, D_4\}) = 3$, so $rank_2(AB_4, \tau_\epsilon) = 3+3 = 6$.

In this list of occurrences, to draw a pattern proportionally to its frequency, it

suffices to uniformly draw a rank between 1 and the highest rank, and return the occurrence which has that rank. For example, we have 2 occurrences of AB in ranks 6 and 7. This means that the probability of drawing the pattern AB among the set of patterns of length exactly equal to 2 is $\frac{2}{9}$, therefore proportional to its frequency. If the user chooses the frequency as interestingness measure, then from the trie of occurrences, we draw $\ell = 2$ with a probability of $\frac{9}{23}$. Therefore, we can note that the pattern AB is drawn in the trie with a probability equal to $\frac{9}{23} \times \frac{2}{9} = \frac{2}{23}$, therefore proportional to its frequency in the database.

5.2. Trie-based pattern sampling algorithm

Algorithm 2 takes as input a trie of occurrences τ, a length-based utility $u \in \mathcal{U}$, and minimum μ and maximum M length constraints. It returns a pattern φ drawn proportionally to its interest in the database corresponding to the trie.
Draw a length ℓ between μ and M. Line 4 draws an integer ℓ between μ and M proportionally to the number of occurrences of length ℓ, i.e. $\tau.\Phi^{-}[\ell]$ multiplied by the utility of a length ℓ, $f_u(\ell)$.
Uniform drawing of an occurrence of length ℓ. To sample an occurrence of length ℓ, we uniformly draw a rank x in the interval $[1..\tau.\Phi^{-}[\ell]]$ (line 5). To find the occurrence corresponding to x, we scan the trie in depth-first search from left to right by looking for the nodes that satisfy the system of inequalities in line 7 which is based on Definition 13. This system of inequalities makes it possible to find the rank of the occurrence from the trie of root τ. Whenever we encounter a node verifying the system of inequalities, we test whether the item it contains is a candidate for the pattern to be returned (line 9), and we add it to the pattern if necessary (line 10). In line 13, we consider the sub-trie whose node satisfying the system of inequalities is the root. Thus, the new rank to visit is the one obtained by subtracting from the old value of x the sum of the weights of the $i-1$ first children of the current node, father of η_i, (line 8) and the negative input to node η_i (line 11). We will then look for the remaining $\ell - 1$ items of the pattern to be returned in the sub-trie of root η_i. The process is iterated until the current value of ℓ is equal to 0. The set of items selected at the different visited nodes form the pattern to return at line 14. So, given a positive integer k_1, $\mathcal{S}amp\ell e_{k_1}(\mathcal{L}_{[\mu..M]}, \mathcal{T}, m_u)$, where $m_u = f_u(|\varphi|) \times \mathit{freq}(\varphi, \mathcal{T})$, is computed by repeating lines 3-14 of Algorithm 2 k_1 times. Furthermore, given a length-based utility u', we just need to replace $f_u(\ell)$ by $f_{u'}(\ell)$ in line 4 in order to compute efficiently $\mathcal{S}amp\ell e_{k_2}(\mathcal{L}_{[\mu..M]}, \mathcal{T}, m_{u'})$, with $0 < k_2$.

Example 12 shows a case of execution of Algorithm 2 for drawing a pattern proportionally to the frequency.

Example 12. Having uniformly drawn a rank $x = 5$ of an occurrence among those of length 2, we are going to show at Figure 5, with Algorithm 2, how to find the corresponding occurrence in the trie of occurrences of Figure 3.

The search for the fifth occurrence among the set of occurrences of length 2 requires 3 iterations. During the first iteration, we select the first child c of the root of the trie τ of identifier $P = A$ because $0 < x \leq (c.\Phi^{-}[2] + c.\Phi^{+}[2] = 3 + 5)$ (line 7), and also we have $x > c.\Phi^{-}[2]$ at line 9. Therefore, item A is part of the occurrence to be returned as output and the current content of the occurrence is $\varphi = A$. The value of ℓ becomes 1 because we are now going to look at the patterns of length $\ell - 1 = 1$ considering the sub-trie τ_A then return the $5 - 3 = 2$-

Algorithm 2 TPSAMPLING

1: **Input :** A trie τ of occurrences of a database $\mathcal{T}$, a length-based utility $u \in \mathcal{U}$ and the minimum and maximum length constraints μ and M
2: **Output :** A pattern φ drawn proportionally to its interest $\varphi \sim f_u(|\varphi|) \times freq(\varphi, \mathcal{T})$
3: $\varphi \leftarrow \emptyset$
4: Draw a length ℓ proportionally to $\tau.\Phi^-[\ell] \times f_u(\ell)$ where $\ell \in [\mu..M]$
5: Draw uniformly a rank x in $[1..\tau.\Phi^-[\ell]]$
6: **while** $(\ell > 0)$ **do**
7: Find the i^{th} child $\eta_i \in \tau.child$ such that :

$$\sum_{1 \leq k < i} \left(\tau.child[k].\Phi^+[\ell] + \tau.child[k].\Phi^-[\ell]\right) < x \leq \sum_{1 \leq k \leq i} \left(\tau.child[k].\Phi^+[\ell] + \tau.child[k].\Phi^-[\ell]\right)$$

8: $x \leftarrow x - \sum_{1 \leq k < i} \left(\tau.child[k].\Phi^+[\ell] + \tau.child[k].\Phi^-[\ell]\right)$
9: **if** $(x > \eta_i.\Phi^-[\ell])$ **then** ▷ Check if the label of the current node is part of the pattern
10: $\varphi \leftarrow \varphi \cup \eta_i.label$
11: $x \leftarrow x - \eta_i.\Phi^-[\ell]$
12: $\ell \leftarrow \ell - 1$
13: $\tau \leftarrow \eta_i$
14: **return** φ

