Pattern Sampling in Distributed Databases

Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Arnaud Soulet

Pattern Sampling in Distributed Databases

Lamine Diop[1,2], Cheikh Talibouya Diop[2], Arnaud Giacometti[1], and Arnaud Soulet[1]

[1]Université de Tours, Tours, France
{arnaud.giacometti, arnaud.soulet}@univ-tours.fr
[2]Université Gaston Berger de Saint-Louis, Saint-Louis, Sénégal
{diop.lamine3,cheikh-talibouya.diop}@ugb.edu.sn

Abstract. Many applications rely on distributed databases. However, only few discovery methods exist to extract patterns without centralizing the data. In fact, this centralization is often less expensive than the communication of extracted patterns from the different nodes. To circumvent this difficulty, this paper revisits the problem of pattern mining in distributed databases by benefiting from pattern sampling. Specifically, we propose the algorithm DDSampling that randomly draws a pattern from a distributed database with a probability proportional to its interest. We demonstrate the soundness of DDSampling and analyze its time complexity. Finally, experiments on benchmark datasets highlight its low communication cost and its robustness. We also illustrate its interest on real-world data from the Semantic Web for detecting outlier entities in DBpedia and Wikidata.

1 Introduction

Many applications require storage and manipulation of distributed databases [14] like large-scale wireless sensor networks [15] or the Semantic Web [2]. In most cases, the centralization of data is very costly, in particular when the databases evolve continuously. Sometimes legal constraints also prevent this centralization [6]. Thus, [16] underlines the importance of extending knowledge discovery to distributed databases. In the context of the Semantic Web, Table 1 illustrates an example of Resource Description Framework (RDF) data distributed over four triplestores $\mathcal{P} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$ accessible via SPARQL queries. In this context, the properties describing the entity identified by TId 1 (e.g., the singer "Youssou N'Dour") are spread over several fragments (i.e., *DBpedia* $\mathcal{D}_1$ with the property A, and *Wikidata* $\mathcal{D}_2$ with the properties B and C). There exist federated systems to execute SPARQL queries on multiple triplestores [9]. Unfortunately, SPARQL is not expressive enough to directly extract patterns like frequent itemsets. By relying on a basic communication model, this paper aims at extracting patterns from a distributed database $\mathcal{P}$ (including RDF data) as if the data were centralized (see $\mathcal{P}^*$ in Table 1).

Few works in the literature [4,13,10,11] are dedicated to pattern mining in distributed databases. Unfortunately, they suffer from three major limitations. First, they exclusively address horizontally partitioned data (i.e, unlike the example $\mathcal{P}$ in Table 1, a transaction cannot be split into two fragments). Second,

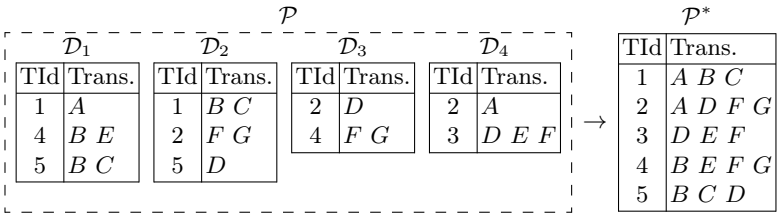

Table 1: Example of a distributed database $\mathcal{P} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$

as these proposals focused on an *exhaustive extraction* of patterns, the volume of data exchanged between the different fragments can be very high. Finally, these proposals need computing capacity on each fragment for locally mining patterns, which is often impossible (for example, in the case of the SPARQL endpoints of the Semantic Web). To overcome these limitations, we propose to benefit from *pattern sampling* [1,3]. Pattern sampling consists in randomly drawing a collection of patterns with a probability proportional to their interest. This technique has a low computational cost, but it is also useful in many tasks such as classification [3], outlier detection [8] or interactive data mining [7].

In this paper, we show how to sample patterns from a distributed database that can be partitioned both horizontally and vertically, without using the computing capacity of the different fragments. Our main contributions are as follows:

- We propose a generic algorithm called Distributed Database Sampling (DDSampling) which randomly draws a pattern from a distributed database proportionally to an interestingness measure combining frequency and length-based utility functions (including length constraints).
- We demonstrate that DDSampling performs an exact sampling and analyze its complexity on average. Experiments show that DDSampling is very fast and that the communication cost of our proposal is much lower than that of data centralization for drawing a few thousand patterns. We also show that in the context of distributed databases, DDSampling is a fault-resistant algorithm against network and node failures.
- We illustrate the interest of DDSampling on a use case by detecting outliers in two real-world triplestores: *DBpedia* and *Wikidata*. These experiments show the importance of using a maximum length constraint and that *output space* sampling is more efficient than *input space* sampling.

This paper is structured as follows. After the related work in Section 2, Section 3 introduces basic definitions and formalizes the problem of pattern sampling in distributed databases. We detail the algorithm DDSampling in Section 4 that exactly draws patterns as if the distributed database was centralized. Section 5 evaluates its performance on benchmarks datasets and illustrates the interest of pattern sampling to detect outliers in the Semantic Web.

2 Related Work

Several approaches in the literature focused on frequent pattern mining in distributed databases. This task is complex because whatever the minimum frequency threshold user-specified on the distributed database (global frequency), it is not possible to constrain the local frequency on each fragment without communicating information between sites. In this context, [4] proposes the first method to extract all the globally frequent patterns by identifying the sites where the patterns are the most frequent and thus, reducing communication costs. More drastically, [13] proposes to save communication costs by limiting themselves to the collection of the maximal frequent patterns. To prevent each fragment from enumerating all its patterns, [10] imposes a minimum local frequency threshold on each fragment. From the different local extractions, [11] builds an approximate global collection of frequent patterns. A centralized pruning proposed by [18] is based on the construction of a tree containing for each pattern all its occurrences (i.e., fragment/transaction pairs), which still requires a considerable volume of communications. More recently, [17] implements a decentralized pruning technique within the extraction on each fragment by exchanging Bloom filters. This approach significantly reduces computation time but the cost of communications remains too large. Indeed, for low support threshold, the volume of extracted patterns invariably generates an enormous communication cost much higher than that of data centralization. In addition, all these frequent pattern mining approaches are limited to horizontal partitioning of data, i.e. the same transaction cannot be distributed on two separate fragments. Finally, all the existing proposals require a computation capacity on each fragment, which is not always possible. For instance, the Semantic Web provides access to distributed data via SPARQL endpoints, but it is not possible to execute a pattern mining routine on these endpoints. For all these reasons, in this paper, we revisit the discovery of patterns in distributed databases in the light of pattern sampling. We will see that our approach does not require computation capacity on the fragments and it reduces the communication costs because all the patterns are not extracted.

