Trie-based Output Space Itemset Sampling

Lamine Diop*†, Cheikh Talibouya Diop‡, Arnaud Giacometti* and Arnaud Soulet*
*University of Tours, 3 Place Jean Jaurès 41029 Blois, France, firstname.lastname@univ-tours.fr
‡ University Gaston Berger of Saint-Louis, BP 234, Saint-Louis, Senegal, cheikh-talibouya.diop@ugb.edu.sn
†EPITA Research Laboratory (LRE), FR-94276 Le Kremlin-Bicetre, lamine.diop@epita.fr

***Abstract*—Pattern sampling algorithms produce interesting patterns with a probability proportional to a given utility measure. Utility changes need quick re-preprocessing when sampling patterns from large databases. In this context, existing sampling techniques require storing all data in memory, which is costly. To tackle these issues, this work enriches D. Knuth's trie structure, avoiding 1) the need to access the database to sample since patterns are drawn directly from the enriched trie and 2) the necessity to reprocess the whole dataset when utility changes. We define the trie of occurrences that our first algorithm TPSpace (Trie-based Pattern Space) uses to materialize all of the database patterns. Factorizing transaction prefixes compresses the transactional database. TPSampling (Trie-based Pattern Sampling), our second algorithm, draws patterns from a trie of occurrences under a length-based utility measure. Experiments show that TPSampling produces thousands of patterns in seconds.**

***Index Terms*—Pattern mining, Big data, Pattern sampling, Itemset, Trie data structure**

I. Introduction

Pattern mining [1] is an active research field that aims at discovering interesting and non-trivial information in large databases. Methods for discovering relevant patterns in a transactional database are known as itemset mining methods. During the last decade, researchers in this field addressed the pattern explosion problem, which was caused by the combination of the volume of data and the combinatorial nature of mining methods. In fact, controlling the size of the set of frequent patterns given a minimum threshold, for example, is extremely difficult. On the one hand, if the minimum threshold is very low, the end-user is overwhelmed by the number of returned patterns. However, if the minimum threshold is very high, the set of patterns may be empty. Many approaches are proposed to solve this problem, such as Top-k pattern mining [2], which returns the k most frequent patterns but lacks diversity. The last approach proposed is based on output pattern sampling [3]. Output pattern sampling is the process of selecting a sample of patterns from the entire dataset. It is a non-exhaustive method for discovering relevant patterns that provides high statistical guarantees thanks to its random nature. Pattern sampling has been shown to be an important component of interactive pattern-based mining systems by providing anytime methods [4] and by integrating user feedback [5], [6].

With state-of-the-art sampling methods, the entire database must be stored in memory, except [7], where the data are natively decentralized in different sites and the algorithm executed in a single machine. In our case, we assume all transactions are on a single machine where the user can run a sampling algorithm. In large databases, a compact data structure can be used to compress the database, as was done by Han et al. [8] for exhaustive frequent patterns mining. No output pattern sampling method in the literature [3], [9], [10], [11] has been applied to compact database representation. Besides, when a user changes a utility measure, reprocessing must be easy. An efficient method should not weight every transaction. Large databases slow down the processing phase. Diop et al. [7] address changing utilities like frequency, area, or decay. However, the proposed solution depends on the database number of transactions.

This paper discusses ways to speed up pattern sampling in large databases when changing utility measures. We sample patterns directly from a compressed database. So, our main goal is to come up with a generic output pattern sampling algorithm that draws patterns from a trie of occurrences based on a length-based utility measure [7].

Our main contributions are as follows.

- We introduce a new structure called *trie of occurrences* and propose TPSpace (Trie-based Pattern Space), its construction algorithm. Each node in a trie has weight information for drawing a pattern. In our case, we weight each node based on the number 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 length-based utility measure. TPSampling is generic because it takes into account any length-based utility measure.

The remainder of this paper is structured as follows. Section II situates our work in the state-of-the-art of pattern sampling and Section III highlights the challenges that must be overcome in order to achieve our goal. Our main contributions are covered in Section IV and Section V. The theoretical analyses are presented in Section VI, the experimental results in Section VII, and Section VIII concludes the paper.

II. Related works

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

A. Pattern sampling techniques

Since the first proposition of pattern sampling method [3] in 2009, numerous algorithms have been proposed for output pattern sampling [9], [10], [12], [11], [13]. Its utility has been widely demonstrated in many areas in recent years, including feature classification [9], outlier detection [4], and interactive discovery [5], [6]. It has also been used in a variety of structured data formats, including graphs [3], itemsets [9], and sequences [11]. To circumvent the long tail issue [9], Diop et al.[11] weight each pattern with a norm-based utility. In this paper, we tackle the state-of-the-art methods based on their efficiency in memory storage and flexibility on utility change.

Efficiency in memory storage: It is worth noting that all of these algorithms operate locally on a single machine. As a result, they completely use the available RAM by storing the whole data set, which might be a challenge with huge databases.

Flexibility on utility change: When the user changes the utility measure, these approaches perform a reprocessing phase that involves weighing each database transaction according to the new utility measure for the *multi-steps* ones. Diop et al.[7] demonstrate that the reprocessing phase is experimentally fast since it is proportional to the number of transactions in the database. This argument is no longer valid in large databases with many transactions.

B. Data structures for pattern mining

Many data structures have been proposed to solve the problem of pattern mining, such as the "FP-Tree" [8] or "Trie" [14]. Because multiple transactions can contain the same information, there are many repetitions in transactional databases. These repetitions make sense in this field because they allow us to discover interesting rules; however, we must first understand how to represent them. In a transactional database, for example, if 90% of transactions that contain the items $\{e1, e2, e3, e4\}$ also contain the items $\{e5, e6\}$, we might as well group them together so that they share the same prefix. This significantly reduces the database's size in memory. It is now possible to represent transactions containing the same item in a single path using "FP-Tree" or "Trie". This is due to the fact that they share the same prefix. The difference between these two structures is that, unlike "trie", which only connects a node to its children, "FP-Tree" connects nodes from different branches to quickly compute the frequency of the patterns. We propose using "trie" [14] to have a compact representation of the database in memory because we do not want to compute this latter. Trie has been used in Big Data for graph distributed computing [15].

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

III. Preliminaries and problem statement

In this section, we first present some basic concepts and definitions that are required for understanding the subject. We conclude with a formalization of the problem addressed in this paper.

