

Language-Model Based Informed Partition of Databases to Speed Up Pattern Mining

CARLOS BOBED, University of Zaragoza, Aragon Institute of Engineering Research (I3A), Spain

JORGE BERNAD, University of Zaragoza, Aragon Institute of Engineering Research (I3A), Spain

PIERRE MAILLOT, University Cote d'Azur Inria, CNRS, I3S, France

Extracting interesting patterns from data is the main objective of Data Mining. In this context, Frequent Itemset Mining has shown its usefulness in providing insights from transactional databases, which, in turn, can be used to gain insights about the structure of Knowledge Graphs [6, 39]. While there have been a lot of advances in the field, due to the NP-hard nature of the problem, the main approaches still struggle when they are faced with large databases with large and sparse vocabularies, such as the ones obtained from graph propositionalizations. There have been efforts to propose parallel algorithms, but, so far, the goal has not been to tackle this source of complexity (i.e., vocabulary size), thus, in this paper, we propose to parallelize frequent itemset mining algorithms by partitioning the database horizontally (i.e., transaction-wise) while not neglecting all the possible vertical information (i.e., item-wise). Instead of relying on pure item co-appearance metrics, we advocate for the adoption of a different approach: modeling databases as documents, where each transaction is a *sentence*, and each item a *word*. In this way, we can apply recent language modeling techniques (i.e., word embeddings) to obtain a continuous representation of the database, clusterize it in different partitions, and apply any mining algorithm to them. We show how our proposal leads to informed partitions with a reduced vocabulary size and a reduced entropy (i.e., disorder). This enhances the scalability, allowing us to speed up mining even in very large databases with sparse vocabularies. We have carried out a thorough experimental evaluation over both synthetic and real datasets showing the benefits of our proposal.

CCS Concepts: • **Information systems** → **Data mining**; • **Computing methodologies** → *Natural language processing*; *Knowledge representation and reasoning*.

Additional Key Words and Phrases: Pattern Mining, Language Models, Knowledge Graphs

ACM Reference Format:

Carlos Bobed, Jorge Bernad, and Pierre Maillot. 2024. Language-Model Based Informed Partition of Databases to Speed Up Pattern Mining. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 184 (June 2024), 27 pages. <https://doi.org/10.1145/3654987>

1 INTRODUCTION

Mining structure out from Knowledge Graphs (KGs) [28] is a field of active research from different points of view, such as indexing [40, 43], summarization [13], or structural comparison [6, 39]. In particular, in this latest task, the use of Frequent Itemset Mining directly over *propositionalizations* of Knowledge Graphs (i.e., translations of the graph resources into flattened transactions with items capturing different aspects of each node and relationships) has proved to be especially useful to provide an explainable way of comparing their structure and evolution over time [6, 39]. However,

Authors' addresses: Carlos Bobed, cbobed@unizar.es, University of Zaragoza, Aragon Institute of Engineering Research (I3A), Zaragoza, Spain; Jorge Bernad, jbernad@unizar.es, University of Zaragoza, Aragon Institute of Engineering Research (I3A), Zaragoza, Spain; Pierre Maillot, pierre.maillot@inria.fr, University Cote d'Azur Inria, CNRS, I3S, Nice, France.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 2836-6573/2024/6-ART184

<https://doi.org/10.1145/3654987>

as we want to codify more information about the resources included in a KG, the size of the vocabulary increases (i.e., the number of items we need gets larger), making it increasingly difficult to apply classical mining techniques.

While Frequent Itemset Mining has been the subject of research for a long time, with a plethora of techniques achieving important results [8, 23], its inherent NP nature still poses scalability problems when mining datasets that have large vocabularies (potentially sparse). In brief, Frequent Itemset Mining consists of mining the sets of items that appear frequently in a *transaction database*, i.e., a database where each row is a set of items (in our case, the propositionalization of a graph, where each transaction would be a resource description). Thus, it is not only important to list the frequent itemsets (which lead to an exponential number of combinations), but also to assess their relevance (how interesting they are in the context of the database - our graph description). Approaches based on the *Minimum Description Length* principle such as KRIMP [50] and SLIM [49] have established a way to assess the interest of each mined itemset according to their ability to compress the database. However, the problem of finding an optimum coverage of the database is directly related to the set cover problem [32], which proved to be NP-complete in the size of the vocabulary (i.e., the number of different items that appear in the database), a critical parameter when translating the graph.

To deal with intractability issues, SLIM [49] proposes a pay-as-you-go algorithm that directly exploits the item co-occurrence to select candidates to estimate their contribution to the compression. This algorithm has been successfully used in [6] to mine structure from graphs via propositionalization; but, although the approach could scale up to knowledge graphs formed up to 100,000,000 triples (different versions of DBpedia [34]), there is still a need for further improvements in the scalability of frequent itemset mining approaches in this direction. Recently, Fisher and Vreeken [21] proposed BinaPs, a neural approach to extract patterns via a binarized encoder architecture. However, by construction, the patterns that BinaPs mines are not assured to be actually present in the database. Thus, while the patterns might be useful for exploration, they could lead to unusable results in a real setting.

Apart from new algorithms, given that vocabulary size and sparsity are one of the main cost factors, another way to efficiently speed up all the approaches is partitioning the database in an informed way. While the co-occurrence matrix information is too local to draw global conclusions for partitioning, it can be extended using approaches from other research fields. The problem of modeling large amounts of co-appearing items/tokens/words has been the subject of study of Natural Language Processing (NLP) for decades. In particular, in the last years, with the advent of neural networks, different language modeling techniques based on word embeddings have been proposed, boosting NLP results in almost every task. Such techniques allow us to obtain a dense representation of words that capture their semantics, adopting the distributional semantics hypothesis popularized by Firth [20]: “You shall know a word by the company it keeps”. Applied to our context, we propose to establish a parallelism between transactions and NL sentences, and re-phrase it as “an item is defined by the items it appears with”.

In this paper, we present an approach to speed up frequent itemset mining by splitting a transaction database into different partitions exploiting the information captured by item embeddings obtained using NLP techniques over the database itemsets. In our proposal, first, we obtain the item embeddings considering each transaction as a sentence, and we calculate a vector representation for each transaction. Then, we apply an unsupervised clustering method to gather together the transactions belonging to each partition. Finally, we apply the frequent itemset method to each of the partitions separately. While our proposal is independent of the frequent itemset mining method, we will focus on MDL-based frequent pattern mining methods (in particular, we have used SLIM [49]) as they provide an objective measure of the informativeness of the extracted patterns

(i.e., compression ratios). Besides, as a clustering method, we propose to use k-means [38] due to its scalability and the fact that being able to set a number of clusters in this context allows us to establish a parallelization target (i.e., we can choose k as some value proportional to the number of available CPUs). The main contributions of this paper are:

- We propose a new method based on language models to split the database into cohesive partitions which reduces the vocabulary size of each partition. Splitting the database results in the ability to parallelize the mining process, leading to better performances, and limiting the possible loss of information incurred by adopting local solutions.
- We explore methods to reconstruct the global information (i.e., merging the different local solutions obtained for each partition), evaluating them thoroughly.
- We carry out extensive experimentation to show the effects of different embedding methods, showing that the information losses do not impact significantly our clustering. We test our approach with the use case of mining structure from RDF graphs (revisiting the graph structure pattern scenario from [6]), showing the benefits of our approach in databases with a large and sparse vocabulary.

With the results obtained, we show that our approach will enable to enhance the amount of information we will be able to mine from graphs in an efficient manner following *propositionalization* approaches. The rest of the paper is as follows. First, in Section 2, we present the background required for our approach in Frequent Itemset Mining and Word Embeddings. Then, in Section 3, we formalize the problem we are tackling in our proposal, and, in Section 4, we present our solution based on the database partitioning considering language models. Section 5 contains the complete evaluation we have carried out to validate our proposal, and Section 6 analyzes the related works. Finally, in Section 7, we summarize the conclusions and possible future work.

2 BACKGROUND

In this section, we provide an overview of some background required to fully comprehend our proposal. In particular, we present the Data Mining task we have at hand along with the KRIMP family of approaches (Section 2.1); then, taken from the NLP field, we present the word embedding techniques we will use to obtain a vectorial representation of the different elements of our transaction databases (Section 2.2); and finally, we introduce the notion of graph propositionalization and our proposal for structure mining of RDF graphs [39] (Section 2.3), our main motivating use case.