Output space sampling methods [1,3] aim at drawing patterns with a probability distribution proportional to their interest. Most sampling techniques fall into two broad categories: stochastic methods [1] and multi-step methods [3]. In order to randomly walk from a pattern X to another, stochastic methods require to consider the global interest of all the neighboring patterns of X. For example, in the case of frequency, it would be necessary to know the global frequency of all the subsets and supersets of X, which would generate many communications. For this reason, we prefer to adopt a multi-step random method. This type of method has already been used for several interestingness measures (e.g., support or area [3] or exceptional measure [12]) and several data types like sequential data [5]. Nevertheless, the context of distributed databases is an orthogonal challenge. In particular, we determine minimal information that the fragments must communicate to make an unbiased draw of patterns.

3 Problem Formulation

3.1 Pattern language and distributed databases

Given a set $\mathcal{I}$ of distinct literals called *items*, an itemset (or pattern) is a subset of $\mathcal{I}$. The language of itemsets corresponds to $\mathcal{L} = 2^{\mathcal{I}}$, and the size or length of an itemset $X \in \mathcal{L}$, denoted by $|X|$, is its cardinality. In our approach, a transactional database $\mathcal{D}$ is a set of pairs (j, X) where j is the unique identifier of a transaction and X is an itemset in $\mathcal{L}$, i.e. $\mathcal{D} \subseteq \mathbb{N} \times \mathcal{L}$. In the following, given a transactional database $\mathcal{D}$, for every integer $j \in \mathbb{N}$, $\mathcal{D}[j]$ represents the itemset of transaction j, i.e. $\mathcal{D}[j] = X$ if $(j, X) \in \mathcal{D}$ (otherwise, we consider that $\mathcal{D}[j] = \emptyset$). Moreover, $|\mathcal{D}|$ is the number of transactions in $\mathcal{D}$ and $||\mathcal{D}|| = \sum_{j \in \mathbb{N}} |\mathcal{D}[j]|$ defines the size of the transactional database $\mathcal{D}$. For example, in Table 1, the transactional database $\mathcal{D}_1$ contains 3 transactions of identifiers 1, 4 and 5. Besides, we have $\mathcal{D}_1[1] = A$, $\mathcal{D}_1[4] = BE$ and $\mathcal{D}_1[5] = BC$. Thus, we have $||\mathcal{D}_1|| = 1 + 2 + 2 = 5$.

Intuitively, a *(transactional) distributed database* is a set of transactional databases, also called *fragments*, where transactions do not overlap. More formally, a distributed database is defined as follows:

Definition 1 (Transactional distributed and centralized databases). *A* (transactional) distributed database $\mathcal{P} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ *is a set of transactional databases* $\mathcal{D}_k$ *(*$k \in [1..K]$*) such that for each* $j \in \mathbb{N}$*, we have* $\mathcal{D}_k[j] \cap \mathcal{D}_l[j] = \emptyset$ *if* $k \neq l$*. Then, the* centralized version *of* $\mathcal{P}$*, denoted by* $\mathcal{P}^*$*, is the transactional database defined by:* $\mathcal{P}^* = \{(j, X) \ : \ X = \bigcup_{k=1}^{K} \mathcal{D}_k[j] \wedge X \neq \emptyset\}$.

For example, in Table 1, it is easy to see that $\mathcal{P}^*$ is the centralized version of the distributed database $\mathcal{P} = \{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$. In the following, we also say that $\mathcal{P}$ is a partitioning of the centralized database $\mathcal{P}^*$. In general, different types of partitioning are distinguished. On the one hand, a distributed database is a *horizontal partitioning* if every transaction is described in only one fragment, i.e. if for every $j \in \mathbb{N}$, $\mathcal{D}_k[j] \neq \emptyset$ and $\mathcal{D}_l[j] \neq \emptyset$ implies that $k = l$. On the other hand, a distributed database is a *vertical partitioning* if every item is present in only one fragment, i.e. if for every $x \in \mathcal{I}$, $x \in \bigcup_{j \in \mathbb{N}} \mathcal{D}_k[j]$ and $x \in \bigcup_{j \in \mathbb{N}} \mathcal{D}_l[j]$ implies that $k = l$. Finally, a partitioning is said to be *hybrid* if it is neither *horizontal* nor *vertical*. For example, in Table 1, $\mathcal{P}$ is a hybrid partitioning of $\mathcal{P}^*$. Indeed, the transaction 2 is described both in the fragments $\mathcal{D}_2$, $\mathcal{D}_3$ et $\mathcal{D}_4$ and the item A is both present in fragments $\mathcal{D}_1$ and $\mathcal{D}_4$.

We consider only two forms of query can be sent to the fragment:

1. `lengthOf` **primitive:** Given a transaction identifier j and a fragment $\mathcal{D}_k$, the query $\texttt{lengthOf}(j, \mathcal{D}_k)$ returns the length of the transaction j in fragment k, i.e. $\texttt{lengthOf}(j, \mathcal{D}_k) = |\mathcal{D}_k[j]|$. In our example, we have $\texttt{lengthOf}(4, \mathcal{D}_1) = 2$ since $|\mathcal{D}_1[4]| = |BE| = 2$.
2. `itemAt` **primitive:** Given a position i, a transaction j and a fragment $\mathcal{D}_k$, the query $\texttt{itemAt}(i, j, \mathcal{D}_k)$ returns the i-th item of the transaction j in fragment $\mathcal{D}_k$ (assuming an arbitrary order over the set of items $\mathcal{I}$). In our example, considering the lexicographic order over $\mathcal{I}$, we have $\texttt{itemAt}(2, 4, \mathcal{D}_1) = E$.