$\ell \leftarrow 2$ et $x \leftarrow 5$

1^{st} iteration	2^{nd} iteration	3^{rd} iteration
L7: $0 < x \leq 5 + 3$	L7: $0 < x \leq 2 + 2$	L7: $0 < x \leq 1 + 1$
L8: $x \leftarrow 5 - 0 = 5$	L8: $x \leftarrow 2 - 0 = 2$	L8: $x \leftarrow 2 - 0 = 2$
L9: $x > 2$	L9: $x \not> 2$	L9: $x > 1$
L10: $\varphi \leftarrow A$		L10: $\varphi \leftarrow AC$
L11: $x \leftarrow 5 - 3 = 2$		L11: $x \leftarrow 2 - 1 = 1$
L12: $\ell \leftarrow 2 - 1 = 1$		L12: $\ell \leftarrow \mathbf{1 - 1 = 0}$
L13: $r \leftarrow r.fils[1] = \tau_A$	L13: $r \leftarrow r.fils[1] = \tau_{AB}$	L13: $r \leftarrow r.fils[1] = \tau_{ABC}$

Fig. 5. Example of drawing an occurrence under length constraint returning AC_4

nd occurrence of length 1 in τ_A. In the second iteration, we select the first child c of the root of the sub-trie τ_A of identifier $P = AB$ because $0 < x \leq (c.\Phi^-[1] + c.\Phi^+[1] = 2 + 2)$ (line 7), and also we have $x < c.\Phi^-[1]$ at line 9 and therefore item B is not part of the occurrence to return, the value of ℓ is always the same. During the 3rd iteration, we select the first child c of the root of the sub-trie τ_{AB} of identifier $P = ABC$ because $0 < x \leq (c.\Phi^-[1] + c.\Phi^+[1] = 1 + 1)$ (line 7). We then note that $x > c.\Phi^-[1]$ and therefore, the item C is part of the occurrence to be returned. The current content of the occurrence is equal to $\varphi = AC$. The value of ℓ is now equal to 0 (line 12) which means that TPSAMPLING stops and returns the pattern $\varphi = AC$. We also note that the fifth occurrence of length 2 is indeed AC in Table 2.

6. Theoretical analysis

This section makes a theoretical study of our approach for sampling from a trie of occurrences. First, it presents the proof of the correctness of drawing a pattern. Secondly, it shows the space complexity (i.e., memory storage) of the trie of occurrences. Third and last, it presents the temporal complexities of our two algorithms for building a trie of occurrences by TPSpace and for drawing a pattern from a trie of occurrences by TPSampling.

6.1. Correction

Property 3 shows that our sampling method TPSampling does an exact draw of a pattern.

Property 3. Let τ a trie of occurrences from a transactional database and u a length-based utility, Algorithm 2 draws a pattern φ according to a probability proportional to its frequency weighted by its utility.

Proof. Let φ_i be an occurrence of a pattern of length ℓ. The probability that φ_i is returned is $\pi(\ell) \times \pi(\varphi_i/\ell)$. Let $Z = \sum_{\ell} f_u(\ell) \times \tau.\Phi^-[l]$. We know from line 4 that $\pi(\ell) = f_u(\ell) \times \tau.\Phi^-[l]/Z$. According to line 5, the rank of an occurrence ℓ is drawn uniformly among the ranks of the occurrences of the same length. Lines 9 to 12 use the definition 13 in order to access the occurrence of a given rank. So from lines 6 to 13, we have $\pi(\varphi_i/\ell) = 1/\tau.\Phi^-[l]$. It follows that $\pi(\varphi_i) = \frac{f_u(\ell) \times \tau.\Phi^-[l]}{Z} \times \frac{1}{\tau.\Phi^-[l]}$. So $\pi(\varphi_i) = \frac{f_u(\ell)}{Z}$. Which ultimately gives $\pi(\varphi) = \frac{f_u(\ell)}{Z} \times freq(\varphi, \mathcal{T})$. Hence the result. □

6.2. Space complexity

The size of memory occupied by the trie of occurrences is not negligible, especially with large datasets. However, it can be optimized during the preprocessing phase by reducing the number of nodes. For example, considering that the items are ordered according to the decreasing order of their frequencies, the number of nodes in the trie is much lower than the upper bound which is in $O(2^{|\mathcal{I}|})$. In that case, a tight upper bound of the number of nodes is detailed in [31].

The size of a trie of occurrences also depends on the information stored in the nodes. In our case, the higher the maximum length constraint, the larger the arrays and the greater the memory size. This means that if the number of nodes in the trie of occurrence is z, μ and M the minimum and maximum length constraints respectively, then the size in memory of the trie is in $O(z \times 2 \times (M - \mu))$. Fortunately, the maximum length constraint must generally be small to avoid the long tail problem. It is also important to note that, to have a good practical consumption of memory storage, we do not materialize the columns of tables that only contain zero values. This trick counterbalances the impact of the maximum length constraint increase.