2.1 On Data Mining

In Data Mining, the pattern mining field aims at extracting interesting subsets of a database. For this purpose, one of the most used instances of this problem is frequent pattern mining, whose goal is to extract all interesting subsets of a given database. Formally, let \mathcal{I} be a set of items. A database \mathcal{D} in pattern mining is a set of *transactions* such that each transaction t is a non-empty subset of \mathcal{I} . The interesting subsets of \mathcal{D} are *itemsets*, which are defined as such subsets X that hold $X \subset \mathcal{I}$ and $X \neq \emptyset$. In frequent pattern mining, the “interestingness” of an itemset X depends on its *support* among the transactions, i.e., the number of transactions that contain X .

$$\text{supp}_{\mathcal{D}}(X) = |\{t \in \mathcal{D} | X \subseteq t\}| \quad (1)$$

The itemsets with support above the *minimal support* threshold minsup are called *frequent itemsets*. A major problem in frequent pattern mining is the very high number of frequent itemsets extracted, which is usually reduced by imposing additional constraints on the selected itemsets. A common approach is to limit the search of frequent itemsets to the *closed frequent itemsets*, which are defined

as such itemsets X such as there is no other itemset Y which holds $X \subset Y$ and $\text{supp}_{\mathcal{D}}(X) = \text{supp}_{\mathcal{D}}(Y)$.

However, imposing constraints on itemsets is usually not enough to obtain a manageable amount of them, so further criteria are used to limit the explosion of possible patterns. In this regard, the KRIMP algorithm [50] is a pattern mining approach based on frequent pattern mining aiming at selecting the most descriptive patterns of the database by adopting the Minimal Description Length principle [46]. Citing the authors, the idea of KRIMP is to find “the set of frequent itemsets that yield the best lossless compression of the database”. One of the advantages of KRIMP is that there are no parameters needed. However, as finding the optimal set is known to be NP-complete, KRIMP is based on greedy heuristics.

Thus, the goal of the KRIMP algorithm is to find a *code table* giving an optimal description of a database. Such code table is a two-column translation table associating itemsets with *codes* (see Figure 1). The set of itemsets contained in the code table is its *coding set*, and it contains the most describing patterns for a database \mathcal{D} . Besides, for completeness, the coding set always contains the singleton itemsets containing each individual item. In particular, the code table containing only the singleton itemsets of database \mathcal{D} is called its *standard code table*. It gives the simplest set of patterns representing a database, with each of them composed of a single item. The lengths of the codes in this table are given by their optimal encoding assuming each item in \mathcal{D} to be fully independent, this is, according to their frequency [50].





Code table CT	
Itemset	Code
A B C	
A B	
A	
B	
C	—

Fig. 1. Example of code table for a database with $\mathcal{I} = \{A, B, C\}$, taken from [50]. The grayscale indicates different codes, while the length of the boxes is related to their length.

To encode a transaction t with a code table CT , KRIMP uses the *cover function*, $\text{cover}(CT, t)$, which assigns a set of disjoint itemsets from the coding set CS to t . Such set of itemsets is called the *cover* of the transaction, and is a set of non-overlapping itemsets holding:

$$t = \bigcup_{X \in \text{cover}(CT, t)} X \quad (2)$$

In KRIMP, the codes assigned to a transaction are selected greedily following their so-called Standard Cover Order, this is, giving priority to the length of the code ($|X|$, in decreasing order), then to its support ($\text{supp}_{\mathcal{D}}(Y)$, in decreasing order), and finally just to the lexicographical order imposed by the items themselves (in increasing order). Thus, the encoding of a database is done by replacing each transaction with the codes of the itemsets of their cover. To guarantee an optimal code size, Vreeken et al. [50] gives a code length measure in bits based on the notion of *usage*. The usage of a code X from a coding table CT of a database \mathcal{D} is defined as:

$$\text{usage}_{\mathcal{D}, CT}(X) = |\{t \in \mathcal{D} | X \in \text{cover}(CT, t)\}| \quad (3)$$

From this definition, the length of the code $\text{code}_{CT}(X)$ of X , $X \in CT$ is defined as:

$$L(\text{code}_{CT}(X)) = -\log \left(\frac{\text{usage}_{\mathcal{D}, CT}(X)}{\sum_{Y \in CT} \text{usage}_{\mathcal{D}, CT}(Y)} \right) \quad (4)$$

Hence, the encoded length of a transaction is defined as:

$$L(t|CT) = \sum_{X \in \text{cover}(CT, t)} L(\text{code}_{CT}(X)) \quad (5)$$

The encoded size of a database \mathcal{D} is defined as:

$$L(\mathcal{D}|CT) = \sum_{t \in \mathcal{D}} L(t|CT) \quad (6)$$

The size of a code table CT of database \mathcal{D} , with a standard code table ST , is defined as¹:

$$L(CT|\mathcal{D}) = \sum_{X \in CT: \text{usage}_{\mathcal{D}, CT}(X) \neq 0} L(\text{code}_{ST}(X)) + L(\text{code}_{CT}(X)) \quad (7)$$

The total encoded size of a database \mathcal{D} and its code table CT is defined as:

$$L(\mathcal{D}, CT) = L(\mathcal{D}|CT) + L(CT|\mathcal{D}) \quad (8)$$

The encoded size of a base can be seen as a reduction of the size of the base according to its regularities. Hence the ratio between the encoded size of a database according to its code table and its encoded size according to the standard code table gives a measure of its regularity. We will call this ratio the *KRIMP compression ratio* defined as:

$$\text{ratio}_{KRIMP} = \frac{L(\mathcal{D}, CT)}{L(\mathcal{D}, ST)} \quad (9)$$

The closer this ratio is close to 0, the more regular the base is. On the other hand, it gets closer to 1 if the base contains few regularities.

To obtain the best compressing model (i.e., the best set of codes that compress the database), KRIMP starts from a set of candidate itemsets (usually the set of closed frequent itemsets) and uses a heuristic based on the size, usage, and support of each the candidate itemsets to find a code table minimizing the encoded size of the transactions in a best effort manner. The itemsets/codes in such a code table are the frequent itemsets that we are looking for². Evolving this algorithm, SLIM algorithm generates the candidate itemsets on the go, estimating their usage. This allows SLIM to be a pay-as-you-go algorithm, allowing to stop either by convergence or by reaching a predefined temporal limit.

2.2 On Word Embeddings

While word embeddings were proposed early in the 2000s [4], in the last ten years we have seen a ground-breaking advance in NLP thanks to these techniques and the recent developments in neural network approaches. In brief, for each word of a given vocabulary, word embeddings calculate a vector in \mathbb{R}^m (i.e., they *embed* the words in a vector space) using different techniques. While this could be achieved by just adopting a one-hot encoding of the vocabulary (thus, leading to very sparse vectors), the point is that these techniques embed the words in a dense vectorial space exploiting the distributional hypothesis [26] in a training corpus. Such hypothesis can be stated as *Words with similar context have similar meanings*, so the embedding technique takes into account the context in which each word appears to calculate their vectors, leading to a space where related/similar words have a measurable similarity. For instance, as donkey and horse tend to appear with similar surrounding words since they have similar meanings, word embedding techniques will produce vectors for donkey and horse that, in some sense, are similar.

¹Note that, to store the code table and not to lose any information, we need to store both the code ID (which has an associated length), and the information about the structure of the code (the code codified with the ST).

²For the sake of readability, we overlook that this set may be suboptimal, given that it applies a heuristic.

Word embeddings are calculated by training a model (usually a neural network), which is called a *language model*. Depending on the characteristics of the language model, word embeddings can be classified into *static* and *contextual* ones. On the one hand, language models that produce static word embeddings [41, 44] are functions $f: V \rightarrow \mathbb{R}^m$ such that for each word w in a vocabulary V returns a unique vector v_w in \mathbb{R}^m . On the other hand, language models that produce contextual word embeddings [10, 15, 37] are functions $f: V^n \rightarrow \mathbb{R}^{n \times m}$, thus, in order to obtain the vector of a particular word, they require a context along the word as input. As an example, if we are working with static word embeddings, the word Java would be mapped to a unique vector regardless of where it appears, while working with contextual word embeddings, the word Java would have different embeddings in the sentences I love Java coffee and I usually program in Java as the contexts where it appears differ.