A. Basic definitions

Let $\mathcal{I} = \{e_1, \cdots, e_N\}$ be a finite set of literals called items. We assume an arbitrary total order $>_\mathcal{I}$ exists between the 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 $\mathcal{D}$ 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 $\mathcal{L}(\mathcal{D})$ the set of all patterns that appear in $\mathcal{D}$ and $\mathcal{L}_{[\mu..M]}(\mathcal{D}) = \{\varphi \in \mathcal{L}(\mathcal{D}) : \mu \leq |\varphi| \leq M\}$. For example, Table I is a transactional database made up of four items, $\mathcal{I} = \{A, B, C, D\}$. In the rest of this paper, we use this database to give some illustrations.

Originally, the goal of a pattern sampling technique is to access the pattern space $\mathcal{L}(\mathcal{D})$ by an efficient sampling procedure simulating a distribution $\mathbb{P}$: $\mathcal{L}(\mathcal{D}) \rightarrow [0; 1]$ which is defined with respect to some utility measure $m : \mathbb{P}(\cdot) = m(\cdot)/Z$ where Z is a normalization constant, the sum of the utilities of all patterns $\varphi \in \mathcal{L}(\mathcal{D})$, defined by $Z = \sum_{\varphi \in \mathcal{L}(\mathcal{D})} m(\varphi, \mathcal{D})$. The selection of k patterns of $\mathcal{L}(\mathcal{D})$ according to a distribution proportional to a utility measure m may be expressed as follows:

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

where $\varphi \sim m(\mathcal{L}, \mathcal{D})$ means that φ is drawn with a probability proportional to m. Formally, $\varphi \sim m(\mathcal{L}, \mathcal{D}) \Leftrightarrow \mathbb{P}(\varphi, \mathcal{L}(\mathcal{D})) = m(\varphi, \mathcal{D})/Z$. In other words, the main objective of the output sampling methods is to get a sample of patterns that is representative of the set of patterns that can be extracted from the database.

For example, if a pattern φ_1 has a utility twice as high as that of a pattern φ_2 according to the utility measure chosen by the user, then φ_1 should be twice as likely to be in the sample as the pattern φ_2. Frequency is the most common utility measure.

Definition 1 (Frequency of a pattern): Let $\mathcal{D}$ be a database and φ be a pattern of $\mathcal{L}(\mathcal{D})$. The frequency of φ in $\mathcal{D}$ is defined as follows: $freq(\varphi, \mathcal{D}) = |\{t_i \in \mathcal{D} : \varphi \subseteq t_i\}|$.

TABLE I: A transactional database $\mathcal{D}$

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

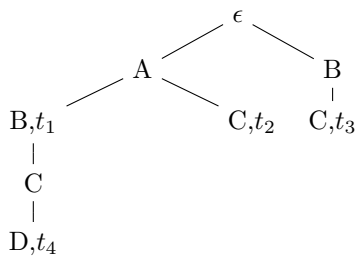

Fig. 1: Representation of a database $\mathcal{D}$ as a trie

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

Example 1: In Table I, the normalization constant, Z, is the sum of the pattern frequencies: $Z = \sum_{\varphi \in \mathcal{L}(\mathcal{D})} \mathit{freq}(\varphi, \mathcal{D}) = 24$. Within the database $\mathcal{D}$, we have $\mathit{freq}(AB, \mathcal{D}) = |\{t_1, t_4\}| = 2$ and $\mathit{freq}(AD, \mathcal{D}) = |\{t_4\}| = 1$. This means that $\mathbb{P}(AB, \mathcal{L}(\mathcal{D})) = 2/24$ and $\mathbb{P}(AD, \mathcal{L}(\mathcal{D})) = 1/24$. So, the probability to draw AB is twice that of AD in $\mathcal{D}$.

By definition, the operator $\mathcal{S}\mathit{ample}_k$ with $k > 0$ is not deterministic if $\mathcal{L}(\mathcal{D})$ has at least two patterns with probabilities that are not null. In other words, two draws with the same utility measure in the same database may not return the same k 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 [16].

Definition 2 (Length-based utility measures [16].): A utility u defined from $\mathcal{L}(\mathcal{D})$ 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 $\varphi \in \mathcal{L}(\mathcal{D})$. Given the set $\mathcal{U}$ of length-based utilities, $\mathcal{M}$ is the set of utility measures m_u such that for every pattern φ and database $\mathcal{D}$,

$$m_u(\varphi, \mathcal{D}) = \mathit{freq}(\varphi, \mathcal{D}) \times u(\varphi) \quad \mathit{with} \quad u \in \mathcal{U}.$$

For example, with the frequency, the utility function $u_{\mathit{freq}}(\varphi) = 1$ for all patterns in $\mathcal{L}(\mathcal{D})$. If we consider the utility function $u_{\mathit{area}}(\varphi) = |\varphi|$, we obtain the area measure: $\mathit{freq}(\varphi, \mathcal{D}) \times |\varphi|$. If $u_{\mathit{decay}}(\varphi) = \alpha^{|\varphi|}$, we get an exponential decay in $\alpha \in]0, 1]$. More generally, we consider the class of utility measures of the form $\mathit{freq}(\varphi, \mathcal{D}) \times u(\varphi)$ where u exclusively depends on the length of itemsets.

B. Key ideas, challenges and problem statements

We first focus on some interesting key ideas and challenges to situate our work before formulating the questions we should properly answer in order to achieve our goal.

a) 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 suggest using a trie structure to build the pattern space while making sure that reprocessing in utility change can be done in a flexible way.

Definition 3 (Trie [14]): 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 $\mathcal{D}$ as a trie in Figure 1. To do this, each node in the trie has the set of transactions in the database that end with its label.
It is important to note that many representations of the database $\mathcal{D}$ as a trie are possible depending on the insertion order of the items (decreasing order of their frequencies, ascending order, lexicographic order, etc.).

b) Challenges with trie-based pattern sampling: It is difficult to sample a pattern in the trie structure. Aside from ensuring that a pattern is drawn exactly with a probability proportional to its weight, we must also pay attention to new length-based utilities to avoid time-consuming reprocessing. In that case, the following are the primary issues that must be addressed:

- efficiently build and weights the trie that corresponds to the database,
- draw a pattern directly from the trie proportionally to its utility in the database.

c) Problem statement: These main challenges can be finally solved by answering the following questions that we formulate here.
Let $\mathcal{D}$ be a database, $u, u' \in \mathcal{U}$ two length-based utilities, and μ and M two integers such that $0 < \mu \leq M$.