Our communication model is generic since more complex queries can be reduced to these two basic primitives for describing the exchanges. For instance, advanced queries may directly obtain a transaction of ℓ items from a fragment corresponding to ℓ `itemAt` primitives. For the Semantic Web, for the primitive `lengthOf`, the SPARQL query `SELECT (COUNT(DISTINCT ?p) AS ?length) WHERE {wd:Q210734 ?p ?o}` returns the length of Youssou N'Dour's transaction (`wd:Q210734`) from *Wikidata* SPARQL endpoint (here, the length is 143 items). In the same way, the query `SELECT DISTINCT ?p WHERE {wd:Q210734 ?p ?o} OFFSET 2 LIMIT 1` gives the second item of the transaction. Of course, it is possible to use the query `SELECT DISTINCT ?p WHERE {wd:Q210734 ?p ?o}` for having the entire transaction instead of using 143 `itemAt` queries. Unlike the two primitives that have 1 as communication cost, the communication cost of this query is 143.

3.2 Class of interestingness measures

Pattern discovery is based on interestingness measures that evaluate the quality of a pattern. One of the most popular of interestingness measures is the frequency which is an intuitive interestingness measure for experts and is an essential atomic element to build many other interestingness measures (like area or discriminative measures). The frequency of an itemset $X \in \mathcal{L}$ in the transactional database $\mathcal{D}$, denoted by $freq(X, \mathcal{D})$, is defined by: $freq(X, \mathcal{D}) = |\{(j, T) \in \mathcal{D} \ : \ X \subseteq T\}|$. In Table 1, we have $freq(DF, \mathcal{P}^*) = 2$ since the itemset DF is included in transactions 2 and 3.

It is also common to associate a utility to an itemset, and to combine the frequency of an itemset with its utility. For example, if we consider the utility function $u(X) = |X|$, we obtain the area measure: $freq(X, \mathcal{D}) \times |X|$. More generally, we consider the class of interestingness measures of the form $freq(X, \mathcal{D}) \times u(X)$ where u exclusively depends on the length of itemsets:

Definition 2 (Length-based utilities and 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 $u(X) = f_u(|X|)$ for each $X \in \mathcal{L}$. Given the set $\mathcal{U}$ of length-based utilities, $\mathcal{M}_\mathcal{U}$ is the set of interestingness measures m_u such that for every pattern X and database $\mathcal{D}$, $m_u(X, \mathcal{D}) = freq(X, \mathcal{D}) \times u(X)$ with $u \in \mathcal{U}$.*

We already see that the utility function defined for every pattern $X \in \mathcal{L}$ by $u_{area}(X) = |X|$ allows us to consider the area measure $freq(X, \mathcal{D}) \times |X|$. Obviously, let us notice that the utility function defined by $u_{freq}(X) = 1$ enables us to consider the frequency as interestingness measure. Besides, the utility function defined by $u_{\leq M}(X) = 1$ iff $|X| \leq M$ (0 otherwise) simulates a maximum length constraint. Indeed, with the induced interestingness measure $freq(X, \mathcal{D}) \times u_{\leq M}(X)$, an itemset with a cardinality strictly greater than M is judged useless (whatever its frequency). Dually, the utility function defined by $u_{\geq m}(X) = 1$ iff $|X| \geq m$ (0 otherwise) simulates a minimum length constraint. Finally, the utility function defined by $u_{decay}(X) = \alpha^{|X|}$ with $\alpha \in]0, 1[$, named exponential decay, is useful for penalizing large itemsets but in a smooth way in comparison with $u_{\leq M}$.

3.3 Pattern sampling in a distributed database

A pattern sampling method aims at randomly drawing a pattern X from a language $\mathcal{L}$ according to an interestingness measure f. $X \sim \pi(\mathcal{L})$ denotes such a pattern where $\pi(.) = f(.)/Z$ is a probability distribution over $\mathcal{L}$ (with Z as normalizing constant). In this paper, our goal is to randomly draw patterns in a distributed database according to an interestingness measure in $\mathcal{M}_\mathcal{U}$:

Given a distributed database $\mathcal{P}$, an interestingness measure $m \in \mathcal{M}_\mathcal{U}$, we aim at randomly drawing a pattern $X \in \mathcal{L}$ with a probability distribution π proportional to its interestingness measure m i.e., $\pi(X) = \frac{m(X,\mathcal{P}^*)}{Z}$ where $\mathcal{P}^*$ is the centralized version of $\mathcal{P}$ and $Z = \sum_{X \in \mathcal{L}} m(X, \mathcal{P}^*)$ is a normalizing constant.

A naive approach could apply the classical two-step random procedure [3] after having centralized all the fragments of the distributed database $\mathcal{P}$ for building $\mathcal{P}^*$ (using `itemAt` queries). The communication cost of this preliminary centralization would be very high. Indeed, it would be proportional to the size of $\mathcal{P}^*$, i.e. $||\mathcal{P}^*|| = \sum_{k=1}^{K} ||\mathcal{D}_k||$. Next section shows that it is only necessary to centralize the lengths of the transaction parts stored in the different fragments of $\mathcal{P}$ in order to draw an exact sample of patterns.

4 Decentralized Pattern Sampling

4.1 DDSampling algorithm

This section presents our algorithm called DDSampling (for Distributed Database Sampling), which randomly draws a pattern from a distributed database $\mathcal{P}$ proportionally to an interestingness measure $m \in \mathcal{M}_\mathcal{U}$.

The key idea of our proposal is first to centralize only the lengths of the transaction parts contained in the different fragments. Indeed, this information requires a low communication cost and it enables us to draw a transaction identifier j according to its weight $\omega(j)$ and an itemset length ℓ proportionally to $\omega_\ell(j)$. Finally, we show how to emulate a decentralized sampling of a subset of $\mathcal{D}[j]$ of length ℓ without centralizing all the items of $\mathcal{D}[j]$.