In our work, we adopt static embeddings mainly for two reasons: 1) they are much faster to train; and 2) they produce embeddings that can be compared directly using the Euclidean distance. To our knowledge, the most used language models for static word embeddings are word2vec [41], GloVe [44], and FastText [7]. For our needs, we discarded FastText as it includes subword information in the embedding which does not add anything in our context as our words (items) are atomic. Word2vec and GloVe capture the distributional semantics of the words in the training corpus by considering context windows around each word in the sentences and solving different problems: word2vec trains a neural network to solve a zero-shot learning multilabel classification problem, while GloVe focuses on solving a regression problem. Their capacity to capture the word relatedness will be directly applied to transactions in our databases as we will see in the next sections³.

2.3 On Graph Propositionalization

To apply classical data mining or machine learning techniques over graphs, one approach usually adopted is to first transform the graph elements into inputs/features that the algorithms are capable of process. While these transformations (i.e., propositionalizations [33]) incur some loss of information, they have been successfully applied in different tasks such as obtaining graph embeddings [47] or mining graph structures with frequent itemset mining [6, 39].

In the current paper, the propositionalization that serves us as a motivating use case is the one proposed in [39], where the resources in an RDF graph (i.e., its nodes) are transformed into single transactions by codifying different aspects of the structural information of their 1-hop neighborhoods as items. In particular, given \mathcal{O} the schema/vocabulary used in the RDF graph, the *itemset vocabulary* for their most expressive propositionalization is defined as:

$$\mathcal{I} = \mathcal{I}_{\mathcal{T}} \cup \mathcal{I}_{P_{IN}} \cup \mathcal{I}_{P_{OUT}} \cup \mathcal{I}_{\mathcal{T}, P_{IN}} \cup \mathcal{I}_{\mathcal{T}, P_{OUT}}$$

where:

- $\mathcal{I}_{\mathcal{T}}$ is the set of items reserved to represent each of the types of the resource.
- $\mathcal{I}_{P_{IN}}$ and $\mathcal{I}_{P_{OUT}}$ are the set of items reserved to represent whether a resource is the source (target for OUT, respectively) of a relationship in any triple in the graph.
- $\mathcal{I}_{\mathcal{T}, P_{IN}}$ and $\mathcal{I}_{\mathcal{T}, P_{OUT}}$ are the set of items reserved to represent that a resource is related to any entity of a given type via a particular relationship. These sets extend the information of $\mathcal{I}_{P_{IN}}$ and $\mathcal{I}_{P_{OUT}}$ with domain/range qualified information⁴.

For example, regarding $\mathcal{I}_{\mathcal{T}}$, i_{Car} would represent that the described resource is an instance of car (i.e., $x \text{ rdf:type } \mathcal{O} : Car$). Thus, using this item-based representation, we can transform a

³We refer the interested reader to the original papers [7, 41, 44] for further details on each particular model.

⁴Both levels of detail (qualified and not qualified) are included to make the translation more robust in the absence of typing information in the RDF graph.

given RDF graph into a transaction database where each transaction codifies the local structure of each node; enabling us to apply any particular frequent itemset mining algorithm. Note that this representation has been successfully used to compare RDF graphs structurally at scale [39] and to assess the data evolution of the graph [6]. We refer the interested reader to those works for further details and examples of the particular propositionalization.

3 PROBLEM STATEMENT

Formally, the KRIMP-based family of algorithms can be seen as functions $A: \mathcal{P}(\mathcal{P}(I)) \times \mathcal{P}(\mathcal{P}(I)) \rightarrow \mathcal{P}(\mathcal{P}(I))$, where I is a set of items and $\mathcal{P}(X)$ is the power set of X , that given a transactional database $\mathcal{D} = \{t_1, \dots, t_n\} \in \mathcal{P}(\mathcal{P}(I))$ and a candidate set $\mathcal{F} = \{f_1, \dots, f_s\} \in \mathcal{P}(\mathcal{P}(I))$, they return a subset $A(\mathcal{D}, \mathcal{F}) = S$, such that $S \subseteq \mathcal{F}$ and $L(\mathcal{D}, S)$ are as minimal as possible. We name such functions *compressing functions*. In the case of the SLIM algorithm, the candidate set \mathcal{F} is any itemset that appears in a transaction in \mathcal{D} , or in other words, any frequent itemset in the database with support greater than 0.

While, in general, these algorithms behave efficiently, they suffer when facing large databases with large vocabulary sizes. One possible way of tackling this problem is to enable the parallelization of these algorithms by partitioning the database with two goals at hand: 1) clustering similar transactions (horizontal partition), and 2) dividing the vocabulary as much as possible (vertical partition). Such a *good* partition of a database would allow us to mine frequent itemsets in each cluster of the partition without losing (too much) global information; thus, locally mined frequent itemsets would correspond to the frequent itemsets in the entire database. Intuitively, let us suppose that we have a database with the four transactions in Table 1. Note that $\{A\ B\ C\}$ and $\{E\ F\}$ are frequent itemsets, if the minimal support is 2. These two patterns are easily identified, if the database is partitioned into $\mathcal{D}_1 = \{t_1, t_2\}$ and $\mathcal{D}_2 = \{t_3, t_4\}$. But if the partition is $\mathcal{D}_1 = \{t_1, t_3\}$ and $\mathcal{D}_2 = \{t_2, t_4\}$, we will mine none of the two above patterns since $\{A, B, C\}$ or $\{E, F\}$ do not occur in enough transactions inside each partition.

Table 1. If the database \mathcal{D} is split into $\mathcal{D}_1 = \{t_1, t_2\}$ and $\mathcal{D}_2 = \{t_3, t_4\}$, it is possible to mine frequent itemsets such as $\{A, B, C\}$ or $\{E, F\}$ independently; but if the partition is $\mathcal{D}_1 = \{t_1, t_3\}$ and $\mathcal{D}_2 = \{t_2, t_4\}$, this would not be possible.

\mathcal{D}	
Id	Trans.
t_1	A B C D
t_2	A B C
t_3	E F G H
t_4	E F J

Once the database \mathcal{D} is partitioned into K clusters, a compressing function A could be applied to each cluster in parallel, obtaining K sets S_i of itemsets along their corresponding cluster. Afterward, such S_i sets must be merged into one set S to obtain the global code table. This requires what we will call a *merge function*: $merge_{\mathcal{D}}(S_1, \dots, S_K) = S$ such that $S \subseteq \cup_{i=1}^K S_i$. Remark that our considered merge functions only take into account subsets of $\cup_{i=1}^K S_i$, pruning the redundant patterns mined in the K clusters⁵.

Formally, we want to solve the following optimization problem:

⁵Other approaches would include recombining local patterns and testing them against the complete database, but they would defeat the speed up purpose.

DEFINITION 1. Informed partition optimization problem. *Given a transactional database \mathcal{D} , a candidate set of itemsets \mathcal{F} , both over a set of items \mathcal{I} , a compressing function A , and an integer $K > 0$, we look for:*

- *A partition $P = \{\mathcal{D}_1, \dots, \mathcal{D}_K\}$ of \mathcal{D} in K disjoint sets: $\mathcal{D} = \mathcal{D}_1 \dot{\cup} \dots \dot{\cup} \mathcal{D}_K$ where $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ ($i \neq j$);*
- *and a merge function $\text{merge}_{\mathcal{D}}$;*

such that the following problem is minimized:

$$\min_{\substack{\{\mathcal{D}_1, \dots, \mathcal{D}_K\} \\ \text{merge}_{\mathcal{D}}}} L\left(\mathcal{D}, \text{merge}_{\mathcal{D}}\left(A(\mathcal{D}_1, \mathcal{F}_1), \dots, A(\mathcal{D}_K, \mathcal{F}_K)\right)\right)$$

where $\mathcal{F}_i = \mathcal{F} \cap \mathcal{D}_i$.

This problem is intractable due to the large search space. The number of partitions is $\left\{\frac{|\mathcal{D}|}{K}\right\}$, where $\left\{\frac{N}{K}\right\}$ are the Stirling numbers of second kind⁶, and the number of merge functions is bounded by $2^{2^{|\mathcal{I}|}}$, giving a search space with $\left\{\frac{|\mathcal{D}|}{K}\right\} 2^{2^{|\mathcal{I}|}}$ points. In this paper, we propose to decouple the problem by: first, finding the partition; and second, looking for a merge function.

4 LANGUAGE-MODEL BASED INFORMED PARTITION OF DATABASES

In order to tackle the partition problem, we advocate for using language models to capture the co-occurrence of the items in the transactions in a meaningful way. In this section, we first give an overview of our proposal (Section 4.1), to then move onto the details of the three main steps that comprise it (Sections 4.2, 4.3, and 4.4).