Example 13. Figure 6 is a version of the trie of occurrences that we represented in Figure 3. In this case, we have omitted the columns that contain only zeros (0).
If we take a maximum length constraint $M = 4$, then only the array of the node

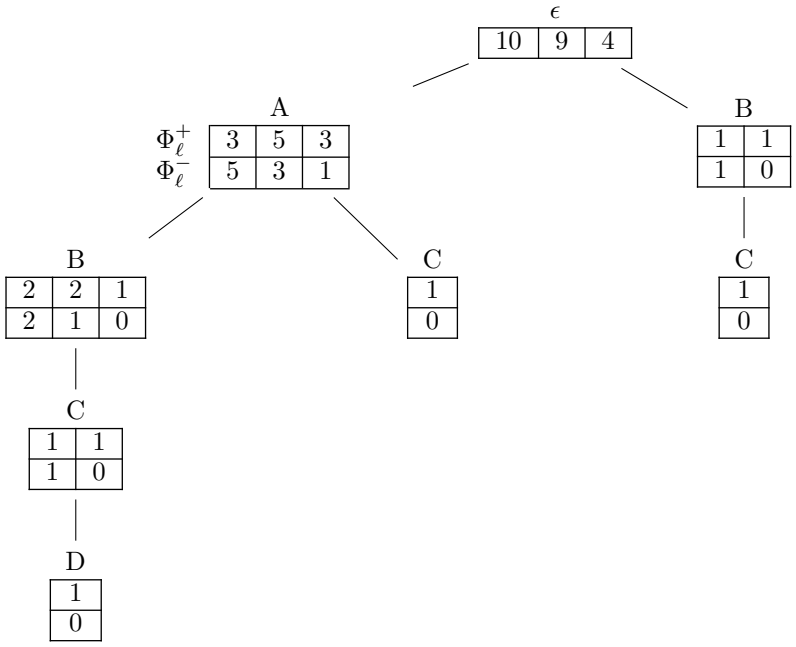

Fig. 6. Storage in practice of the enriched trie in Figure 3

of identifier A and that of the root will be impacted. We will have in particular one more column at the identifier node A containing 1 at the index 1 (to count the pattern $ABCD$ of size 4) and 0 at the index 2. At the root, we will have one more square to count the pattern 4. If now, we have $M = 5$, then the trie remains unchanged because no pattern of length 5 appears in any transaction of the database $\mathcal{T}$.

As can be seen, this optimization can make it possible to have a significant gain on the size in memory of a trie of occurrences compared to the theoretical representation where all the columns are materialized.

6.3. Time complexity

The time complexity of our method can be divided into two steps: the preprocessing time of the construction of the trie of occurrences and the drawing time of an occurrence.

Preprocessing time. It is the most expensive phase of TPSpace. A first pass on the database is necessary to retrieve the items from the database $\mathcal{T}$ and compute their frequencies in $O(||\mathcal{T}||)$ where $||\mathcal{T}||$ is the sum of the lengths transactions from the database $\mathcal{T}$. The previously retrieved items are ordered according to the total order relation chosen $>_\mathcal{I}$ in $O(|\mathcal{I}| \times \log(|\mathcal{I}|))$. Then, before adding a transaction to the trie, we order its items according to the total order relation $>_\mathcal{I}$ in $O(T_{\max} \times \log(T_{\max}))$ where $T_{\max}$ is the maximum length of transactions in the database $\mathcal{T}$. Finally, if z is the total number of nodes

in the trie following the total order relation $>_{\mathcal{I}}$, μ and M the minimum and maximum length constraints respectively, then the weighting of the nodes is done in $O(z \times 2 \times (M \times \mu))$. Thus, the total complexity for building the trie of occurrence of the transactional database $\mathcal{T}$ built on the set of literals $\mathcal{I}$ is in $O(||\mathcal{T}|| + |\mathcal{I}| \times \log(|\mathcal{I}|) + |\mathcal{T}| \times T_{\max} \times \log(T_{\max}) + z \times (M \times \mu))$. However, remember that this preprocessing of a database is done only once.

Reprocessing time for utility change. When the utility changes without updating the length constraints μ and M, the complexity of the reprocessing time is in $O(M - \mu)$, which is particularly fast. This is because only the array of the root of the trie is traveled to compute the new weight of each length $\ell \in [\mu..M]$.

It's clear that in this paper we deal only with utility change for the reprocessing because we suppose that M is large and no longer needed to be increased to avoid the long tail problem. But, when the user needs to increase the maximum length constraint from M to M', the complexity of the reprocessing is bounded by $O((M' - M) \times z)$, where z is the total number of nodes in the trie. In fact, the weights of $M + 1$ can be deduced from those of M, and those of M' from those of $M' - 1$.

Drawing time of an occurrence. Let us denote by d the degree (number of children) of a node of the trie and by d_{max} the maximum degree of the trie, $d_{\max} \leq |\mathcal{I}|$. Line 7 of TPSampling finds i^{th} node in $O(\log(d_{max}))$. Thus, by going deeply through the trie of occurrences, TPSampling draws an occurrence in $O(T_{\max} \times \log(d_{max}))$. So, a sample of k patterns is obtained by TPSampling in $O(k \times T_{\max} \times \log(d_{max}))$. This complexity is comparable to that of the two-step algorithm [4] (with length constraints) which draws a sample of k patterns in $O(k \times T_{\max} \times \log(|\mathcal{T}|))$.

7. Experiments

This experimental section aims to assess the efficiency of our approach to large transactional datasets. It's important to note that our main goal is not to compare pattern sampling algorithms with exhaustive pattern mining methods based on compact structure [32, 33]. Previous works [3, 12, 10] already justified the advantages of output pattern sampling over exhaustive pattern mining. The difficulties encountered in exhaustive pattern mining, such as setting the threshold or controlling the output size, are not resolved by the compact representation.