Preprocessing phase. In this phase (see lines 1-2 of Algorithm 1), we first compute and store locally a matrix M that contains for every transaction j and every fragment $\mathcal{D}_k$ of $\mathcal{P}$, the length of the transaction j in $\mathcal{D}_k$.

Definition 3 (Weight matrix). *Given a distributed database $\mathcal{P} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$, let $M \in \mathbb{R}^{|\mathcal{P}^*| \times |\mathcal{P}|}$ be the matrix defined for every $j \in [1..|\mathcal{P}^*|]$ and $k \in [1..K]$ by $M_{jk} = \texttt{lengthOf}(j, \mathcal{D}_k)$.*

In practice, it is important to note that the matrix M is computed offline before the drawing phase. In the following, $M_{j\bullet}$ denotes the sum $M_{j\bullet} = \sum_{k=1}^{K} M_{jk}$. It is easy to see that $M_{j\bullet}$ represents the length of the transaction j in $\mathcal{P}^*$, i.e. $M_{j\bullet} = |\mathcal{P}^*[j]|$. For example, Table 2 presents the weight matrix M of the distributed database of Table 1. We can also check that $M_{1\bullet} = 1 + 2 = |\mathcal{P}^*[1]|$.

j	M	$M_{j\bullet}$	$\sum_{\ell=0}^{M_{j\bullet}} \omega_\ell^{freq}(j) =$	$\omega^{freq}(j)$	$\omega^{area}(j)$	$\omega^{\leqslant 2}(j)$	$\omega^{decay}(j)$
1	1 2 0 0	3	$1 + 3 + 3 + 1 =$	8	12	7	1.331
2	0 2 1 1	4	$1 + 6 + 4 + 4 + 1 =$	16	32	11	1.6441
3	0 0 0 3	3	$1 + 3 + 3 + 1 =$	8	12	7	1.331
4	2 0 2 0	4	$1 + 6 + 4 + 4 + 1 =$	16	32	11	1.6441
5	2 1 0 0	3	$1 + 3 + 3 + 1 =$	8	12	7	1.331

Table 2: Weight matrix M and transaction drawing weights

Algorithm 1 DDSampling

Input: A distributed database $\mathcal{P} = \{\mathcal{D}_1, \cdots, \mathcal{D}_K\}$ and a length-based utility $u \in \mathcal{U}$
Output: An itemset $X \in \mathcal{L}$ randomly drawn w.r.t. $freq(X, \mathcal{P}^*) \times u(X)$

// **Preprocessing Phase**
Compute M defined by $M_{jk} := \texttt{lengthOf}(j, \mathcal{D}_k)$ for $j \in [1..|\mathcal{P}^*|]$ and $k \in [1..K]$
Compute the weights ω defined by $\omega(j) := \sum_{\ell=0}^{M_{j\bullet}} \binom{M_{j\bullet}}{\ell} \times f_u(\ell)$ for $j \in [1..|\mathcal{P}^*|]$
// **Drawing Phase – Step 1***: sampling of a transaction*
Draw a transaction identifier $j \in [1..|\mathcal{P}^*|]$ proportionally to ω: $j \sim \omega(\mathcal{P}^*)$
// **Drawing Phase – Step 2***: decentralized sampling of an itemset*
Compute the weights defined by $\omega_\ell(j) := \binom{M_{j\bullet}}{\ell} \times f_u(\ell)$ for every $\ell \in [0..M_{j\bullet}]$
Draw a length ℓ proportionally to $\omega_\ell(j)$: $\ell \sim \omega_{[0..M_{j\bullet}]}(j)$
$\vartheta := \emptyset$ and $X := \emptyset$
while $|X| < \ell$ **do**
  $i \sim u([1..M_{j\bullet}] \setminus \vartheta)$
  $k := \min\{p \in [1..K] \ : \ i \leq \sum_{m=1}^{p} M_{jm}\}$
  $i' := \sum_{m=1}^{k} M_{jm} - i + 1$ and $x := \texttt{itemAt}(i', j, \mathcal{D}_k)$
  $X := X \cup \{x\}$
  $\vartheta := \vartheta \cup \{i\}$
od
return X

Given a distributed database $\mathcal{P}$ and its associated weight matrix M, Property 1 shows how the weights $\omega(j)$ and $\omega_\ell(j)$ can be computed for each transaction j and length ℓ for any length-based utility function u:

Property 1. Let $\mathcal{P} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ be a distributed database and $u \in \mathcal{U}$ a length-based utility. Given the weight matrix M associated with $\mathcal{P}$, for each transaction $j \in [1..|\mathcal{P}^*|]$ and $\ell \in [1..M_{j\bullet}]$, we have: $\omega_\ell(j) = \sum_{X \subseteq \mathcal{P}^*[j] \wedge |X|=\ell} u(X) = \binom{M_{j\bullet}}{\ell} \times f_u(\ell)$. Moreover, we have $\omega(j) = \sum_{\ell=0}^{M_{j\bullet}} \omega_\ell(j)$.

Due to space limitation, the proofs have been omitted. Intuitively, this property is valid because all the itemsets of length ℓ in a transaction j have the same utility. In Algorithm 1, this property is useful during the preprocessing phase to compute the weights of all the transactions. This preprocessing phase is illustrated in Table 2 with four length-based utility functions: u_{freq}, u_{area}, $u_{\leq 2}$ and u_{decay} (with $\alpha = 0.1$). For example, because $u_{freq}(X) = 1$ for every $X \in \mathcal{L}$, we have $\omega^{freq}(1) = \sum_{\ell=0}^{3} \binom{3}{\ell} = 1 + 3 + 3 + 1 = 8 = 2^3$. Con-

sidering the area utility function u_{area}, we have $\omega^{area}(1) = \sum_{\ell=0}^{3} \binom{3}{\ell} \times \ell = (1 \cdot 0) + (3 \cdot 1) + (3 \cdot 2) + (1 \cdot 3) = 3 + 6 + 3 = 12$ since $u_{area}(X) = \ell$ for every pattern X of length ℓ. With the maximum length constraint, it is easy to see that $\omega^{\leqslant 2}(1) = \sum_{\ell=0}^{2} \binom{3}{\ell} = 1 + 3 + 3 = 7$. Finally, with the decay utility function u_{decay} and $\alpha = 0.1$, we have $\omega^{decay}(1) = \sum_{\ell=0}^{3} \binom{3}{\ell} \times 0.1^{\ell} = (1 \cdot 0.1^0) + (3 \cdot 0.1^1) + (3 \cdot 0.1^2) + (1 \cdot 0.1^3) = 1 + 0.3 + 0.03 + 0.001 = 1.331$.