1) What should be done to a (classical) trie to allow direct pattern sampling without the underlying transactional database?
2) How to draw a pattern φ from $\mathcal{L}_{[\mu..M]}(\mathcal{D})$ proportionally to $m_u(\varphi, \mathcal{D})$ directly from the weighted trie?
3) How to compute $\mathcal{S}\mathit{ampl}e_{k_1}(\mathcal{L}_{[\mu..M]}, \mathcal{D}, m_u)$ and $\mathcal{S}\mathit{ampl}e_{k_2}(\mathcal{L}_{[\mu..M]}, \mathcal{D}, m_{u'})$, with $0 < k_1 \leq k_2$, without rebuilding the trie of occurrences?

The notations of this paper are summarized in Table II.

IV. TPSpace: Trie-based Pattern Space

To define our new data structure, we need to introduce the notion of occurrence:

Definition 4 (Occurrence and language of occurrences): Let $\mathcal{D}$ be a transactional database, and μ and M two integers, $0 < \mu \leq M$. If a transaction t of identifier i in $\mathcal{D}$ contains the pattern $\varphi \in \mathcal{L}_{[\mu..M]}(\mathcal{D})$, then we denote by φ_i the occurrence of φ in t_i. The set of occurrences

TABLE II: Notations

Symbol	Definition
$\mathcal{L}_{[\mu..M]}(\mathcal{D})$	Set of patterns of $\mathcal{D}$ with lengths between the length constraints μ and M
u	Length-based utility that belongs to the set of length-based utility $\mathcal{U}$
$m_u(\varphi, \mathcal{D})$	The utility measure of the pattern φ in $\mathcal{D}$ that combines frequency and utility u
f_u	Utility function defined from $\mathbb{N}$ to $\mathbb{R}^+$ such that $u(\varphi) = f_u(\lvert\varphi\rvert)$ for all pattern φ
$\mathcal{S}amp\ell e_k(\mathcal{L}_{[\mu..M]}, \mathcal{D}, m_u)$	Set of k patterns at most drawn from $\mathcal{L}_{[\mu..M]}(\mathcal{D})$ proportionally to m_u
φ_i	Occurrence of the pattern φ in the transaction t_i
$\mathcal{L}^o_{[\mu..M]}(\mathcal{D})$	Set of occurrences of $\mathcal{D}$ with lengths between the length constraints μ and M
η	Node of the trie $\mathcal{T}$
P	Identifier of a node, it's also a prefix
$\widetilde{P}$	Label of the node η identified by P, $\widetilde{P} = \eta.label$
$\mathcal{T}_P$	Sub-trie of the trie $\mathcal{T}$ whose root is the node identified by P
$\phi^+_\ell(P, \mathcal{D})$ (resp. $\phi^-_\ell(P, \mathcal{D})$)	Set of occurrences of length ℓ with (resp. without) the item $\widetilde{P}$ in the sub-trie $\mathcal{T}_P$
$\Phi^+_\ell(P, \mathcal{D})$ (resp. $\Phi^-_\ell(P, \mathcal{D})$)	Cardinality of $\phi^+_\ell(P, \mathcal{D})$ (resp. of $\phi^-_\ell(P, \mathcal{D})$)
$\mathcal{T}.\Phi_\ell(P, \mathcal{D})$	Cardinality of the set of occurrences of length ℓ in the trie $\mathcal{T}$
$rank_\ell(\varphi_j, \mathcal{T})$	Rank of the occurrence φ_j of length ℓ in all the set of occurrences in $\mathcal{L}^o_{[\ell..\ell]}(\mathcal{D})$
$rank_{>_\mathcal{I}}(\varphi_j, \phi_1(P, \mathcal{D}))$	Rank of φ_j in $\phi_1(P, \mathcal{D})$ based on lexicographical order when $\lvert\varphi_j\rvert = 1$

in $\mathcal{D}$ under length constraints μ and M is denoted by $\mathcal{L}^o_{[\mu..M]}(\mathcal{D}) = \{\varphi_i : (\exists(\varphi, t_i) \in \mathcal{L}_{[\mu..M]}(\mathcal{D}) \times \mathcal{D})(\varphi \subseteq t_i)\}$. The length of φ_i is equal to the length of φ.

Note that $\mathcal{L}^o(\mathcal{D})$ is a set of occurrences while $\mathcal{L}(\mathcal{D})$ is a set of patterns, and, unlike a pattern, an occurrence belongs to one and only one transaction. The frequency of a pattern in $\mathcal{L}(\mathcal{D})$ is the cardinality of the set of its occurrences in $\mathcal{L}^o(\mathcal{D})$. We have then

$$\mathbb{P}(\varphi, \mathcal{L}_{[\ell..\ell]}(\mathcal{D})) = \frac{|\{\varphi_i \in \mathcal{L}^o_{[\ell..\ell]}(\mathcal{D})\}|}{|\mathcal{L}^o_{[\ell..\ell]}(\mathcal{D})|}.$$

Example 3: To draw a pattern of length 2 proportionally to its frequency in $\mathcal{D}$, it suffices to uniformly draw an occurrence in the set $\mathcal{L}^o_{[2..2]}(\mathcal{D}) = \{AB_1, AB_4, AC_2, AC_4, BC_3, BC_4, AD_4, BD_4, CD_4\}$. As a result, the probability of drawing the pattern AB in the set $\mathcal{S}$ is equal to $\mathbb{P}(AB, \mathcal{L}_{[2..2]}(\mathcal{D})) = \frac{|\{AB_1, AB_4\}|}{|\mathcal{L}^o_{[2..2]}(\mathcal{D})|} = \frac{2}{9}$.

Using a uniform drawing of occurrences, we may draw a pattern of length ℓ based on its frequency among patterns of the same length. If we can draw a length ℓ proportionally to the sum of patterns utilities of length ℓ, we can then pick a pattern from the database based on its utility.

A. Definition of a trie of occurrences

Since many representations are possible according to the order insertion, we start by defining the total order relations used in this paper before formalizing 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 database $\mathcal{D}$ is defined.