4.1 Overview: Partition of the Database

As introduced in Section 3, the informed partition optimization problem is comprised of three main steps: 1) obtaining a partition of the database, 2) applying a compressing function (i.e., the selected frequent itemset algorithm) to each partition, and 3) merging the local solutions to obtain a solution for the complete database. These three steps materialize in different sections of Algorithm 1:

Algorithm 1 Pseudocode of the Partition Algorithm

```

1:  $V_i \leftarrow \text{obtainItemEmbeddings}(\text{Database})$ 
2:  $V_t \leftarrow \text{obtainTransactionEmbeddings}(\text{Database}, \{V_i\})$ 
3:  $P_k \leftarrow \text{applyClustering}(\{V_t\}, \text{clusteringMethod})$ 
4:  $CT_k \leftarrow \emptyset$ 
5: for  $p_i \in P_k$  do // In parallel
6:    $ct_i \leftarrow \text{miningAlgorithm}(p_i)$ 
7:    $CT_k \leftarrow CT_k + ct_i$ 
8: end for
9:  $FCT \leftarrow \text{mergeSets}(CT_k, \text{mergeCriteria})$ 
10: return  $FCT$ 

```

Database Partition: First of all, adopting a static language model, we obtain item embeddings from the database (Line 1). Then, we obtain transaction embeddings building them on top of the

⁶See <https://oeis.org/A008277> for the definition of Stirling numbers of the second kind (Last accessed 3rd April, 2024).

obtained item embeddings (Line 2). Once we have the transactions embedded in a vector space, we group them using a clustering method (Line 3)⁷.

Compressing Function: For each of the partitions, we apply a frequent itemset mining algorithm (Lines 5-8). While our approach is algorithm agnostic, we advocate for SLIM [49] algorithm, which offers a pay-as-you-go approach solidly based on the MDL principle (and thus, allowing us as well to limit the mining time for each partition if we need it).

Merge Function: Finally, each of the local solutions (frequent itemsets mined for each partition separately) must be merged according to some specified criteria (Line 9).

In the following sections, we delve into the details of the partition (embedding and clustering) and merge steps.

4.2 Item and Transaction Embeddings

As the first step of database partition, we build static *item* embeddings obtained from the database \mathcal{D} . In this context, the set of all items \mathcal{I} plays the role of the vocabulary (words) in the word embedding context, a transaction t is a sentence made of items (words), and the database \mathcal{D} is the corpus. So, to train word2vec or Glove models, each item $i_k \in \mathcal{I}$ is converted to a word index $indx(i_k) \in \mathbb{N}$, and each transaction $t = \{i_1, \dots, i_n\}$ is converted to a sequence of words indexes $s = indx(i_1) \dots indx(i_n)$. Then, the sentence s is input to the model to train the embeddings.

Note that the distributional semantic hypothesis would be translated into *items that usually co-occur in the same transactions will tend to have similar vector embeddings*. Let us remark that “*similar vectors*” does not necessarily imply nearby vectors using a distance metric as the Euclidean distance. Moreover, we also want to remark that we studied the use of *dynamic* ones, training a reduced version of BERT [15], but the results were similar to the ones obtained with *static* embeddings showing no special benefits, while the time required to train the language model defeated our purpose (seconds vs hours in this case).

Since we are interested in clustering transactions, not items, we need to create transaction embeddings from the item vectors. This is similar to the sentence embedding problem (i.e., obtaining a vector representation of a sentence). The two most frequent strategies to deal with it are 1) to obtain the mean and 2) to concatenate the word embeddings composing the sentence. Since transactions can contain different numbers of items, the concatenation strategy leads to transaction embeddings of different dimensions, not appropriate for applying a clustering algorithm. Thus, we chose the mean strategy, given the item vector embeddings, $emb : \mathcal{I} \rightarrow \mathbb{R}^m$, and a transaction $t = \{i_1 \dots i_s\} \subseteq \mathcal{I}$, the transaction embedding is $emb_T(t) = v_t = \frac{1}{|t|} \sum_{k=1}^s emb(i_k)$.

Finally, we would like to remark that the embeddings trained with word2vec and GloVe have two very interesting properties for our purpose. On the one hand, they can be compared with the cosine similarity [41]. Thus, embeddings with a cosine similarity near to 1, or, equivalently, normalized embeddings to length 1 such that the Euclidean distance⁸ is near to 0, tend to be related. Thus, we ensure that we can apply a clustering algorithm based on the Euclidean distance to obtain related transactions. This, in our context, means that the clustered transactions in this space should contain similar items. Section 5 will confirm experimentally this hypothesis. On the other hand, items with similar contexts tend to have similar embeddings, as it is proved in [35]. Thus, the Euclidean distance between transaction embeddings, unlike the common crisp distance used in Jaccard one to cluster transactions in the literature, captures which items usually appear together in a continuous

⁷We also considered applying the clustering in the item space instead of the transaction space (i.e., clustering the items, and then splitting the database by selecting the transactions that contained any of the items in the cluster), but given the potential co-occurrence of items, this lead to non-meaningful partitions. The exploration of situations where it can be beneficial to use both clusterings (item and transaction levels) is left as future work.

⁸Note that if v and w are vectors with $\|v\| = \|w\| = 1$, then $\|v - w\|^2 = 2 - 2 \cos(\widehat{vw})$.

way, allowing clustering transactions that have few items in common (Jaccard distance close to 0), but which, due to co-apparition frequencies, would be regarded as noise in other clusters.

4.3 Clustering Transactions

Once the transactions are converted into vectors in \mathbb{R}^m as explained in the previous section, they can be used with any algorithm in the literature to cluster continuous data. In our case, we chose k -means algorithm to cluster the transactions for several reasons:

- *Ability to choose the number of clusters.* To parallelize the computations for each cluster according to our computation capabilities (number of CPUs, GPUs, nodes, etc), we need control over the number of produced clusters; thus, we can select a k considering the desired distribution level.
- *Appropriate dissimilarity measure.* All clustering algorithms start with a dissimilarity matrix that measures how different the points to cluster are [52]. By construction, the transaction embeddings are prepared to be compared with the cosine similarity, or in other words, they are compared with the square of the Euclidean distance between the normalized embeddings. k -means algorithm is specially designed to excel when the dissimilarity measure is precisely the square of the Euclidean distance.
- *Scalability.* Using other hierarchical methods such as DBScan [19] was previously evaluated and deemed to be too expensive, defeating the objective of improving the overall performance.

While clustering transactional databases is an active matter of study [3, 9, 11, 24, 25, 55, 56], we did not find an approach with our goal of building item-cohesive transaction clusters. Nevertheless, in Section 5, we used two of the state-of-the-art algorithms, obtaining better results with our clustering algorithm based on embeddings. Further experiments would be required to assert that our clustering method is globally better than current approaches, but we consider it to be out of the scope of this paper as our goal was to speed up the pattern mining process, leaving this aspect as future work.

4.4 Merging the Pattern Collections

The final step of our method is to reconcile the partial solutions and return a global set of patterns that describes the original database. Given that our main goal is to speed up pattern mining in sparse vocabulary databases, we do not consider the contribution of single frequent itemsets but each mined frequent itemset set as a whole (i.e., each entire partial solution) as, otherwise, the cost of merging the solutions would defeat the original purpose⁹. To merge the solutions, we adhere to the MDL principle, and, without loss of generality, we will consider each frequent itemset set as a code table CT_k to calculate the compression ratio it achieves with the original database as discrimination criteria. Based on this, we have considered three possible methods to propose a final set of mined itemsets: Naive Aggressive Method, Informed Method, and Naive Conservative Method.

Naive Aggressive Method. We consider each CT_k as a final candidate solution and we select the one that yields the better model following the MDL principle:

$$NAM(\mathcal{D}, \{CT_k\}) = \min_k \frac{L(\mathcal{D}, ext(CT_k))}{L(\mathcal{D}, CT)} \quad (10)$$

⁹We implemented the acceptance and post-pruning strategy of KRIMP, and the costs of evaluating the contribution of each single code/mined itemset to the overall compression ratio - i.e., codifying the whole database two times for each of them - made that approach unfeasible.

with $ext(CT_k)$ being the code table extended with the singletons covering the items that were not present in the database partition, and $L(\mathcal{D}, CT)$ being the encoded size of the database \mathcal{D} using the codetable CT as defined in [50] and presented in Formula 8.