Drawing phase. In this phase, we can apply a direct generalization of the two-step random procedure proposed in [3] to draw itemsets with a probability proportional to their interest in the manner of the area measure. We start by drawing in Step 1 a transaction identifier j with a probability proportional to its weight $\omega(j)$ (see line 3 of Algorithm 1). The only difference is that the weights $\omega(j)$ are computed during the preprocessing phase using the weight matrix M and Property 1.

In Step 2, as the weights $\omega_\ell(j)$ of the transaction j are not stored during the preprocessing phase to reduce the storage cost, line 4 computes them for any length ℓ. After drawing the length ℓ of the itemset that will be returned (lines 4-5), DDSampling draws an itemset of length ℓ from the different fragments of $\mathcal{P}$. At each iteration of the while loop (lines 7-13), the key idea is to draw *without replacement* the position i of an item in the transaction j (line 8) and to search the fragment $\mathcal{D}_k$ that contains this item (line 9). Then, we compute the position i' of this item in the fragment $\mathcal{D}_k$ before querying the corresponding item x (see line 10). Finally, the item x is added to the itemset X (line 11) that will be returned (line 14) and the position i is added to the set ϑ (line 12) in order to avoid sampling the same position (and item) twice (see line 8). This process is repeated ℓ times in order to return an itemset X of length ℓ. For example, considering the toy example in Table 2, if we draw the position $i = 2$ in the transaction $j = 1$, we find that the involved fragment is $\mathcal{D}_2$ since $2 > M_{11} = 1$ whereas $2 \leq M_{11} + M_{12} = 3$. Then, we compute $i' = 2 - 1 = 1$ and the item $\texttt{itemAt}(1, 1, \mathcal{D}_2) = B$ is added to the itemset X.

4.2 Theoretical analysis of the method

The following property states that DDSampling returns an exact sample of itemsets without centralizing the distributed database:

Property 2 (Correction). Given a distributed database $\mathcal{P} = \{D_1, ..., D_K\}$ and a length-based utility function $u \in \mathcal{U}$, Algorithm 1 draws an itemset X according to a distribution proportional to $m_u(X, \mathcal{D}) = \mathit{freq}(X, \mathcal{D}) \times u(X)$ where $\mathcal{D} = \mathcal{P}^*$.

This follows from the fact that the three different draws (i.e., transaction j, length ℓ and the sampled itemset) take into account the number of itemsets occurring in each transaction weighted by the utility function. We now study the complexity of our method by distinguishing the two main phases: the preprocessing phase (where the matrix M is computed) and the drawing phase of itemsets. For each phase, we evaluate the complexity in time and in communication.

Time and space complexity. In the preprocessing phase, the weight matrix M is first computed with a complexity in time $O(|\mathcal{P}^*| \cdot |\mathcal{P}|)$. Then, the weight $\omega(j)$ of all transactions $j \in [1..|\mathcal{P}^*|]$ is computed in time $O(|\mathcal{P}^*| \cdot |\mathcal{I}|)$ due to the use of the binomial function. Thus, the preprocessing phase is performed in time $O(|\mathcal{P}^*| \cdot (|\mathcal{P}| + |\mathcal{I}|))$. The draw of itemsets is less expensive. First, the draw of a transaction identifier can be achieved in $O(log(|\mathcal{P}^*|))$. Then, the drawing of an itemset from this transaction is done in time $O(|\mathcal{I}|)$. Therefore, the drawing complexity of an itemset is in $O(log(|\mathcal{P}^*|) + |\mathcal{I}|)$. Besides, the space complexity only depends on the weight matrix dimension and it is in $O(|\mathcal{P}^*| \cdot |\mathcal{P}|)$. As this storage cost is really low in practice, we store the matrix in memory. In an extreme case, the matrix could be stored in a database with a B-tree index to have quick access to the rows.

Communication complexity. In order to evaluate the communication cost, we simply count the number of queries `lengthOf` and `itemAt` required for the two main phases. First, it is easy to see that for the preprocessing phase, the construction of the weight matrix M requires $O(|\mathcal{D}_1| + \cdots + |\mathcal{D}_K|)$ exchanges (using `lengthOf` queries), which is in general significantly lower than the cost of a complete centralization of $\mathcal{P}$ in $O(||\mathcal{D}_1|| + \cdots + ||\mathcal{D}_K||) = O(||\mathcal{P}^*||)$ exchanges (using `itemAt` queries). For the drawing of an itemset of length ℓ, it is clear that ℓ `itemAt` queries are necessary to return an itemset. Therefore, the average communication cost to draw an itemset is equal to the average length $E[L]$ of an itemset returned by DDSampling. Given a distributed database $\mathcal{P}$, we can easily see that $E[L]$ is equal to $\sum_{\ell=1}^{+\infty} P(\ell) \times \ell$ where $P(\ell) = \frac{\sum_{j \in [1..|\mathcal{P}^*|]} \omega_\ell(j)}{\sum_{j \in [1..|\mathcal{P}^*|]} \omega(j)}$ is the probability to draw an itemset of length ℓ. Considering our toy example in Table 2 and the maximum length utility function $u_{\leq 2}$, we have $P(L = 1) = \frac{3+6+3+6+3}{7+11+7+11+7} = \frac{21}{43}$ and $P(L = 2) = \frac{3+4+3+4+3}{7+11+7+11+7} = \frac{17}{43}$. Thus, the average communication cost for drawing an itemset is equal to $E[L] = (\frac{21}{43} \cdot 1) + (\frac{17}{43} \cdot 2) \approx 1.28$. Note that without length constraint (using u_{freq} utility function), this cost is higher and equal to $E[L] = (\frac{21}{56} \cdot 1) + (\frac{17}{56} \cdot 2) + (\frac{11}{56} \cdot 3) + (\frac{2}{56} \cdot 4) \approx 1.71$.