So, we first evaluate the memory size occupied by the trie of occurrences according to the maximum length constraint and the total order relation on the items $>_{\mathcal{I}} \in \{>_{\mathcal{I}}^{freq}, >_{\mathcal{I}}^{lexico}\}$. Second, we study the building time of a trie of occurrences according to the maximum length constraint on different datasets and then evaluate the speed of drawing a pattern by TPSampling. Finally, we show the scalability of TPSampling on large databases. The experiments were conducted with 4 UCI databases for `Connect`, `Pumsb`, `Susy`, and `USCensus`. We also used 2 synthetic datasets built with the IBMGenerator[1] `T10I4D2000K` and

[1] https://github.com/zakimjz/IBMGenerator

Table 3. Characteristics of datasets

$\mathcal{T}$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert t\rvert_{\min}$	$\lvert t\rvert_{\max}$	$\lvert t\rvert_{avg}$
Connect	129	67,557	43	43	43.00
Pumsb	2,113	49,096	74	74	74.00
USCensus	396	1,000 K	25	68	68.00
Susy	190	5,000 K	19	19	19.00
T10I4D2000K	2,719	2,000 K	10	30	20.11
T10I6D3000K	3,952	3,000 K	10	35	22.61

Table 4. Characteristics of tries of the datasets

	Nb of nodes in the trie		Gain (in %)
	$>_{\mathcal{I}}^{freq}$	$>_{\mathcal{I}}^{lexico}$	$\frac{Nb_{lexico}-Nb_{freq}}{Nb_{lexico}}$
Connect	359,291	1,014,837	64.60
Pumsb	1,126,154	2,629,640	57.17
USCensus	312,808	607,611	48.52
Susy	10,424,240	12,630,372	17.47
T10I4D2000K	9,957,321	–	–
T10I6D3000K	10,617,309	–	–

T10I6D3000K. These synthetic datasets allow us to evaluate our approach with large transactional databases. Table 3 details on the one hand the characteristics of the benchmarks according to the number of items, of transactions, and the maximum and the average length of the transactions. On the other hand, it presents the number of nodes in the trie corresponding to each database and according to the total order of the items. For sampling with length constraint, the value of the minimum length constraint is fixed at $\mu = 1$ throughout the experiments. The prototype of our method is implemented in Python version 3 and all the experiments are performed on a 2.71 GHz 2 Core CPU with 12 GB RAM. All experimental data sets used, as well as the source code, are available at https://github.com/TPSampling/TPSampling. We also implement an approach combining the two-step algorithm [4] with maximum length constraints as a baseline.

7.1. Storage cost of the trie of occurrences

The cost[2] of storing a trie of occurrences in a database depends on both the total order relation $>_{\mathcal{I}}$ and the maximum length constraint.

As we can see in Table 4, the number of nodes depends on the order in which the items are inserted in the trie. Overall, we noted that the number of nodes is much smaller with the $>_{\mathcal{I}}^{freq}$ relation than with the $>_{\mathcal{I}}^{lexico}$ relation according to the gain obtained in the last column. For example, with the Connect database, the number of nodes in the trie built with the order relation $>_{\mathcal{I}}^{freq}$ is 64.6% less than the number of nodes in the trie built with the total order relation

[2] Computed with the python package asizeof http://code.activestate.com/recipes/546530-size-of-python-objects-revised/

$>_{\mathcal{I}}^{lexico}$. This characteristic is very important in our proposal because each node must contain two arrays whose size depends on the maximum length constraint. Thus, the smaller the number of nodes, the lower the storage cost. With the gain obtained in Table 4, we can already note that the total order relation plays a very important role in optimizing the cost of memory storage, especially with large datasets like `Susy`, `T10I4D2000K`, and `T10I6D3000K`. In the two last datasets, it is not possible to run TPSampling with le lexicographical order due to an exception of "Out of memory".

Figure 7 shows the evolution of the memory size required by the tries of each database according to the maximum length constraint $M \in [2..10]$ and the chosen order relation. We take as a baseline the storage cost of the tabular representations of the databases weighted according to the maximum length constraints of the two-step method [4](with length constraints).
We first observe that the memory size of tries of occurrences increases according to the maximum length constraint. Indeed, the two-dimensional matrixes that make it possible to count the patterns by length increase in size according to the maximum length constraint. This increase is more remarkable in the case where we considered the relation of total order $>_{\mathcal{I}}^{lexico}$ because the number of nodes in the trie constructed with the relation of total order $>_{\mathcal{I}}^{lexico}$ is higher than that of the trie built with the relation $>_{\mathcal{I}}^{freq}$. Therefore, the size of the trie is more expensive with the $>_{\mathcal{I}}^{lexico}$ relation than with the $>_{\mathcal{I}}^{freq}$ relation. Thus, in the case where the database is very large, for example with `Susy`, `T10I4D2000K`, and `T10I6D3000K`, it is preferable to insert the items of the instances of the database in the trie following the relation total order $>_{\mathcal{I}}^{freq}$. The evolution of the size of the trie according to the maximum length constraint in memory is almost stationary with the UCI datasets `Connect`, `Pumsb`, `USCensus`, and `Susy`. When the maximum length constraint is greater than 6, `T10I4D2000K` becomes more expensive than `Susy` while the latter has more nodes in its trie of occurrences. This is because `T10I4D2000K` has more nodes near the root of its trie than that of `Susy`. These experimental results show that our approach is sensitive to the total number of items on which the database is built. For instance, TPSampling $+>_{\mathcal{I}}^{freq}$ returns an "Out of memory" exception with `T10I6D3000K` when the maximum length constraint is greater than 7. We also noted that if the database is large `Connect` (67,557 instances having an average length of 43.0) and `Pumsb` (49,096 instances having an average length of 74.0), the memory size of the trie of occurrences constructed according to the order relation $>_{\mathcal{I}}^{freq}$ is 200 times smaller than that of the representation of the database used by the two-step approach [4]. The implementation of the two-Step method generates an "Out of memory" exception with `Susy`, while both Two-Step and TPSampling $+>_{\mathcal{I}}^{lexico}$ return "Out of memory" exception with `T10I4D2000K` and `T10I6D3000K`.