Informed Method. We incrementally join partial solutions if they yield better compression ratios in a greedy way. Algorithm 2 shows the overview of this method. Starting with the partial solution that yielded the best compression ratio (Lines 1-2), we test whether the other partial solutions contribute to the overall solution (Lines 5-15). We estimate their potential contribution to the current solution (Line 6) and use the most promising one as the current candidate. The contribution is estimated as the product of two parameters:

- The information about the original database that the candidate has captured, measured by its compression ratio as above defined. The lower the better as it shows that the partial solution explains greatly the original database.
- The similarity to the current item distribution of the union of accepted partitions ($crtP$) to the candidate one ($crtCandP$) measured as the Generalized Jaccard Index of both datasets:

$$GCI(P_{crt}, P_{cand}) = \frac{\sum_{i \in V(P_{crt}) \cup V(P_{cand})} \min(supp_{P_{crt}}(i), supp_{P_{cand}}(i))}{\sum_{i \in V(P_{crt}) \cup V(P_{cand})} \max(supp_{P_{crt}}(i), supp_{P_{cand}}(i))} \quad (11)$$

In this case, in our heuristic, we consider the lower the better: We have already accepted the codes mined from the item distributions witnessed (separately) in P_{crt} , so we ideally would like to add a set of codes originated from a different item distribution (i.e., low GCI between P_{crt} and P_{cand}) to compress parts (i.e., low compression ratio yield by P_{cand}) of the original database not covered yet (note that the non-witnessed items lower this value).

Once we have the next potential candidate, we calculate the compression ratio that its addition would yield (Line 7) and check whether it improves the explanation of the original database so far (Lines 8-13). In any case, we discard the candidate as it has already been processed (Line 15).

Algorithm 2 Informed Merge Algorithm

```

1:  $crtCT, crtP \leftarrow NAM(\mathcal{D}, \{CT_k\}) // < CT_k, P_k >$ 
2:  $crtCR \leftarrow compressionRatio(\mathcal{D}, crtCT)$ 
3:  $cands \leftarrow \{CT_k\} - crtCT$ 
4:  $parts \leftarrow \{P_k\} - crtP$ 
5: while  $cands \neq \emptyset$  do
    $crtCand, crtCandP \leftarrow$ 
6:    $nextCandByPot(\mathcal{D}, crtCT, crtP, cands, parts)$ 
7:    $crtCandCR \leftarrow compressionRatio(\mathcal{D}, \cup(crtCT, crtCand))$ 
8:   if  $crtCandCR \leq crtCR$  then
9:     // accept the candidate
10:     $crtCT \leftarrow \cup(crtCT, crtCand)$ 
11:     $crtP \leftarrow \cup(crtP, crtCandP)$ 
12:     $crtCR \leftarrow crtCandCR$ 
13:   end if
14:    $cands \leftarrow cands - crtCand$ 
15: end while
16: return  $crtCR$ 

```

Naive Conservative Method. While encoding the database is by no means as costly as mining the itemsets, if the database is large (which is our main target, large and sparse databases) its cost must not be disregarded. Thus, as the previous methods required to calculate the compression ratio in each stage, we also considered the possibility of using directly the union of all the frequent itemset sets as the final solution, assuming that all of them might contribute to the final solution partially.

In the next section, we evaluate our approach including a comparison of the three methods.

5 EVALUATION

To validate our proposal, we have performed an extensive experimental evaluation to answer our main research questions regarding *Embeddings and Cluster Quality*:

- Q1 Does our transaction embedding technique capture the semantics/similarity of the transactions?
- Q2 How good are the clusters our approach obtains at the transactional level?
- Q3 Does our partitioning approach lead to more ordered partitions?

and regarding *Approach Performance*:

- Q4 If Q3, is it worthy in terms of global time and obtained compression ratios?
- Q5 Which merge strategy seems to be the best according to MDL principles?
- Q6 How does our approach behave with large and sparse vocabulary databases?

In the following, we present all the details of the experimental setup and the different evaluations we have carried out to answer each of the research questions.

5.1 Experimental Setup

We have developed a prototype to perform the database partition which can handle different embedding and clustering techniques¹⁰. In particular, our experiments have been performed using word2vec and GloVe¹¹ as implemented by gensim [45], with the default parameters during 20 epochs, but for the minimum support of each item/word, which we have lowered to 1 to avoid item omissions (unless stated otherwise in the particular experiment). For the transaction classification evaluation, we used the k -neighbors classifier [22] from the *sklearn* python library. Finally, as a clustering algorithm, we have used the parallel k -means implementation provided by faiss [31].

The frequent itemset mining algorithm we have used as a basis is SLIM [49], an any-time version of KRIMP [50]. In particular, we have used a modified version of the original code to enlarge the size of the vocabulary that it could handle (in the original version, there was a limitation of 16 bits for the IDs of the items)¹². As a comparison, due to the clustering algorithm similarity, we have taken tk-means [25] and the SoA approach in transaction clustering, tx-means [25]. tk-means is similar to k -means algorithm, but it uses the Jaccard distance between transactions¹³ instead of the Euclidean distance. As k -means, tk-means needs as input parameters the number K of clusters, producing, in this case, K clusters and L *trash* ones (it is possible to have $L = 0$). These *trash clusters* appear due to the fact that Jaccard distance is equal to 1 for any pair of transactions that do not intersect. On the other hand, tx-means also uses the Jaccard distance, but it tries to infer the best number of clusters using recursive bipartitions of the data until convergence. Such convergence is defined in terms of the *Bayesian Criterion Information* [48] of a set of points. Note that tx-means does not need any input parameter, and we use it following the recommendations of its authors

¹⁰Code available at <https://github.com/sid-unizar/PartMining>

¹¹We had also tested other different embedding techniques, both at item level - GloVe - and sentence/doc level - doc2vec and BERT, achieving similar results as the presented ones.

¹²Code available at <https://github.com/MaillPierre/SWKrimpSim>

¹³The Jaccard distance between two transactions t_1 and t_2 is $d_J(t_1, t_2) = 1 - \frac{|t_1 \cap t_2|}{|t_1 \cup t_2|}$.

in [25]¹⁴. We have left BinaPs [21] out of the comparison as we detected that, even though their loss function penalizes the fact that a particular pattern is not actually present in the database, the patterns that it extracts are not forced to be present, and, when dealing with large vocabularies, we ended up with a set of codes that were not actually supported by the database.

We have conducted experiments with three different types of datasets to answer all the above-presented research questions:

- *Classical Databases (CDBs)*: As the main goal was to speed up mining, we have selected the five most complex databases from the LUCS-KDD dataset [14] according to the time that it took SLIM to converge to a solution. The selected ones were *adult*, *chessBig*, *connect*, *letrecog*, and *pendigits*.
- *Synthetic Databases (SDBs)*: Following Guidotti et al. [25], we evaluate the quality of our clusters using synthetic databases, which we build directly using the code made available by the authors¹⁴. The construction of a database \mathcal{D} is based on six parameters: number of items $|I|$; number of transactions $|\mathcal{D}|$; length of the transactions L ; number of clusters C ; percentage P of overlapping items in the transactions of different clusters; and percentage O of outlier items. In our experiments (Section 5.2.2), we fixed the number of transactions to 100,000, the number of items to 2,000, the average length of the transactions to 50, and the number of clusters to 6. The percentage of overlapping items P and outliers O vary with $P = \{0, 20, 50\}$ and $O = \{0, 30\}$, leading to 6 datasets. We denote these datasets with $PX.OY$, where X and Y loop through the values of P and O , respectively.
- *Large Databases (LDBs)*: Following the motivating works on knowledge graph structure mining and comparison [6, 39], we chose three versions of the DBpedia for structural comparison purposes, namely DBpedia 3.6, Dbpedia 2014, and DBpedia 2016-10¹⁵. Such knowledge graphs were converted into transactional databases using a conversion that captured the basic aspects of each node structure, leading to large databases with sparse vocabularies, which were especially challenging for frequent itemset mining.

Table 2. Details of the datasets used in the experiments.