Rejection rate. Two main problems arise in distributed databases: network communication errors (*network failure*) and node inaccessibility (*node failure*). These phenomena induce a bias in the performed drawings because we have to reject a pattern X as soon as an `itemAt` query fails during its drawing. In the case of *network failure*, let ϵ_{net} be the probability that an `itemAt` query failed. Assuming that the failures are independent, we can show that $\mathbf{P}(\text{reject}) = \sum_\ell P(\ell) \cdot \left(1 - (1 - \epsilon_{net})^\ell\right)$. Thus, if ϵ_{net} is a small number, we have $\mathbf{P}(\text{reject}) \approx \epsilon_{net} \cdot E[L]$ (where $E[L]$ is the average length of a sampled itemset). Now, in the case of *node failure*, let ϵ_{node} be the probability that a node is down. If the distributed database is an horizontal partitioning of a centralized database, it is clear that $\mathbf{P}(\text{reject}) = \epsilon_{node}$. Otherwise, if the distributed database is a vertical or hybrid partitioning of $\mathcal{P}^*$, assuming that the items are uniformly distributed over the nodes, we can show that $\mathbf{P}(\text{reject}) = \sum_\ell P(\ell) \cdot \left(1 - (1 - \epsilon_{node})^\ell\right)$. Thus, if ϵ_{node} is a small number, we have $\mathbf{P}(\text{reject}) \approx \epsilon_{node} \cdot E[L]$. Nowadays, ϵ_{net} and ϵ_{node} are very small. Therefore, it follows that the rejection is negligible.

5 Experimental Evaluation

In this section, we evaluate the efficiency of our approach compared to a centralized solution (see Section 5.1) and its interest to find outliers in knowledge bases of the Semantic Web (see Section 5.2). Note that our prototype is implemented in Java and is available at `https://github.com/DDSamplingRDF/ddsampling.git`. All experiments are conducted on a 3.5 GHz 2 core processor with the Windows 10 operating system and 16 GB of RAM.

5.1 Efficiency and robustness of DDSampling

In our first experiments, we use 4 UCI datasets that we uniformly partition into $K = 10$ fragments to simulate distributed databases. In the case of horizontal partitioning (resp. vertical partitioning), each transaction (reps. each item) is randomly placed to a fragment with the same probability $1/K$; in the case of hybrid partitioning, all the items of a transaction are randomly placed to a fragment (with the same probability $1/K$). The first columns of Table 3 show statistical information about all datasets. In all experiments, we use an interval length constraint $u_{\geq 1}$ and $u_{\leq M}$ (with $M = \{3, 5\}$). This choice avoids drawing too much infrequent patterns, in particular for datasets containing long transactions.

Execution times and communication costs. Table 3 indicates the execution times of our method by distinguishing the preprocessing and sampling phase, only in the case of vertical partitioning for $M = 3$. As expected, the preprocessing time (which can be prepared offline) increases with the size of the dataset. However, it is very small (less than 3 s). Regarding the sampling phase, whatever the dataset, it is always under 0.02 ms (per pattern). For the communication costs, we consider the three types of partionings. In the preprocessing phase, the communication cost corresponds to the number of `lengthOf` calls for constructing the weight matrix. This cost is naturally higher for hybrid and vertical partitionings since the items of a transaction may not be in the same fragment. In the drawing phase, the communication cost corresponds to the number of `itemAt` calls, and Table 3 shows the mean number of calls for drawing a pattern. This cost does not depend on the type of partitioning, but on the maximum length ($M \in \{3, 5\}$). Finally, we compare the communication cost between distributed and centralized approaches by evaluating the number N_{max} of drawn patterns in the worst case (when $M = 5$ for vertical partitioning) that are necessary for the sampling approach to be as costly as data centralization. For all datasets, we can see that DDSampling can draw a few thousand patterns with a communication cost lower than that of a data centralization.

Robustness. As seen in Section 4.2, network or node failures induce the rejection of some patterns. In this context, we evaluate the mean rejection rate by drawing 10,000 patterns by varying p and z. We repeat 100 times each experiment by randomly generating the dataset partition and changing down nodes. As rejection rates are independent of datasets, Figure 1 plots the average rejection rate and the standard deviation computed by averaging the results from the

				Nb of `lengthOf` calls			#`itemAt`			Vertical	
Centralized databases				Distributed databases			Distributed		N_{max}	Time (s)	
$\mathcal{D}$	$\|\mathcal{I}\|$	$\|\mathcal{D}\|$	$\|\|\mathcal{D}\|\|$	Hor.	Hybrid	Vertical	M=3	M=5	vertical	Prep.	Sampl.
`Chess`	75	3,196	118k	3,196	31,312	31,427	2.91	4.83	17,976	0.13	$1{\cdot}10^{-5}$
`Connect`	129	67,557	2,905k	67,557	668,296	668,547	2.92	4.86	460,165	2.59	$2{\cdot}10^{-5}$
`Mush.`	119	8,124	187k	8,124	74,036	74,180	2.85	4.70	23,973	0.21	$1{\cdot}10^{-5}$
`Wave.`	67	5,000	110k	5,000	45,081	45,324	2.84	4.68	13,820	0.16	$1{\cdot}10^{-5}$

Table 3: Communication costs and execution times for 4 UCI partitioned datasets

4 UCI datasets. For network failures, the average rejection rate does not depend on a particular partitioning type and Figure 1 (left) presents the evolution of the average rejection rates according to p for $M \in \{1, 2, 3, 4, 5\}$. In accordance with the theoretical analysis, for a given p, we see that the rejection rate increases with M since $E[L]$ increases with M. Moreover, for a given M, we see that the increase of the rejection rate is sub-linear because $E[L]$ decreases with p. Nevertheless, it remains inferior to 50% if p is inferior to 0.1, which already is a level of network failure that is much higher than what is observed in practice. Figure 1 (right) presents the average rejection rates with the proportion of down nodes. First, as proved in Section 4.2, we observe that the average rejection rate is lower for horizontal partitioning than for hybrid or vertical partitioning (since $E[L] \geq 1$). Second, the standard deviation of vertical partitioning is higher than that of other partitioning. Indeed, with vertical partitioning, the average rejection rate can be very low or very high depending on whether the most frequent items are placed or not into down nodes. However, as for network failures, we note that the rejection rate remains acceptable.