7.2. Speed of the approach

This section studies also the speed of our trie-based sampling method by distinguishing between preprocessing time, reprocessing time, and that of drawing a pattern. We will start evaluating the preprocessing time of each dataset of Table 3 and then compare it with that of the two-step method. We finish by computing the average time to draw a pattern with TPSampling in the 6 datasets according to the maximum length constraint and the chosen interestingness measure.

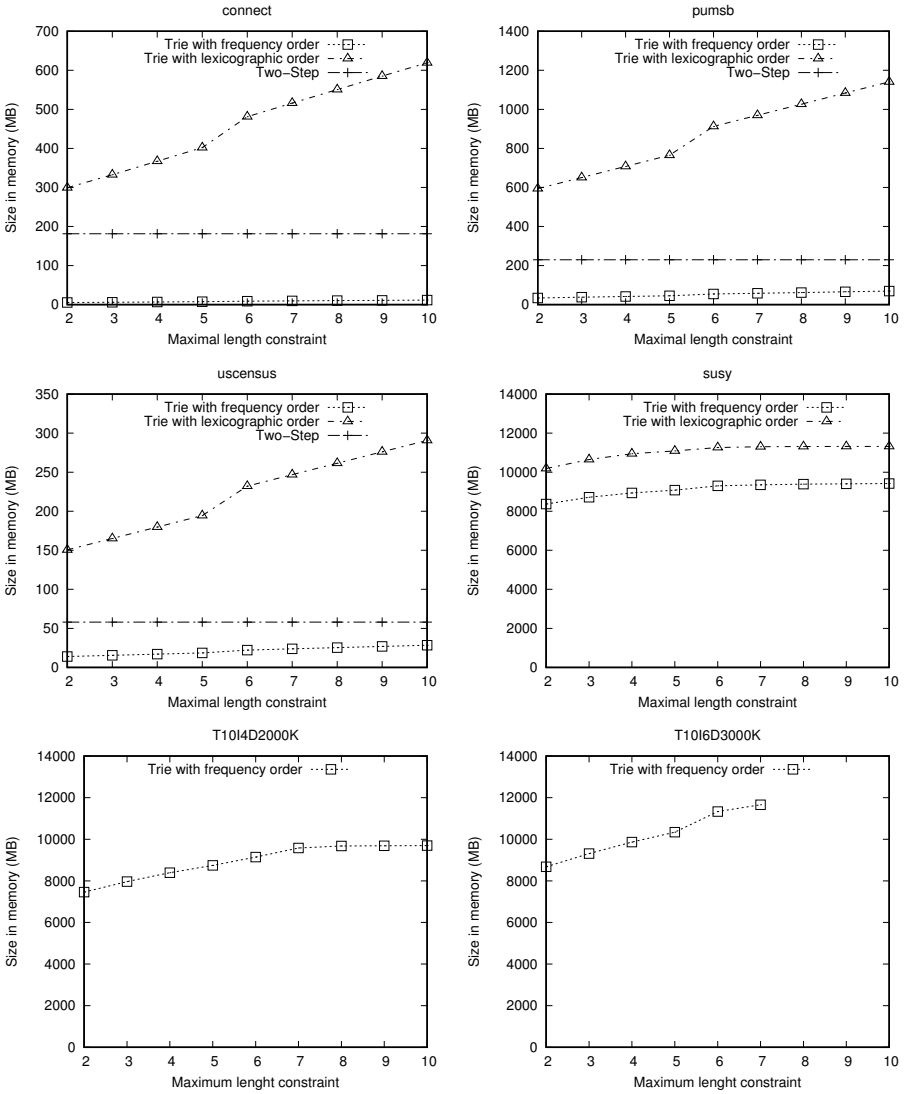

Fig. 7. Evolution of the memory size of the tries according to the length constraint

Evaluation of the preprocessing time. Interestingly, the time to build a trie of occurrences is independent of any length-based interestingness measure. In our experiments, we consider the maximum length constraints $M \in \{2, 6\}$. These values seem sufficient to avoid drawing patterns from the long tail. Table 5 presents the preprocessing times for the construction of the tries of occurrences and those for weighting the databases with the two-step algorithm according to the maximum length constraint. Each experiment is repeated 10 times to have the average preprocessing times and the standard deviations.

As we can see, the two-step method is faster than the trie-based method for

Table 5. Preprocessing times in seconds according to the measures (A = Area, D = Decay, F = Freq) and ($M \in \{2, 6\}$)

		Two-Step ([4]+ constraints)		TPSampling ($>_{\mathcal{I}}^{freq}$)		TPSampling ($>_{\mathcal{I}}^{lexico}$)	
$\mathcal{T}$	m	M=2	M=6	M=2	M=6	M=2	M=6
Connect	A	0.44 ± 0.02	0.44 ± 0.04	9.98 ± 0.30	13.33 ± 3.45	15.57 ± 0.37	25.72 ± 0.69
	D	0.44 ± 0.03	0.44 ± 0.04				
	F	0.43 ± 0.04	0.45 ± 0.03				
Pumsb	A	0.54 ± 0.04	0.53 ± 0.03	18.66 ± 0.40	24.12 ± 5.62	78.89 ± 1.65	94.79 ± 3.24
	D	0.54 ± 0.04	0.54 ± 0.04				
	F	0.56 ± 0.04	0.53 ± 0.04				
USCensus	A	0.01 ± 0.00	0.01 ± 0.00	5.26 ± 1.85	7.39 ± 0.17	8.09 ± 0.32	12.27 ± 1.27
	D	0.01 ± 0.00	0.01 ± 0.00				
	F	0.01 ± 0.00	0.01 ± 0.00				
Susy	A	–	–	347.99 ± 8.01	593.01 ± 44.24	532.01 ± 41.65	910.45 ± 65.47
	D	–	–				
	F	–	–				
T10I4D2000K	A	–	–	356.54 ± 9.31	418.87 ± 24.13	–	–
	D	–	–				
	F	–	–				
T10I6D3000K	A	–	–	792.56 ± 45.21	1094.49 ± 51.42	–	–
	D	–	–				
	F	–	–				