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

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

In the following we denote $>_\mathcal{I} \in \{>^{lexico}_\mathcal{I}, >^{freq}_\mathcal{I}\}$. Note that there are other types of total order relations in the literature that can be applied to literals.

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

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 $\mathcal{T}$ defined on $\mathcal{I}$ is a tree where every node $\eta \in \mathcal{T}$ except the root contains a label denoted by $\eta.label \in \mathcal{I}$, and where the root of the trie contains the label ϵ. Thus, any node $\eta \in \mathcal{T}$ can be identified by the sequence of node labels on the path from the root of $\mathcal{T}$ to the node η. If $\epsilon e_{i_1} \dots e_{i_k}$ is this sequence, we write it down more simply as $P = e_{i_1} \dots e_{i_k}$, and we denote by $\widetilde{P} = e_{i_k}$ the label of the identified node η, $\widetilde{P} = \eta.label$. For the root, we set that $P = \emptyset$.

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

Definition 7 (Sub-trie): Let $\mathcal{T}$ be a trie and P the identifier of a node. We denote by $\mathcal{T}_P$ the sub-trie of $\mathcal{T}$ whose root is the node identified by 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 $>_\mathcal{I}$, $\varphi \circ \varphi' = \varphi \cup \{e' \in \varphi' : (\forall e \in \varphi)(e >_\mathcal{I} e')\}$.

If $>_\mathcal{I}$ is the lexicographic order, then we have $B \circ AC = BC$ and $A \circ BC = ABC$. With this concatenation operator, we can define a prefix that will be used in the definition of a truncated database.

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

In the database $\mathcal{D}$ in Fig. 1, $P = AB$ 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 determine which sub-trie a pattern occurs in, we introduce the concept of a truncated database.

Definition 10 (Truncated database): Let $>_\mathcal{I}$ be a total order relation on all items, $\mathcal{D}$ be a transactional database, and P be a node identifier. A truncated database of $\mathcal{D}$ on P, denoted by $\mathcal{D}_P$, is a transactional database that holds a copy of any transaction t of $\mathcal{D}$ with prefix P minus the items of trans that appear before $\widetilde{P}$. $\mathcal{D}_P = \{(i, \widetilde{P} \circ \varphi) \in \mathbb{N} \times \mathcal{L}(\mathcal{D}) : (i,t) \in \mathcal{D} \wedge t = P \circ \varphi\}$.

Example 5: Let us consider the trie of the transactional database $\mathcal{D}$ in Fig. 1. Occurrences in the truncated database $\mathcal{D}_{AB}$ 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. The occurrences stored in the sub-trie whose root is identified by the prefix AC: C_2 are the occurrences in the truncated database $\mathcal{D}_{AC}$.

Now, by splitting these groups of occurrences by length, we will identify which of them are represented at the top level of the truncated database of $\mathcal{D}$ on the prefix P.

Definition 11 (Computing weights Φ_ℓ^- and Φ_ℓ^+): Let $\mathcal{D}$ be a transactional database and P a prefix. The set of occurrences of length ℓ of the truncated database on the prefix P is defined by: $\phi_\ell(P, \mathcal{D}) = \{(i, \varphi) \in \mathbb{N} \times \mathcal{L}(\mathcal{D}) : (i, \varphi') \in \mathcal{D}_P \wedge \varphi \subseteq \varphi' \wedge |\varphi| = \ell\}$. $\Phi_\ell(P, \mathcal{D})$ denotes the total number of occurrences of length ℓ in the truncated database $\mathcal{D}_P$: $\Phi_\ell(P, \mathcal{D}) = |\phi_\ell(P, \mathcal{D})|$. The set of occurrences $\phi_\ell(P, \mathcal{D})$ can be split into two parts:

- The set of occurrences of length ℓ of the database truncated on the prefix P and containing the item $\widetilde{P}$ is defined by:
$$\phi_\ell^+(P, \mathcal{D}) = \{(i, \varphi) \in \phi_\ell(P, \mathcal{D}) : \widetilde{P} \in \varphi\}.$$
Its cardinality is denoted by $\Phi_\ell^+(P, \mathcal{D}) = |\phi_\ell^+(P, \mathcal{D})|$.
- The set of occurrences of length ℓ of the database truncated on the prefix P without the item $\widetilde{P}$ is defined by:
$$\phi_\ell^-(P, \mathcal{D}) = \{(i, \varphi) \in \phi_\ell(P, \mathcal{D}) : \widetilde{P} \notin \varphi\}.$$
Its cardinality is denoted by $\Phi_\ell^-(P, \mathcal{D}) = |\phi_\ell^-(P, \mathcal{D})|$.

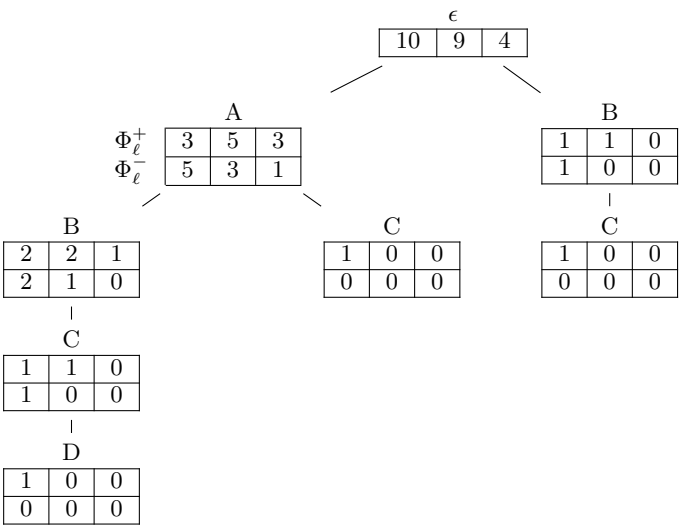

Fig. 2: Trie of occurrences of the trie provided by Fig. 1

Example 6: Given the truncated database $\mathcal{D}_A = \{(1, AB), (2, AC), (4, ABCD)\}$ on the node A, we have $\phi_2^+(A, \mathcal{D}) = \{(1, AB), (2, AC), (4, AB), (4, AC), (4, AD)\}$ and $\phi_2^-(A, \mathcal{D}) = \{(4, BC), (4, BD), (4, CD)\}$ on the other hand. Then $\Phi_2^+(A, \mathcal{D}) = 5$ and $\Phi_2^-(A, \mathcal{D}) = 3$.