Database	#Transactions	$ I $	C
Classical DBs			
adult	48,842	97	2
chessBig	28,056	58	18
connect	67,557	129	3
letrecog	20,000	102	26
pendigits	10,992	89	10
Synthetic DBs ($X \in \{0, 20, 50\}$ $Y \in \{0, 30\}$)			
PX.OY	100,000	2,000	6
Large DBs			
DBpedia 3.6	2,476,538	16,466	-
DBpedia 2014	5,063,500	45,162	-
DBpedia 2016-10	6,601,796	61,998	-

The detailed characteristics of each dataset can be seen in Table 2. Both CDBs and SDBs datasets have their transactions annotated for clustering purposes, which allows us to evaluate the quality

¹⁴Code available at <https://github.com/riccotti/TX-Means>

¹⁵We used the list of included files as stated in [6], which can be found at <https://github.com/sid-unizar/PartMining>. The original KGs (before transactional conversion) contain 22, 109, 064, 106, 674, 049, and 102, 273, 104 RDF triples, respectively.

of our clusters serving as ground truth. Finally, all the experiments were run on a desktop computer with an Intel Core i7-6700K processor (4 cores, 8 threads) at 4.00 GHz and 32 GB of RAM memory.

5.2 Embeddings and Cluster Quality

In this set of experiments, we answer the first block of research questions about the quality of the embeddings and the clusters (Q1-Q3). First, we check that our transaction embeddings capture the similarity of the transactions and can be compared with the Euclidean distance of the normalized vectors (Q1); then, we evaluate the quality of the clusters obtained using our approach (both extrinsically and intrinsically, Q2-Q3). For the extrinsic evaluation, we adopt the approach in [25]: we partition already labeled transaction databases (both real and synthetic ones) to compare the obtained clusters with the clusters' ground truth (i.e., the labeled ones) using the *Normalized Mutual Information* (NMI) [2]. For the intrinsic evaluation, we use a modified version of the normalized entropy (the entropy divided by the information length) and the average vocabulary reduction to measure the homogeneity of the clusters (Q3).

5.2.1 Embedding Quality Evaluation: Classification task. To show that in our transaction embedding space nearby embeddings are similar, we evaluate their capability to classify the transactions in the CDBs dataset (i.e., nearby embeddings are assigned the same label). Thus, we trained word2vec and GloVe models with different embedding dimensions (50, 100, and 200), and evaluated a k -neighbors classifier using the cosine distance.

To evaluate the classifier, we ran 10 times an evaluation based on splitting the databases using the proportion 80%-20%, where the 20% was used to obtain the best value $k \in [1, 20]$ of the number of neighbors. With the 80% split, we performed 10-fold cross-validation to obtain the cross-validation accuracy. In Table 3, the mean results are shown. Besides the accuracy achieved by each of our trained models, we also report the results of the KRIMP classifier and the C4.5 decision tree given by the authors in [50]. Finally, since the CDBs databases were built by discretizing continuous attributes (if any) from previous datasets, we detected that the same transaction might appear with different labels in some of the datasets. So, we also report the highest possible accuracy¹⁶ (Max. Acc.) as well as the baseline accuracy (Base Acc.), i.e., the percentage of the most frequent class.

Table 3. Results of the classification task. Italics underlining marks the best results among the word2vec and GloVe models, while bold text marks the best ones among all models.

Database	BaseMax. Acc. Acc.		w2v			GloVe			KRIMP C4.5
			50	100	200	50	100	200	
adult	76.1	88.5	84.4	<u>84.5</u>	<u>84.5</u>	84.2	84.1	84.1	84.3 <u>85.5</u>
chessBig	16.2	100.0	49.2	49.3	49.0	58.0	<u>61.7</u>	60.2	57.9 <u>78.5</u>
connect	65.8	100.0	67.2	68.2	<u>68.3</u>	66.4	66.4	66.6	69.4 <u>80.1</u>
letrecog	4.1	93.6	75.0	<u>75.4</u>	75.0	71.4	71.0	68.5	70.9 <u>77.5</u>
pendigits	10.4	99.9	96.1	<u>96.2</u>	<u>96.2</u>	93.7	92.9	91.0	95.0 95.6

As we can see, using our embeddings and the cosine similarity to compare them is a competitive method. Our simple k -neighbors algorithm obtains comparable results to KRIMP and C4.5 classifiers (beating KRIMP in 3 out of 5 ones). Note that our purpose was not to obtain a classifier for transactional databases, but to test experimentally our hypothesis that similar transaction embeddings are near. According to these results, we can answer Q1 affirmatively.

¹⁶For a transaction with several labels, we consider the most repeated one to calculate the maximal accuracy.

Analyzing further the results, we observe that word2vec models tend to obtain better results, along with higher dimensions for the embedding space (regardless of the model, embeddings of dimension 200 show equal or better performance in 3 datasets). Finally, while we consider the use of transaction embeddings very promising in this task (especially, if we turn the attention to contextual embeddings), we must leave delving into how transaction embeddings can help classification tasks as future work.

5.2.2 Cluster Quality Evaluation. To evaluate the quality of the clusters, we compare them with the ground truth defined by the labels of the transactions in the databases (recall that the datasets we use are labeled with the classes each transaction belongs to - by construction in the case of the synthetic ones). We use the k-means algorithm over the normalized embeddings to cluster the transactions, and we compare the real and the k-means partitions using their NMI. Note that if two partitions have clusters with similar points, their *NMI* tends to 1; while if the partitions are randomly created, their *NMI* tends to 0. Besides, we compare our method with SoA clustering algorithms for transactional databases: tk-means [24] and tx-means [25]. For this comparison, we used both CDBs and SDBs datasets and, as in the previous experiment, we trained word2vec and GloVe models for each database, using dimensions 50, 100, and 200. For better readability, we present only the results of the 200-dimensional embeddings. The complete results can be consulted in <https://github.com/sid-unizar/PartMining>.

To compare our approach to tk-means and tx-means, we ran each algorithm 30 times and compared their NMI values calculated against the ground truth. We used two slightly different setups defined by the characteristics of the algorithms:

- For tk-means, we set the value of k to 4, 8, and 16 for both approaches (tk-means and ours) when evaluating the CDBs Dataset, and to 6 when evaluating the SDBs Dataset (the synthetic databases datasets were created with such a number of clusters). To analyze the statistical relevance of the difference between the NMI results for tk-means and our method, we used Welch's test (the samples are independent).
- For tx-means, as the number of the clusters is not fixed, we first ran tx-means, check how many clusters it produces, and then ran our approach with that value for k . In this case, as the pairs of samples depend on the number k of clusters, to analyze the statistical relevance, we used a paired t-test (the samples are not independent in this case).

In Tables 4 and 5, we can see the mean values of the NMI values obtained for CDBs and SDBs datasets, respectively. The values marked with an asterisk imply that there exists statistical evidence for a confidence level of 0.01 for their respective test (Welch's or paired t-test). Moreover, we underline the NMIs values which hold statistical evidence to be the greatest ones for a given comparison/configuration.

CDBs Dataset. The analysis of the results in Table 4 shows that our approach equals or outperforms consistently both the results of tk-means and tx-means, but for the connect database, which seems to be especially hard for all the algorithms. Besides, the behavior of the word2vec transaction embeddings is almost always better than GloVe ones for these real datasets.

SDBs Dataset. The results obtained for the synthetic databases can be seen in Table 5. In this case, note that, as we control the creation parameters, we can build clusters whose vocabularies do not overlap. In those situations (no overlapping and no outlier items, i.e., $P = 0$ and $O = 0$), we expect our approach to capture the real partition perfectly. Analyzing the results (Table 5), we can see that this is the case and that there is statistical evidence that our method gets better results than tk-means for all the databases. Regarding tx-means, we obtain similar results, with no statistical evidence of any algorithm being better (except for one case P50.00 where GloVe

Table 4. Cluster quality evaluation with the Classical Databases (CDBs) dataset: Comparison with tk-means and tx-means.

Database	k-means (k=4)			k-means (k=8)			k-means (k=16)			k-means		
	tk	w2v 200	GloVe 200	tk	w2v 200	GloVe 200	tk	w2v 200	GloVe 200	tx	w2v 200	GloVe 200
adult	0.117	0.146*	0.122	0.109	0.120*	0.104*	0.096	0.102*	0.091*	0.113	0.143*	0.122*
chessBig	0.039	0.111*	0.053*	0.054	0.121*	0.073*	0.065	0.131*	0.094*	0.053	0.130*	0.088*
connect	0.005	0.003*	0.001*	0.007	0.003*	0.002*	0.009	0.003*	0.002*	0.002	0.001	0.0001*
letrecog	0.067	0.050	0.054*	0.105	0.130*	0.077*	0.169	0.196*	0.100*	0.108	0.131*	0.078*
pen-digits	0.368	0.386	0.103*	0.503	0.547*	0.222*	0.553	0.596*	0.276*	0.472	0.613*	0.302*
Avg	0.1198	0.139	0.066	0.155	0.184	0.095	0.178	0.205	0.112	0.149	0.203	0.118

embeddings obtain the best results). In this set of experiments, in contrast to CDBs Dataset, the behavior of the GloVe transaction embeddings is better than word2vec ones.