weighting a database. This can be explained by the fact that the preprocessing of the two-step method is done in one pass on the database with linear complexity relative to the size of the database. Contrariwise, TPSpace needs 2 passes in the case where we want to insert the items following a total order relation $>_{\mathcal{I}}$. To this, is added the time to insert the items of each transaction in the trie of occurrence according to the total order relation chosen by the user. We recall that with Susy, T10I4D2000K, and T10I6D3000K databases, an "Out of memory" exception was thrown by the implementation of the two-step algorithm because of their very large sizes after preprocessing which do not hold in memory. With the largest database, T10I6D3000K, the construction of the trie of occurrences according to the total order relation $>_{\mathcal{I}}^{freq}$ lasts on average 13 and 18 minutes with the maximum length constraints $M = 2$ and $M = 6$ respectively.Interestingly, we do this preprocessing once only, then we can use the resulted trie of occurrences with any length-based utility.

Evaluation of the reprocessing time for utility change. Our approach allows users to draw patterns according to any length-based utility measures that do not change the minimum μ and the maximum M length constraints. The reprocessing time, when utility changes, depends only on the minimum and maximum length constraint. It is linear to the difference between the minimum and the maximum length constraints whatever the size of the database because of the reusability of the trie of occurrences. Utility measures like frequency, area, and exponential decay have not a notorious impact on the speed of the reprocessing phase. Contrariwise, Two-Step *[4]+length constraint* should do a new preprocessing when utility changes. Table 6 shows that in the reprocessing phase when the utility changes while our approach needs a few time, less than 10×10^{-6} seconds with $M = 10$. Decay is more consuming than Freq and Area due to the cost of the power function. Interestingly, the reprocessing time is the

Table 6. Reprocessing times in seconds ($\times 10^{-6}$ for TPSampling) according to the measures (A = Area, D = Decay, F = Freq) and ($M \in [2..10]$)

		Two-Step ([4]+ constraints)			TPSampling ($>_{\mathcal{I}}^{freq}$) $\times 10^{-6}$		
M	Datasets	A	D	F	A	D	F
2	Connect	0.44	0.44	0.43	1.47	2.08	1.51
	Pumsb	0.54	0.54	0.56			
	USCensus	0.01	0.01	0.01			
3	Connect	1.06	1.23	0.95	2.08	2.78	1.93
	Pumsb	1.11	1.05	1.12			
	USCensus	0.26	0.33	0.26			
4	Connect	1.08	1.24	1.02	2.73	3.76	2.53
	Pumsb	1.15	1.39	1.03			
	USCensus	0.33	0.33	0.29			
5	Connect	1.09	1.26	1.09	3.33	4.44	3.17
	Pumsb	1.17	1.41	1.09			
	USCensus	0.39	0.42	0.41			
6	Connect	1.13	1.34	1.12	3.98	5.39	3.73
	Pumsb	1.22	1.48	1.03			
	USCensus	0.39	0.43	0.40			
7	Connect	1.17	1.39	1.17	4.69	6.28	4.33
	Pumsb	1.23	1.48	1.09			
	USCensus	0.42	1.46	0.41			
8	Connect	1.28	1.43	1.24	5.30	7.14	4.94
	Pumsb	1.26	1.52	1.10			
	USCensus	0.43	0.48	0.43			
9	Connect	1.29	1.45	1.27	5.87	7.91	5.50
	Pumsb	1.31	1.54	1.21			
	USCensus	0.44	0.51	0.44			
10	Connect	1.37	1.46	1.31	6.50	9.09	6.09
	Pumsb	1.36	1.55	1.29			
	USCensus	0.45	0.57	0.46			

same for all datasets with the same length constraints. These results show how important is the trie data structure for user-centered pattern mining.

Evaluation of the drawing time per pattern. Now, we will evaluate the speed of drawing a pattern with our approach on the 6 datasets of Table 3. Precisely, we are going to compute the average drawing time of a pattern according to the maximum length constraint belonging to the interval [1..10] and an exponential decay with a fixed value $\alpha = 0.1$. Figure 8 shows the evolution of the average drawing time of a pattern by TPSampling compared to that of the two-step method with length constraints. Each average value is obtained by repeating, for each value of $M \in [1..10]$, 100 times the draw of a pattern. Standard deviations are omitted as they are very small.

As we can see, with the TPSampling method, the drawing times are less than 0.2 millisecond on Connect, Pumsb, USCensus, and Susy datasets used with a maximum length constraint $M \in [1..10]$. With large databases T10I4D2000K and T10I6D3000K, TPSampling lasts at most 2.5 milliseconds to draw a pattern