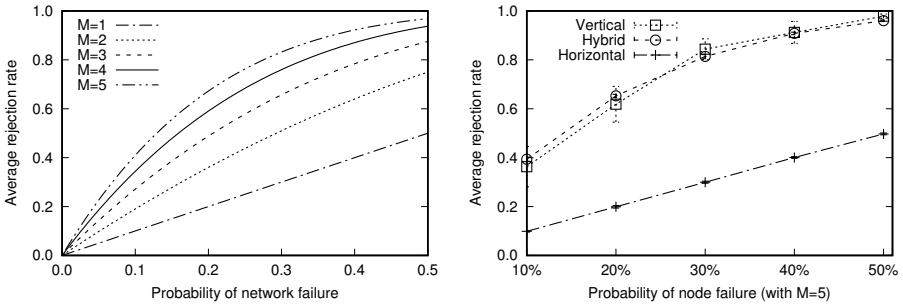

Fig. 1: Average number of rejections with failures

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert_{DBpedia}$	$\lvert\mathcal{I}\rvert_{Wikidata}$	$\lvert t\rvert_{min}$	$\lvert t\rvert_{max}$	$\lvert t\rvert_{avg}$	Time (s) Prep.	Sampl.
Person	772,432	13,142	6,213	8	552	50.02	8,400.80	0.34
Organisation	338,402	19,022	5,504	8	328	36.22	1,847.88	0.27

Table 4: Characteristics and execution times for classes *Person* and *Organisation*

5.2 Outlier detection in distributed databases using pattern sampling

This section aims at detecting outliers in knowledge bases of the Semantic Web by approximating Frequent Pattern Outlier Factor (FPOF) with sampled patterns. More precisely, we use this measure for identifying misclassified entities from two classes (*Person* and *Organisation*) described in *DBpedia* and *Wikidata*. In this context, each entity of a class C is a transaction distributed over *DBpedia* and *Wikidata* where each property p from a RDF triple (x, p, y) is an item. Table 4 provides some statistics about the two classes. Execution times are longer than that for UCI benchmarks because we use public SPARQL endpoints.

Long tail problem and norm constraints. The exact FPOF of a transaction can be approximated using pattern sampling [8]. More precisely, given a database $\mathcal{D}$ and a sample $\mathcal{S}$, the approximated FPOF of a transaction $t \in \mathcal{D}$ is defined by $\widetilde{fpof}_{\mathcal{S}}(t, \mathcal{D}) = \frac{|\{X \in \mathcal{S} \,:\, X \subseteq t\}|}{max_{u \in \mathcal{D}}(|\{X \in \mathcal{S} \,:\, X \subseteq u\}|)}$. Given a maximum length constraint M, we can show that $\widetilde{fpof}_{\mathcal{S}}(t, \mathcal{D})$ tends to the exact FPOF $fpof_{\leq M}(t, \mathcal{D}) = \frac{\sum_{X \subseteq t, |X| \leq M} freq(X, \mathcal{D})}{max_{u \in \mathcal{D}}(\sum_{X \subseteq u, |X| \leq M} freq(X, \mathcal{D}))}$ when the size of $\mathcal{S}$ tends to infinity and $\mathcal{S}$ is sampled according to $m(X, \mathcal{D}) = freq(X, \mathcal{D}) \times u_{\leq M}(X)$. Figure 2 depicts the FPOF distributions of all entities for *Person* and *Organisation* without constraint ($M = \infty$) or with a maximum length constraint $M \in \{1, 2, 3, 4, 5, 10\}$. We see that without constraint or with a large value for M (≥ 5), a large majority of FPOF values are equal to zero, which implies that it is impossible to distinguish outliers from normal entities. Indeed, without constraint, FPOF suffers from the long tail problem [5]. Therefore, the use of a maximum length constraint is crucial for detecting outliers by means of FPOF. Consequently, we use $M = 3$ in the following experiments.

Output space vs. input space sampling. This experiment compares input and output space sampling to determine which method is the best for the same budget (same number of patterns, same communication cost). For a pattern budget k, we start by drawing a sample $\mathcal{S}_{out}$ of k patterns with DDSampling and we calculate its communication cost $Cost_{out}$. Then, we draw a sample of transactions $\tilde{\mathcal{D}}$ requiring the same communication cost: $Cost_{in} = Cost_{out}$. Finally, we draw a sample $\mathcal{S}_{in}$ of k patterns from $\tilde{\mathcal{D}}$. Given a sample $\mathcal{S}$, we evaluate the quality of its approximated FPOF by using the Euclidean distance $\epsilon(\mathcal{S}, \mathcal{D})$ as error: $\epsilon(\mathcal{S}, \mathcal{D}) = \sqrt{\sum_{t \in \mathcal{D}} (\widetilde{fpof}_{\mathcal{S}}(t, \mathcal{D}) - fpof_{\leq M}(t, \mathcal{D}))^2}$. Figure 3 reports $\epsilon(\mathcal{S}_{out}, \mathcal{D})$ and $\epsilon(\mathcal{S}_{in}, \mathcal{D})$ w.r.t. the pattern budget (each measure is the arithmetic mean of