Table 5. Cluster quality evaluation with Synthetic Databases (SDBs) dataset: Comparison with tk-means and tx-means.

Database	tk	k-means (k=6)		tx	k-means	
		w2v 200	GloVe 200		w2v 200	GloVe 200
P0.O0	0.878	1.000*	1.000*	0.999	0.999	0.996
P0.O30	0.834	1.000*	1.000*	1.000	1.000	0.997
P20.O0	0.771	0.987*	0.991*	0.981	0.956	0.970
P20.O30	0.771	0.974*	1.000*	0.979	0.960	0.971
P50.O0	0.799	0.961*	0.987*	0.931	0.942	0.974*
P50.O30	0.596	0.901*	0.984*	0.974	0.894*	0.970
Avg	0.774	0.970*	0.993*	0.977	0.958*	0.979

We want to point out an interesting observation about the time complexity of our method: tk-means and tx-means take $O(It \cdot K \cdot |\mathcal{D}| \cdot |\mathcal{I}|)$ time, where It is the number of iterations until convergence, K is the number of clusters, $|\mathcal{D}|$ is the number of transactions in the database, and $|\mathcal{I}|$ is the number of items. Our algorithm follows a similar formula, $O(It \cdot K \cdot |\mathcal{D}| \cdot M)$, where M is the dimension of the embedding. Note that M is a constant, and so, for large sets of items, the time complexity of our method is reduced to $O(It \cdot K \cdot |\mathcal{D}|)$ (which is especially important for our target scenarios of large and sparse vocabularies).

Thus, according to these experimental results, we can answer Q2 stating that our approach obtains clusters of quality similar to or better than SoA approaches focused on this particular task. Thus, although our main purpose was to check the quality of the clusters, we consider that item/transaction embedding is a promising technique to cluster transactional or categorical data; however, it is beyond the reach of this paper to study it in-depth.

As final remarks of this set of experiments, we can draw the following conclusions:

- word2vec transaction embeddings behave better: Although GloVe embeddings obtain better results than word2vec ones for the SDBs Dataset, the overall results indicate that word2vec embeddings are the best choice.
- Embeddings of dimension 200 are a good option: Similar to the usual dimension values adopted for the static embeddings in NLP, these embeddings of dimension 200 have a good performance.

Thus, in the following sections, we will use the word2vec 200-dimensional transaction embeddings for our experiments.

5.2.3 Entropy and Vocabulary Reduction. We now focus on Q3 which requires measuring the order of each of the resulting partitions. In this case, we use two measures:

- **Conditioned Normalized Entropy¹⁷:** We measure the order of each partition calculating their entropy and normalizing their values using the maximum diversity index (i.e., the number of bits required to represent the whole set of items when they all have the same probability). We name it *Conditioned* because we use the maximum diversity index of the whole database (the whole vocabulary) instead of just the vocabulary present in the partition to capture the increase of order obtained by partitioning the database. It is defined as follows: Let \mathcal{D} be a transactional database over a set of items \mathcal{I} , and $P = \{\mathcal{D}_1, \dots, \mathcal{D}_K\}$ a partition of \mathcal{D} ; let us denote by $\mathcal{V}(\mathcal{D}_j)$ the vocabulary of \mathcal{D}_j (i.e., the subset of items that appears in transactions in \mathcal{D}_j). The conditioned normalized entropy of \mathcal{D}_j is

$$CNE(\mathcal{D}_j) = \frac{\sum_{i \in \mathcal{V}(\mathcal{D}_j)} (-p_i \cdot \log_2 p_i)}{\log_2 |\mathcal{I}|}$$

where $p_i = \text{supp}_{\mathcal{D}_j}(i) / \sum_{t \in \mathcal{D}_j} |t|$. The conditioned normalized entropy of the partition P is the mean of the $CNE(\mathcal{D}_j)$: $CNE(P) = 1/K \sum_{j=1}^K CNE(\mathcal{D}_j)$

- **Vocabulary Reduction:** We calculate the average size of the vocabulary of the partitions to check that our approach can discriminate items that do not appear together so frequently, leading to smaller vocabularies.

For comparison purposes, we adopt random partition as baseline, and tk-means and tx-means as alternative transaction clustering approaches. For our approach and tk-means, we adopt the same number of clusters as in the previous setup (i.e., 4,8,16) - tx-means do not require a number of clusters. Figures 2 and 3 show the averaged values for these two measures for different clustering methods. We use CDBs Dataset to test our approach against databases whose vocabularies are not big and sparse (and so, more challenging for our approach).

In Figure 2, first, we can see how random partitioning does not reduce the entropy regarding the global value (the entropy of the original database), which makes sense as the DBs are big enough so that the partitions follow the same distribution as the original one. Besides, we can see how tx-means at first sight might seem the approach that behaves the best, but, as we increase the number of partitions, our approach beats it in four out of five datasets. Besides, note that tx-means decides the number of clusters, which might lead to too much partitioning. tk-means, on the other hand, leads to a lower ordering than our approach.

When we move to the vocabulary size reduction (Figure 3), we can first see how the random partitioning does not affect the vocabulary size of each partition (but for *adult* dataset), which shows the level cohesion of the dataset vocabularies. In this case, we can see how tx-means achieves a strong reduction in all databases but for *connect*. However, our approach, again, in all the cases, steadily increases its capability of decreasing the vocabulary size even in datasets such as *connect*, where the rest of the approaches fail to obtain vocabulary partitions (this will have implications later for execution times).

Thus, we can answer Q3 stating that our approach allows us to obtain more ordered partitions, an order that increases as we increase the number of partitions we consider.

¹⁷Not to be confused with Conditional Shannon Entropy, where the conditional probabilities of two random variables come into play.

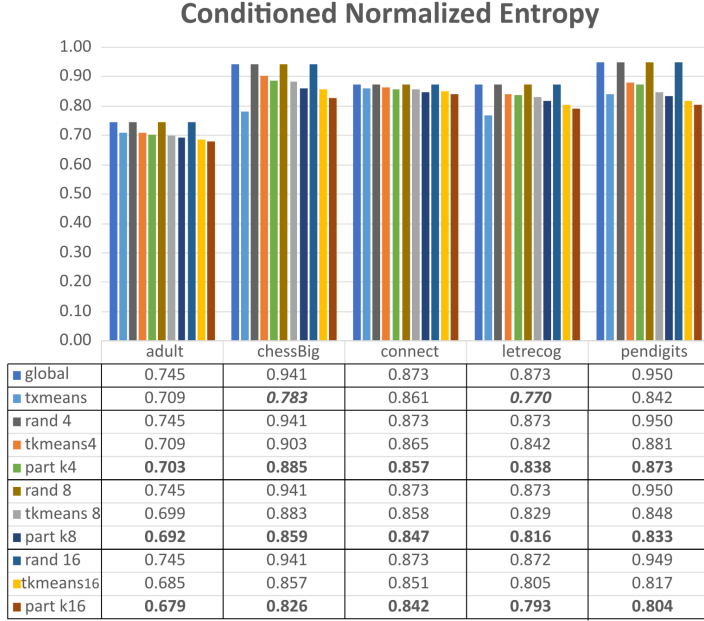


Fig. 2. Average normalized entropy reduction in the partitions of the different datasets. Global is the overall entropy of the original database.

5.3 Approach Performance

Once we have checked Q1-Q3, in this section we focus on Q4 and Q5. We first focus on execution times and overall compression ratios (Q4), to then move to analyze further the impact of the merge algorithm adopted (Q5).

5.3.1 Overall Execution Times and Compression Ratios. To measure the benefits of our approach, we focused on the compression ratio that SLIM achieves on top of the different partition strategies (i.e., how well the mined patterns describe the original database), as well as on the execution times that each approach requires (Figures 4 and 5, respectively). Such execution times include different steps depending on the approach (for parallel executions, we consider the maximum time spent by all the partitions):

- Random partitioning, tx-means and tk-means include 1) partitioning the original database using their different approaches, and 2) executing SLIM on every partition in parallel.
- Our approach includes 1) obtaining the database item embeddings, 2) calculating the embeddings of the transactions, 3) partitioning the database using k-means, 4) executing SLIM on every partition in parallel, and 5) merging the codetables applying Naive Conservative Method (NCM).