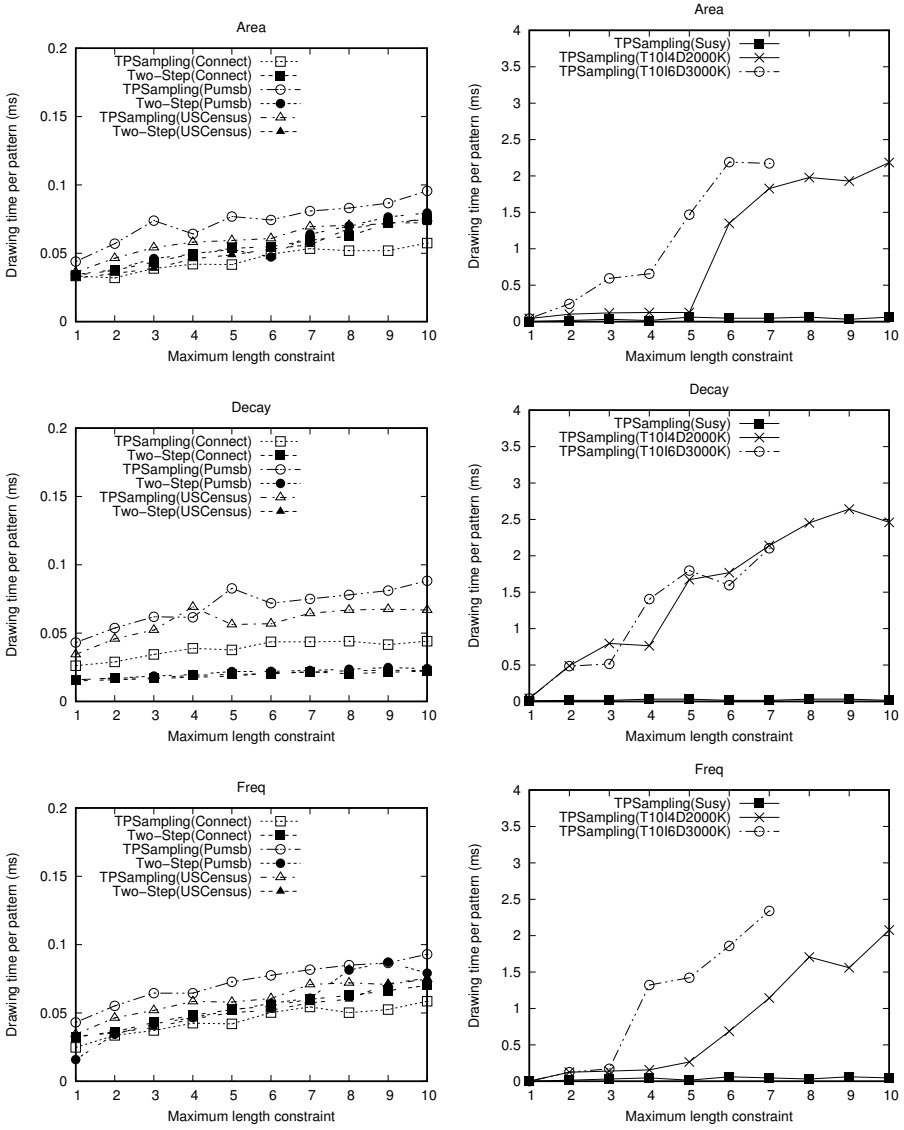

Fig. 8. Evolution of the average of drawing time per pattern according to M ($\alpha = 0.1$ for the exponential decay)

while $M \in [1..7]$. When $M > 7$, TPSampling returns an "Out of memory" exception with `T10I6D3000K` dataset. On `Connect`, `Pumsb`, and `USCensus` datasets, we can note that the speeds of the two methods are almost equal. Indeed, for the drawing of a pattern of length ℓ within a transaction t, the two-step algorithm exactly visits ℓ items of t. Conversely, TPSampling can visit more than ℓ nodes before returning the corresponding pattern. We note on the one hand that the drawing times of TPSampling+Area and TPSampling+Freq evolve almost

in the same way. On the other hand, the drawing time of a pattern increases slightly according to the maximum length constraint with `Connect`, `Pumsb`, and `USCensus`. We can say that TPSampling is almost as fast as the two-step algorithm to draw thousands of patterns per second.

8. Conclusion

This paper proposed a new method of output pattern sampling based on a compact data structure called "trie". Considering a trie of occurrence built by our first algorithm TPSpace, it shows how to directly sample patterns according to any length-based utility measure with the second algorithm TPSampling. It is a generic algorithm that draws patterns from a trie of occurrences following a distribution proportional to a length-based utility measure chosen by the user. Experiments show that TPSpace is very parsimonious in storage cost and TPSampling is as good as the two-step method with length constraints to draw thousands of patterns per second. Unfortunately, the effectiveness of TPSpace decreases when the distinct items used in the database are very numerous. But unlike the two-step approaches, the trie of occurrences is the same for any length-based utility measure provided that the same values are kept for the maximum length constraint. For instance, once the trie of occurrences is built with a given minimum and maximum length constraints, the user can then draw patterns with frequency, area, and any exponential decay $\alpha \in]0, 1]$ with the same built trie. They also show that our proposed method scales very well with large transactional databases.

In future work, we would like to draw sequential patterns from a trie of occurrences. Indeed, the pattern numbering system we used to draw a sample proportionally to a length-based utility measure is a promising avenue for avoiding the rejection method used for sequential pattern mining [12]. By resolving the problem of rejection in sequential pattern sampling, the numbering system we present in this paper becomes a very promising perspective to draw a representative set of patterns from the sequential data stream. In that case, each sequence arriving into the system is processed like a trie with one branch where a pattern can be drawn just by having its number among the set of distinct sub-sequences within the sequence.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93, 1993, pp. 207–216.

[2] P. Fournier-Viger, A. Gomariz, T. Gueniche, E. Mwamikazi, and R. Thomas, "Tks: efficient mining of top-k sequential patterns," in *International Conference on Advanced Data Mining and Applications*. Springer, 2013, pp. 109–120.

[3] M. Al Hasan and M. J. Zaki, "Output space sampling for graph patterns," *Proc. of the VLDB Endowment*, vol. 2, no. 1, pp. 730–741, 2009.