At the top level of a trie $\mathcal{T}_P$, we will record the cardinalities of these sets as weight, differentiating the subsets of occurrences of length ℓ containing or not the item $\widetilde{P}$.

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

- a label denoted by $\eta.label$ belonging to $\mathcal{I} \cup \{\epsilon\}$, ϵ being the label reserved for the root.
- If η is the root then $P = \emptyset$, we have $\eta.\Phi_\ell(P, \mathcal{D}) = |\phi_\ell(P, \mathcal{D})|$.
- a list of children denoted by $\eta.child$. Subsequently, we denote $|\eta.child|$ the number of children of the node η and $\eta.child[i]$ the i-th child of η for $i \in [1..|\eta.child|]$.
- an array of positive weights defined by $\eta.\Phi_\ell^+(P, \mathcal{D})$ for $\ell \in [\mu..M]$ with P the identifier of the node η in $\mathcal{T}$, and μ and M the length constraints.
- an array of negative weights defined by $\eta.\Phi_\ell^-(P, \mathcal{D})$ for $\ell \in [\mu..M]$ with P the identifier of the node η in $\mathcal{T}$, and μ and M the length constraints.

Example 7: Let us consider the transactional database of Fig. 1, the minimum $\mu = 1$ and maximum $M = 3$ length constraints, we build the trie of occurrences according to the total order relation $>_\mathcal{I}$ in Fig. 2.

In this example, the set of labels for the children of the root is $\{A, B\}$. The number of patterns of length $\ell = 2$ in the trie $\mathcal{T}$ is equal to $\mathcal{T}.\phi_2(\emptyset, \mathcal{D}) = 9$. Let η be the node identified by $P = A$. Then we have $\eta.\phi_2^+(P, \mathcal{D}) = 5$ and $\eta.\phi_2^-(P, \mathcal{D}) = 3$ to say that the sub-trie $\mathcal{T}_A$ contains 5 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 $\widetilde{P} = A : CD_4,\ BD_4,\ BC_4$.

B. TPSpace: Algorithm for building a trie of occurrences

We describe how to generate a trie of occurrences from a transactional database to effectively handle length-based utility measures. First, note that the transactions are added to the trie iteratively. In our case, we must compute the positive and negative contributions of each transaction t into a node P identified by P, where P is a prefix of t.

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

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

By convention, $\Phi^{\star}_{\ell}(P, \mathcal{D}_0) = 0$ whatever the identifier P.

Proof 1: Omitted due to space limitation.

When adding the transaction t_i in the trie $\mathcal{T}$, 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 8: Considering Example 6 let's compute the weights $\Phi^{+}_{2}(P, \mathcal{D})$ and $\Phi^{-}_{2}(P, \mathcal{D})$, with $P = A$, using Property 1. By definition, we have $\mathcal{D}_A = \{(1, AB), (2, AC), (4, ABCD)\}$. According to Property 1, we have: $\Phi^{+}_{2}(P, \mathcal{D}_1) = 0 + \binom{|t_1|-1}{2-1} = \binom{2-1}{1} = \binom{1}{1} = 1$. Then, by adding t_2 we have $\Phi^{+}_{2}(P, \mathcal{D}_2) = \Phi^{+}_{2}(P, \mathcal{D}_1) + \binom{|t_2|-1}{2-1} = 1 + \binom{2-1}{1} = 1 + \binom{1}{1} = 1 + 1 = 2$. Adding t_3 does not affect the weights of the node identified by $P = A$ because, in this case, P is not a prefix of t_3. So we have $\Phi^{+}_{2}(P, \mathcal{D}_3) = \Phi^{+}_{2}(P, \mathcal{D}_2) = 2$. Finally, by adding t_4, we have $\Phi^{+}_{2}(P, \mathcal{D}_4) = \Phi^{+}_{2}(A, \mathcal{D}_3) + \binom{|t_4|-1}{2-1} = 2 + \binom{4-1}{1} = 2 + \binom{3}{1} = 2 + 3 = 5$. We also have $\Phi^{-}_{2}(P, \mathcal{D}_1) = 0 + \binom{|t_1|-1}{2} = \binom{2-1}{2} = \binom{1}{2} = 0$. With adding t_2 we have $\Phi^{-}_{2}(P, \mathcal{D}_2) = \Phi^{-}_{2}(P, \mathcal{D}_1) + \binom{|t_2|-1}{2} = 0 + \binom{2-1}{2} = \binom{1}{2} = 0$. Likewise, adding t_3 does not affect the weights of the node identified by $P = A$. So we have $\Phi^{-}_{2}(P, \mathcal{D}_3) = \Phi^{-}_{2}(P, \mathcal{D}_2) = 0$. Finally, by adding t_4, we get $\Phi^{-}_{2}(A, \mathcal{D}_4) = \Phi^{-}_{2}(A, \mathcal{D}_3) + \binom{|t_4|-1}{2} = 0 + \binom{4-1}{2} = \binom{3}{2} = 3$.

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

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

Algorithm 1 TPSpace

```
1: Input: A transactional database D, the minimum μ and
   maximum M length constraints
2: Output: A trie of occurrences T
3: T ← CreateNode(ϵ), i ← 0          ▷ Creation of the trie root
4: for t ∈ D do
5:     for ℓ ← μ to M do
6:         Compute c.Φ_ℓ(∅, D_i)
7:     η ← T and i ← i + 1
8:     for j ← 1 to |t| do
9:         c ← SearchChild(t[j], η) and P = t[1] ··· t[j]
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:            Compute c.Φ+_ℓ(P, D_i)
15:            Compute c.Φ−_ℓ(P, D_i)
16:        η ← c
17: return T
```

Algorithm 1 describes the TPSpace method to create a trie of occurrences of an input database $\mathcal{D}$ 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 1, we compute and add its total contribution in the trie according to the lengths (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 1. 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 $\mathcal{T}$ of the database $\mathcal{D}$.

V. TPSampling: Trie-based Pattern Sampling

This section introduces trie of occurrences basics to understand our method. Then, it presents the algorithm TPSampling to draw a pattern proportionally to a given utility measure.