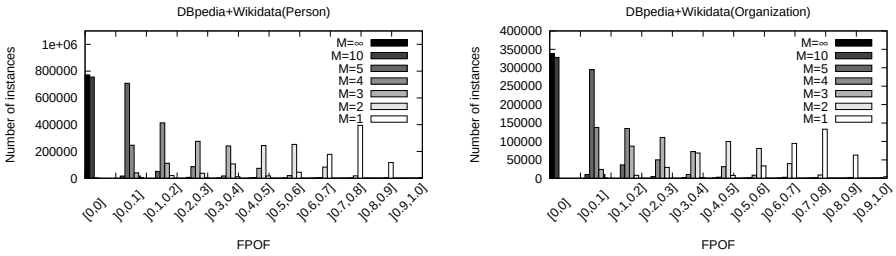

Fig. 2: Long tail problem of the FPOF distributions

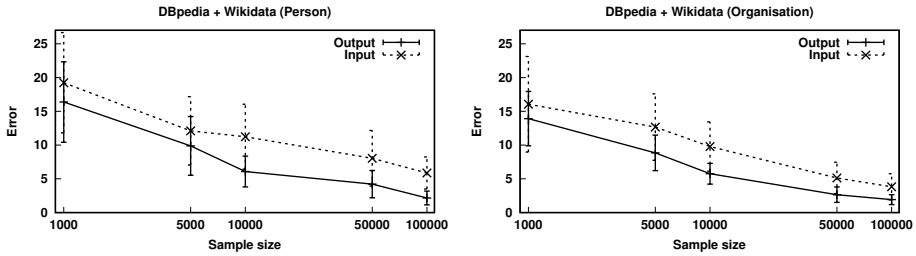

Fig. 3: Evolution of the Euclidean norm error for input/output space sampling

100 repeated samples with its standard deviation). Of course, both errors tend to zero when the sample size tends to infinity. But, it is clear that the convergence is faster and more stable with output space sampling (e.g., the FPOF quality of $\mathcal{S}_{out}$ with 10k patterns equals that of $\mathcal{S}_{in}$ with 100k patterns). This experience shows that output space sampling is more efficient at equal budget.

Qualitative evaluation. We manually analyze the 50 best and worse entities of *Person* according to the FPOF. It is interesting to note that all entities with the highest FPOF are real persons, and not outliers. On the contrary, only 36% of the entities with the lowest FPOF are real persons, and 64% of them can be considered as outliers. Indeed, 44% of entities are fictional characters and more importantly, 8% of them should be classified in *Organisation* (a sibling class of *Person*), and 12% of them should be classified in another class (e.g., *Event*).

6 Conclusion

This paper proposes the first pattern sampling method in a distributed database. It allows to consider different interestingness measures and interestingly, hybrid or vertical partitioning. As only transaction lengths are centralized, the communication costs of DDSampling are low because the exchange of items is done only when the patterns are drawn. The experimental study emphasizes this low

communication cost on several benchmarks datasets whatever the partitioning. We also illustrate the interest of the sampled patterns in RDF data for detecting abnormal entities among persons and organizations without centralizing all the data. In future work, we plan to replace the exact drawing of transactions with a stochastic method so that we do not have to centralize the lengths of all transactions for each fragment.

References

1. Al Hasan, M., Zaki, M.J.: Output space sampling for graph patterns. Proc. of the VLDB Endowment 2(1), 730–741 (2009)
2. Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Scientific american 284(5), 28–37 (2001)
3. Boley, M., Lucchese, C., Paurat, D., Gärtner, T.: Direct local pattern sampling by efficient two-step random procedures. In: Proc. of KDD. pp. 582–590 (2011)
4. Cheung, D.W., Ng, V.T., Fu, A.W., Fu, Y.: Efficient mining of association rules in distributed databases. IEEE Transactions on Knowledge and Data Engineering 8(6), 911–922 (1996)
5. Diop, L., Diop, C.T., Giacometti, A., Haoyuan, D.L., Soulet, A.: Sequential pattern sampling with norm constraints. In: Proc. of ICDM 2018 (2018)
6. Domadiya, N., Rao, U.P.: Privacy preserving distributed association rule mining approach on vertically partitioned healthcare data. Procedia computer science 148, 303–312 (2019)
7. Dzyuba, V., van Leeuwen, M.: Learning what matters–sampling interesting patterns. In: Proc. of PAKDD 2017. pp. 534–546. Springer (2017)
8. Giacometti, A., Soulet, A.: Anytime algorithm for frequent pattern outlier detection. International Journal of Data Science and Analytics 2(3-4), 119–130 (2016)
9. Gombos, G., Kiss, A.: Federated query evaluation supported by sparql recommendation. In: Proc. of HIMI 2016. pp. 263–274. Springer (2016)
10. Jin, R., Agrawal, G.: Systematic approach for optimizing complex mining tasks on multiple databases. In: Proc. of ICDE. pp. 17–17 (April 2006)
11. Kum, H.C., Chang, J.H., Wang, W.: Sequential pattern mining in multi-databases via multiple alignment. Data Mining and Knowledge Discovery 12(2-3), 151–180 (2006)
12. Moens, S., Boley, M.: Instant exceptional model mining using weighted controlled pattern sampling. In: Proc. of IDA 2014. pp. 203–214. Springer (2014)
13. Otey, M.E., Wang, C., Parthasarathy, S., Veloso, A., Meira, W.: Mining frequent itemsets in distributed and dynamic databases. In: Proc. of ICDM 2003. pp. 617–620. IEEE (2003)
14. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)
15. Shen, H., Zhao, L., Li, Z.: A distributed spatial-temporal similarity data storage scheme in wireless sensor networks. IEEE Transactions on Mobile Computing 10(7), 982–996 (2011)
16. Zhang, S., Zaki, M.J.: Mining multiple data sources: local pattern analysis. Data Mining and Knowledge Discovery 12(2-3), 121–125 (2006)
17. Zhu, X., Li, B., Wu, X., He, D., Zhang, C.: CLAP: Collaborative pattern mining for distributed information systems. Decision support systems 52(1), 40–51 (2011)
18. Zhu, X., Wu, X.: Discovering relational patterns across multiple databases. In: Proc. of ICDE. pp. 726–735. IEEE (2007)