[4] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner, "Direct local pattern sampling by efficient two-step random procedures," in *Proc. of the 17th ACM SIGKDD*, 2011, pp. 582–590.

[5] A. Giacometti and A. Soulet, "Anytime algorithm for frequent pattern outlier detection," *International Journal of Data Science and Analytics*, vol. 2, no. 3-4, pp. 119–130, 2016.

[6] M. van Leeuwen, *Interactive Data Exploration Using Pattern Mining*. Berlin,

Heidelberg: Springer Berlin Heidelberg, 2014, pp. 169–182. [Online]. Available: https://doi.org/10.1007/978-3-662-43968-5_9

[7] V. Dzyuba, M. v. Leeuwen, S. Nijssen, and L. De Raedt, "Interactive learning of pattern rankings," *Int. Journal on Artificial Intelligence Tools*, vol. 23, no. 06, p. 32 pages, 2014.

[8] A. Giacometti and A. Soulet, "Interactive pattern sampling for characterizing unlabeled data," in *Proc. of IDA 2017*. Springer, 2017, pp. 99–111.

[9] V. Dzyuba, M. van Leeuwen, and L. De Raedt, "Flexible constrained sampling with guarantees for pattern mining," *Data Mining and Knowledge Discovery*, pp. 1266–1293, 2017.

[10] L. Diop, C. T. Diop, A. Giacometti, and A. Soulet, "Pattern sampling in distributed databases," in *Advances in Databases and Information Systems*, J. Darmont, B. Novikov, and R. Wrembel, Eds. Cham: Springer International Publishing, 2020, pp. 60–74.

[11] L. Diop, C. T. Diop, A. Giacometti, D. Li Haoyuan, and A. Soulet, "Sequential Pattern Sampling with Norm Constraints," in *IEEE International Conference on Data Mining (ICDM)*, Singapore, Singapore, Nov. 2018.

[12] L. Diop, C. T. Diop, A. Giacometti, D. Li, and A. Soulet, "Sequential pattern sampling with norm-based utility," *Knowledge and Information Systems (KAIS)*, Oct. 2019.

[13] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, May 2000.

[14] L. Diop, C. T. Diop, A. Giacometti, and A. Soulet, "Pattern on demand in transactional distributed databases," *Information Systems*, vol. 104, p. 101908, 2022.

[15] M. Boley, T. Gärtner, and H. Grosskreutz, "Formal concept sampling for counting and threshold-free local pattern mining," in *Proc. of SDM 2010*. SIAM, 2010, pp. 177–188.

[16] M. Bhuiyan, S. Mukhopadhyay, and M. A. Hasan, "Interactive pattern mining on hidden data: a sampling-based solution," in *Proc. of ACM CIKM*, 2012, pp. 95–104.

[17] M. Boley, S. Moens, and T. Gärtner, "Linear space direct pattern sampling using coupling from the past," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 69–77.

[18] G. Li and M. J. Zaki, "Sampling minimal frequent boolean (DNF) patterns," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 87–95.

[19] S. Moens and B. Goethals, "Randomly sampling maximal itemsets," in *Proc. of IDEA Workshop 2013*, 2013, pp. 79–86.

[20] S. Moens and M. Boley, "Instant exceptional model mining using weighted controlled pattern sampling," in *Proc. of IDA 2014*. Springer, 2014, pp. 203–214.

[21] A. A. Bendimerad, M. Plantevit, and C. Robardet, "Unsupervised exceptional attributed sub-graph mining in urban data," in *Proc. of ICDM 2016*. IEEE, 2016, pp. 21–30.

[22] A. Giacometti and A. Soulet, "Dense neighborhood pattern sampling in numerical data," in *Proc. of SDM 2018*, 2018, pp. 756–764.

[23] M. Gueguen, O. Sentieys, and A. Termier, "Accelerating itemset sampling using satisfiability constraints on FPGA," in *IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2019.

[24] A. Giacometti and A. Soulet, "Reservoir pattern sampling in data streams," in *Machine Learning and Knowledge Discovery in Databases. Research Track*, N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 337–352.

[25] P. Efraimidis and P. Spirakis, *Weighted Random Sampling*. Boston, MA: Springer US, 2008, pp. 1024–1027. [Online]. Available: https://doi.org/10.1007/978-0-387-30162-4_478

[26] K.-H. Li, "Reservoir-sampling algorithms of time complexity o(n(1 + log(n/n)))," *ACM Transactions on Mathematical Software*, vol. 20, pp. 481–493, 1994.

[27] D. E. Knuth, *the Art of Computer Programming*. Reading, Massachusetts: Addison–Wesley, 1968, Third edition, 1997.

[28] A. Ferrández and J. Peral, "Mergedtrie: Efficient textual indexing," *PLOS ONE*, vol. 14, no. 4, pp. 1–19, 04 2019.

[29] F. Bodon and L. Rónyai, "Trie: An alternative data structure for data mining algorithms," *Mathematical and Computer Modelling*, vol. 38, no. 7, pp. 739 – 751, 2003, hungarian Applied Mathematics.

[30] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[31] N. Shahbazi and J. Gryz, "Upper bound on the size of fp-tree," in *Advances in Databases*

and Information Systems, J. Darmont, B. Novikov, and R. Wrembel, Eds. Cham: Springer International Publishing, 2020, pp. 23–33.

[32] F. Bodon, "A trie-based apriori implementation for mining frequent item sequences," in *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, ser. OSDM '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 56–65. [Online]. Available: https://doi.org/10.1145/1133905.1133913

[33] E. Ansari, G. Dastghaibyfard, M. Keshtkaran, and H.Kaabi, "Distributed frequent itemset mining using trie data structure," *IAENG International Journal of Computer Science*, vol. 35, 01 2008.