A. Drawing approach

To draw a pattern of length ℓ proportionally to a length-based utility u 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 integer $\ell \in [\mu..M]$ proportionally to $\Phi_{\ell}(\emptyset, \mathcal{D}) \times f_u(\ell)$. Second, we uniformly draw an occurrence of length ℓ from $\mathcal{L}^{o}_{[\ell..\ell]}(\mathcal{D})$, but directly from the trie. More precisely, a numbering system assigns a number to each occurrence

and then, we draw a random number for selecting the occurrence.
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). The occurrences represented at the root of a sub-trie are numbered from left to right and the children of a node are ordered,
- At the top 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 $\mathcal{D}$ and $\mathcal{T}$ is a trie constructed from $\mathcal{D}$, we denote $rank_\ell(\varphi_j, \mathcal{T})$ the rank of this occurrence in $\mathcal{L}^o_{[\ell..\ell]}(\mathcal{D})$ relative to the trie $\mathcal{T}$. This rank can be defined recursively as follows.

- If $\varphi_j \in \phi^-_\ell(P, \mathcal{D})$, and more precisely if $\varphi_j \in \phi_\ell(P \circ e_i, \mathcal{D})$ where $e_i = \mathcal{T}_P.child[i]$, then $rank_\ell(\varphi_j, \mathcal{T}_P) = \sum_{k=1}^{i-1} \Phi_\ell(P \circ e_k, \mathcal{T}) + rank_\ell(\varphi_j, \mathcal{T}_{P \circ e_i})$.
- If $\varphi_j \in \phi^+_\ell(P, \mathcal{D})$, then $\varphi_j = \widetilde{P} \circ \varphi'_j$. And in this case, if $\varphi'_j \in \phi_{\ell-1}(P \circ e_i, \mathcal{D})$ where $e_i = \mathcal{T}_P.child[i]$, then $rank_\ell(\varphi_j, \mathcal{T}_P) = \Phi^-_\ell(P, \mathcal{T}) + \sum_{k=1}^{i-1} \Phi_{\ell-1}(P \circ e_k, \mathcal{T}) + rank_{\ell-1}(\varphi'_j, \mathcal{T}_{P \circ e_i})$.
- Finally, if φ_j is an occurrence of length 1, we define $rank_1(\varphi_j, \mathcal{T}_P)$ by: $rank_1(\varphi_j, \mathcal{T}_P) = rank_{>_\mathcal{I}}(\varphi_j, \phi_1(P, \mathcal{D}))$ 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 9: If we consider the patterns of length 2 and the total order relation $>^{freq}_\mathcal{I}$, the list in Table III gives the rank of each occurrence in the trie of Fig. 2:

TABLE III: 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

$\phi_2(\emptyset, \mathcal{D}) = \{CD_4, BD_4, BC_4, AD_4, AC_4, AB_4, AB_1, AC_2, BC_3\}$. We also know that $CD_4 \in \phi^-_2(A, \mathcal{D})$, which implies that $rank_2(CD_4, \mathcal{T}) = rank_2(CD_4, \mathcal{T}_A) = rank_2(CD_4, \mathcal{T}_{AB}) = rank_2(CD_4, \mathcal{T}_{ABC})$. Then we have $rank_2(CD_4, \mathcal{T}) = \Phi^-_2(ABC, \mathcal{T}) + rank_1(D, \mathcal{T}_{ABC}) = 0 + rank_{>^{freq}_\mathcal{I}}(D, \{D\}) = 1$.
Let us compute $rank_2(AB_4, \mathcal{T})$. We know that $AB_4 \in \phi^+_2(A, \mathcal{D})$, then $rank_2(AB_4, \mathcal{T}) = \Phi^-_2(A, \mathcal{T}) + 0 + rank_1(B_4, \mathcal{T}_{AB}) = 3 + rank_{>^{freq}_\mathcal{I}}(B_4, \{B_1, B_4, C_4, D_4\})$. Or $(4, D) >^{freq}_\mathcal{I} (4, C) >^{freq}_\mathcal{I} (4, B) >^{freq}_\mathcal{I} (1, B)$, then $rank_{>^{freq}_\mathcal{I}}(B_4, \{B_1, B_4, C_4, D_4\}) = 3$, which results in $rank_2(AB_4, \mathcal{T}) = 3 + 3 = 6$.

B. Trie-based pattern sampling algorithm

Algorithm 2 takes as input a trie of occurrences $\mathcal{T}$, 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 utility in the corresponding database.

Algorithm 2 TPSampling

1: **Input :** A trie $\mathcal{T}$ of occurrences of a database $\mathcal{D}$, 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 $\varphi \sim f_u(|\varphi|) \times freq(\varphi, \mathcal{D})$
3: $\varphi \leftarrow \emptyset$ and $P \leftarrow \emptyset$
4: Draw a length ℓ proportionally to $\mathcal{T}.\Phi_\ell(P, \mathcal{D}) \times f_u(\ell)$ where $\ell \in [\mu..M]$
5: Draw uniformly a rank x in $[1..\mathcal{T}.\Phi_\ell(P, \mathcal{D})]$
6: **while** $(\ell > 0)$ **do**
7: Find the i^{th} child $\eta_i \in \mathcal{T}_P.child$ such that :

$$\sum_{1 \le k < i} W_k(P \circ \eta_i.label, \mathcal{D}) < x \le \sum_{1 \le k \le i} W_k(P \circ \eta_i.label, \mathcal{D})$$

with

$$W_k(P \circ \eta_i.label, \mathcal{D}) = \sum_{\star \in \{+,-\}} \mathcal{T}_P.child[k].\Phi^\star_\ell(P \circ \eta_i.label, \mathcal{D})$$

8: $$x \leftarrow x - \sum_{1 \le k < i} W_k(P \circ \eta_i.label, \mathcal{D})$$

9: **if** $(x > \eta_i.\Phi^-_\ell(P \circ \eta_i.label, \mathcal{D}))$ **then** ▷ Check if the label of the current node is part of the pattern
10: $\varphi \leftarrow \varphi \cup \eta_i.label$
11: $x \leftarrow x - \eta_i.\Phi^-_\ell(P, \mathcal{D})$
12: $\ell \leftarrow \ell - 1$
13: $P \leftarrow P \circ \eta_i.label$
14: **return** φ

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, $\mathcal{T}.\Phi_\ell(P, \mathcal{D})$, 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 $[1..\mathcal{T}.\Phi_\ell(P, \mathcal{D})]$ (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 $\mathcal{T}$. 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.

VI. Theoretical analysis

This section examines our trie sampling strategy in terms of soundness and complexity (memory storage and temporal). Property 2 shows that our sampling method TPSampling does an exact draw of a pattern.

Property 2 (Soundness): Let $\mathcal{T}$ a trie of occurrences from a transactional database and u a length-based utility, Algorithm 2 draws a pattern φ proportionally to its frequency weighted by its length-based utility.

Proof 2: Omitted due to space limitation.

A. Space complexity

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. Furthermore, a tight upper bound of the number of nodes is detailed in [17].

B. Time complexity

The time complexity of our method can be divided into three phases: preprocessing time to build the trie of occurrences, re-preprocessing times in utility change, and drawing time of an occurrence.

a) 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 $\mathcal{D}$ and to compute their frequencies in $O(||\mathcal{D}||)$ where $||\mathcal{D}||$ is the sum of the lengths transactions from the database $\mathcal{D}$. The previously retrieved items are ordered according to the chosen relation $>_{\mathcal{I}}$ in $O(|\mathcal{I}| \times \log(|\mathcal{I}|))$. Then, before adding a transaction to the trie, we order its items in $O(T_{\max} \times \log(T_{\max}))$ where $T_{\max}$ is the maximum length of transactions in the database $\mathcal{D}$. Finally, let z be the total number of nodes in the trie, μ and M the minimum and maximum length constraints respectively, then the weighting of the nodes is done in $O(z \times 2 \times (M - \mu))$. Thus, the total complexity for building the trie of occurrence of the transactional database $\mathcal{D}$ built on the set of literals $\mathcal{I}$ is in $O(||\mathcal{D}|| + |\mathcal{I}| \times \log(|\mathcal{I}|) + |\mathcal{D}| \times T_{\max} \times \log(T_{\max}) + z \times (M - \mu))$.

b) Reprocessing time in 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 tiny. This is because only the array of the root of the trie is traversed to compute the new weight of each length $\ell \in [\mu..M]$.

c) 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 [9] (with length constraints) which draws a sample of k patterns in $O(k \times T_{\max} \times \log(|\mathcal{D}|))$.

VII. Experiments

This experimental section aims to assess the efficiency of our approach to large transactional databases. The experiments were conducted with 2 UCI databases `Susy` and `USCensus`, and 2 synthetic databases built with the IBM-Generator[1] `T10I4D2000K` and `T10I6D3000K`. Table IV provides benchmarks by number of items, transactions, and maximum and average transaction length. It shows the number of trie nodes for each database and according to the total order relation. 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. The source code is available at https://github.com/TPSampling/TPSampling. We also implement an approach of Two-Step proposed by Boley et al.[9] under length constraints as a baseline.

TABLE IV: Characteristics of databases

$\mathcal{D}$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert t\rvert_{\min}$	$\lvert t\rvert_{\max}$	$\lvert t\rvert_{avg}$
`USCensus`	396	1 M	25	68	68.00
`Susy`	190	5 M	19	19	19.00
`T10I4D2000K`	2,719	2 M	10	30	20.11
`T10I6D3000K`	3,952	3 M	10	35	22.61

A. 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. According to the gain obtained in the last column of Table VI, the number of nodes is substantially lower with the $>_{\mathcal{I}}^{freq}$ relation than with the $>_{\mathcal{I}}^{lexico}$ relation. As a result, the fewer the nodes, the lower the storage cost. Due to an "Out of memory" issue, it is not feasible to perform TPSampling with the lexicographical order in the last two databases.

Fig. 3 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. These experimental results show that our approach is sensitive to the total order relation. For instance, TPSampling $+>_{\mathcal{I}}^{freq}$ returns an "Out of memory"

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

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

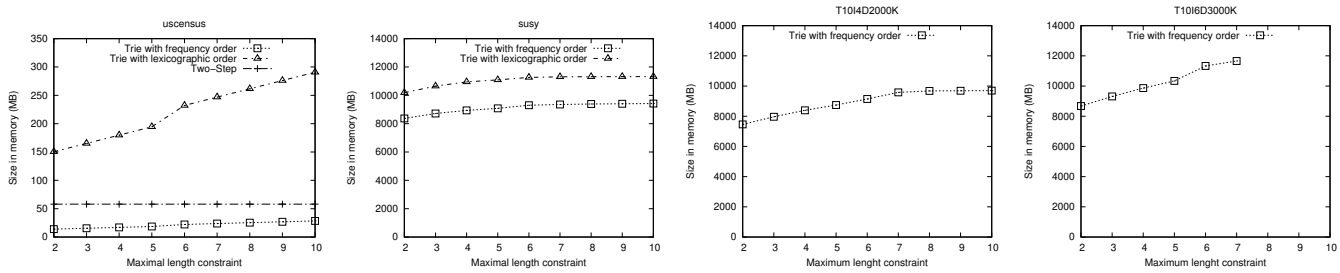

Fig. 3: Evolution of the memory size of the tries according to the length constraint

TABLE V: Preprocessing times in seconds according to the measures and ($M \in \{2, 6\}$)

		Two-Step		TPSampling ($>_{\mathcal{I}}^{freq}$)		TPSampling ($>_{\mathcal{I}}^{lexico}$)	
$\mathcal{D}$	m	M=2	M=6	M=2	M=6	M=2	M=6
USCensus	Area	0.01 ± 0.00	0.01 ± 0.00	5.26 ± 1.85	7.39 ± 0.17	8.09 ± 0.32	12.27 ± 1.27
	Decay	0.01 ± 0.00	0.01 ± 0.00				
	Freq	0.01 ± 0.00	0.01 ± 0.00				
Susy	Area	–	–	347.99 ± 8.01	593.01 ± 44.24	532.01 ± 41.65	910.45 ± 65.47
	Decay	–	–				
	Freq	–	–				
T10I4D2000K	Area	–	–	356.54 ± 9.31	418.87 ± 24.13	–	–
	Decay	–	–				
	Freq	–	–				
T10I6D3000K	Area	–	–	792.56 ± 45.21	1094.49 ± 51.42	–	–
	Decay	–	–				
	Freq	–	–				

TABLE VI: Characteristics of tries of the databases

	Nb of nodes in the trie		Gain (in %)
	$>_{\mathcal{I}}^{freq}$	$>_{\mathcal{I}}^{lexico}$	$\frac{Nb_{lexico}-Nb_{freq}}{Nb_{lexico}}$
USCensus	312,808	607,611	48.52
Susy	10,424,240	12,630,372	17.47
T10I4D2000K	9,957,321	–	–
T10I6D3000K	10,617,309	–	–

exception with T10I6D3000K when the maximum length constraint is greater than 7. We also found that the trie of occurrences created according to $>_{\mathcal{I}}^{freq}$ used less memory than the Two-Step database representation. The later 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.

B. Speed of the approach

This section analyses preprocessing, reprocessing, and pattern draw of our approach.

Evaluation of the preprocessing time. Interestingly, the time to build a trie of occurrences is independent of any length-based utility measure. In our experiments, we consider the maximum length constraints $M \in \{2, 6\}$. Table V presents the preprocessing times to build the tries of occurrences and those of Two-Step according to the maximum length constraint. Each experiment is repeated 10 times to have the average preprocessing times and the standard deviations.

Because it only requires one pass through the database, Two-Step is faster than TPSpace in preprocessing. However, on Susy, T10I4D2000K, and T10I6D3000K, it throws a "Out of memory" exception, but TPSampling+ $>_{\mathcal{I}}^{freq}$ lasts on average 18 minutes with the maximum length constraints $M = 6$ in T10I6D3000K. Interestingly, we only do this preprocessing once, after which we may utilise the resulting trie of occurrences with any length-based utility.

Evaluation of the reprocessing time for utility change. The reprocessing time, when utility changes, depends linearly to the difference between the minimum and the maximum length constraints only. Utility measures like frequency, area, and exponential decay have not a notorious impact on the speed of the reprocessing phase. Contrariwise, Two-Step should do a new preprocessing from scratch when utility changes. Experiments show that in the reprocessing phase our approach needs a few time, less than 10×10^{-6} seconds with $M = 10$ while Two-Step needs 0.45 seconds with USCensus. Interestingly, the reprocessing time is the same for all databases with the same length constraints. These results show the importance of the trie when changing the length-based utility measure.

Evaluation of the drawing time per pattern. We test the speed of our approach on the 4 databases and figure out the average drawing time of a pattern with a maximum length constraint in [1..10] and an exponential

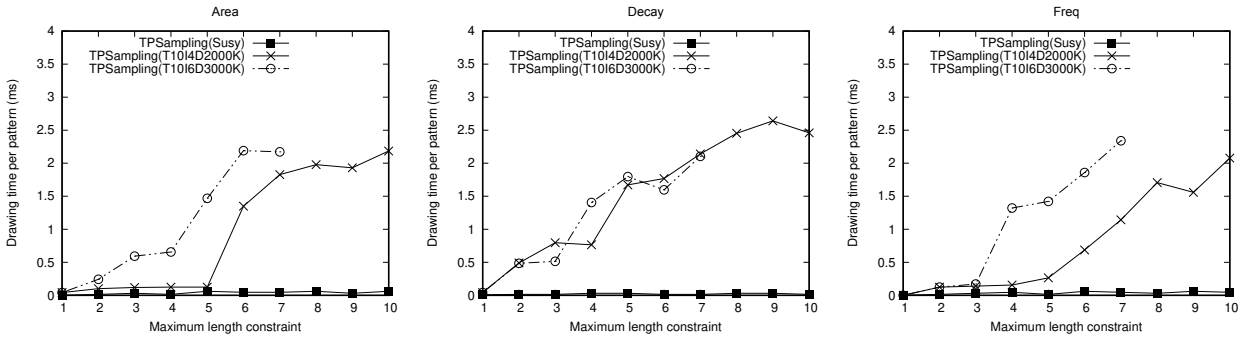

Fig. 4: Evolution of the average of drawing time per pattern according to M ($\alpha = 0.1$ for the exponential decay)

decay $\alpha = 0.1$. Fig. 4 compares the average drawing time of a pattern using TPSampling with length constraints. For $M \in [1..10]$, we repeat 100 times the draws of a pattern. Standard deviations are omitted as they are tiny. With large databases like `Susy`, `T10I4D2000K`, and `T10I6D3000K`, TPSampling can draw a pattern in as little as 2.5 milliseconds while $M \in [1..7]$. When M is more than 7, TPSampling throws a "Out of memory" issue with the `T10I6D3000K` database. We see that the drawing times of TPSampling+Area and TPSampling+Freq increase practically identically. TPSampling is efficient since it produces thousands of patterns per second.

VIII. Conclusion

This paper proposed a generic trie-based output pattern sampling method using two efficient algorithms. TPSampling samples patterns based on any length-based utility measure, using a trie of occurrence built by TPSpace. After building a trie of occurrences with fixed length constraints, the user can draw patterns with frequency, area, and exponential decay $\alpha \in]0, 1]$. The experiments also show that our approach is very flexible on utility change and works well with large transactional databases thanks to the prefix-based compression.

We hope to parallelize our method in the future by adapting the BSP-based framework proposed by Diop and Ba [18]. In such case, the trie might be spread over many machines to parallelize the computation in the preprocessing phase, as well as the drawing and reprocessing phases.

Acknowledgements. This work has been partly supported by the CEA-MITIC (African Center of Excellence in mathematics, IT and ICT).

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] 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.

[5] M. van Leeuwen, *Interactive Data Exploration Using Pattern Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 169–182.

[6] 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.

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

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

[9] 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.

[10] 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.

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

[13] L. Diop, "High average-utility itemset sampling under length constraints," in *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 134–148.

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

[15] L. Xiang, A. Khan, E. Serra, M. Halappanavar, and A. Sukumaran-Rajam, "cuts: scaling subgraph isomorphism on distributed multi-gpu systems using trie based data structure," in *SC '21*. ACM, 2021, pp. 69:1–69:14.

[16] 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.

[17] 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.

[18] L. Diop and C. Ba, "Parallelization of sequential pattern sampling," *2021 IEEE International Conference on Big Data (Big Data)*, pp. 5882–5884, 2021.