Regarding the quality of the locally mined itemsets, we can see in Figure 4 that, as expected, all partitioned approaches achieve worse compression ratios (higher ratios) than mining the whole database, as some global interactions are missed in the partitioning. In particular, we can see how tx-means fails to convey good overall descriptions out from the partitions, and how our approach achieves better ratios than tk-means in all the scenarios. Random partitioning is the one that achieves the closest ratios to mining the original database, which makes sense as the item

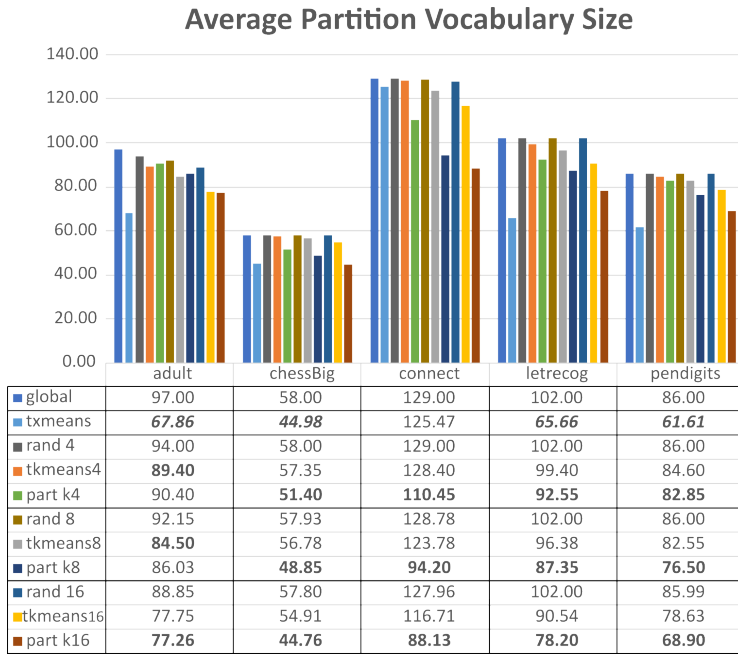


Fig. 3. Average vocabulary size in the partitions of the different datasets. Global is the vocabulary size of the original database.

distribution has been randomly sampled from the original one. However, as we will see when considering Q6, this has an impact in large and sparse vocabulary scenarios, which is our main target¹⁸.

In Figure 5, we can see the average execution times for each of the approaches as above mentioned. Analyzing the times along with the quality of the mined patterns, we can see how tx-means and tk-means neither provide better compression ratios nor better execution times. Besides, this effect is especially remarkable when we focus on *connect*, the database with the largest vocabulary in this dataset, where even random partitioning (which has no calculation overhead) does not outperform our approach. Again, we would like to stress that these datasets are especially challenging for our proposal (small and rather cohesive vocabularies). Thus, when facing Q4, we can state that even though there is a loss in the compression ratio (as expected, given the locality of the partitioned solutions), the improvement in mining time is noticeable even in this challenging scenario.

5.3.2 Merging Approaches. In the previous section, we used NCM as partial solutions merging method for our approach as it is the first method that might come into mind when designing a solution like ours. However, to answer Q5, we need to compare our proposed merging methods.

As we can see in Figure 6, Naive Aggressive Method (NAM) is by far the worst approach: It actually emphasizes the fact that the partitioning we apply is well-informed as a local solution (the codetable obtained using just one part of the database) does not compress well the whole set. Focusing on both Naive Conservative (NCM) and Informed Method (IM), we can see how the Informed Method obtains a slight improvement in the compression ratios. However, this was

¹⁸For this reason, we do not mark the values obtained by random partitioning when they are the best ones in Figures 4 and 5.

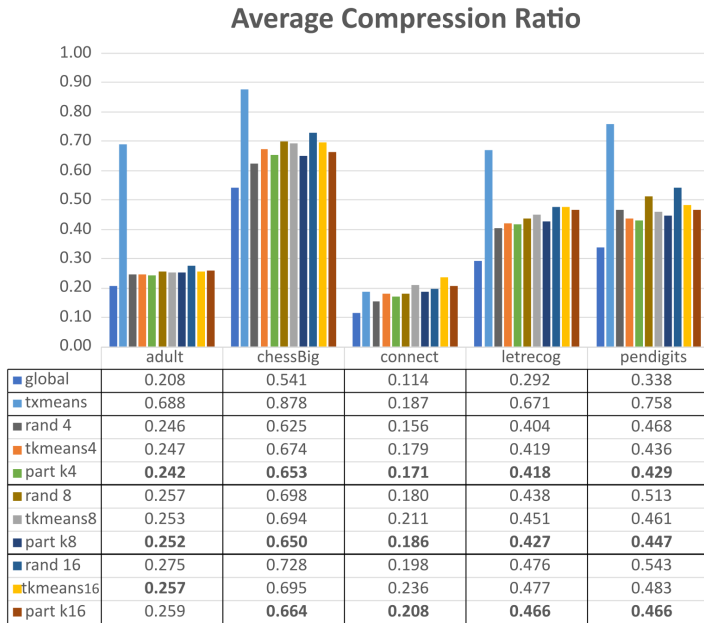


Fig. 4. Average compression ratio achieved by mining the partitions obtained by the different datasets. Global is the compression ratio achieved for the original database. The Naive Conservative Method is applied for all of them.

almost neglectable and it required to compress several times the database, which introduces an overhead that might not be worthy. Note as well that the better the partitioning, the more IM should converge to NCM (ideally, if we had a perfect partition, all the mined code tables would be disjoint, so IM would end up being NCM). Thus, to answer Q5 we can state that NCM obtains the Pareto optimum when we focus on the pair $\langle \text{processing time}, \text{compression Ratio} \rangle$.

5.4 Results On Large and Sparse Datasets

Finally, in order to answer Q6, the main goal of our contribution, we used the LDBs dataset to show the benefits of our proposal. We focus on the databases that result from the structure extraction of DBpedia. Mining these databases proved to be useful for comparing knowledge graphs structurally and evaluating their structural evolution [6, 39], so one of our initial target objectives was to improve the details obtained by improving the mining procedure. In this case, for DBpedia versions, we compare to the compression ratios obtained by SLIM as reported in [6], and the execution details were as follows:

- The reported values using directly SLIM were obtained limiting its running time to 24h for DBpedia 3.6, and to 48h for DBpedia 2014 and DBpedia 2016-10 (SLIM did not converge before the time limit in any case).
- In our approach, we limited the SLIM running time to 1h per partition to show a challenging scenario. The processing times required to obtain the item embeddings, the database partitions, and the total process times (whole partitioning + mining) can be seen in Table 6.
- We ran two configurations for random partitioning: one limiting the mining time to 1h per partition, and another one (more challenging) raising this limit to 4h. We established this

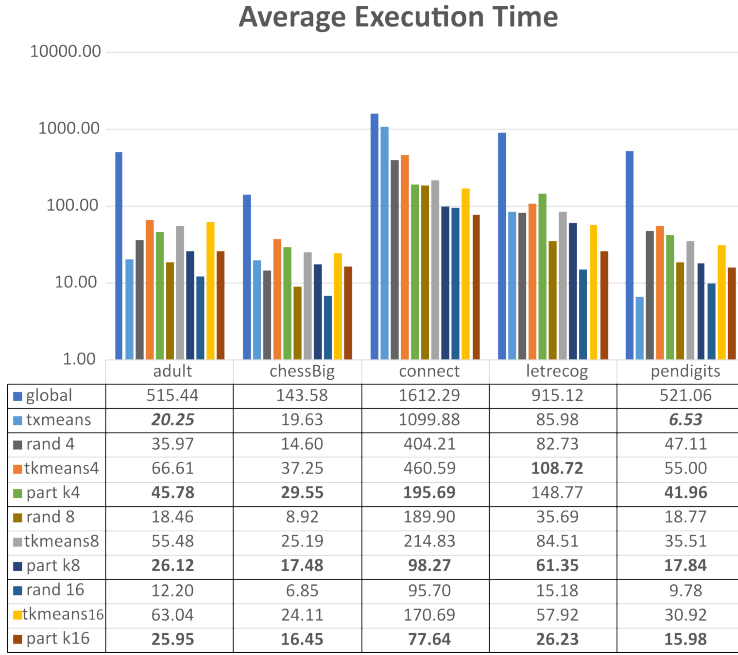


Fig. 5. Average execution time of the different mining approaches. Global is the baseline execution time obtained by applying SLIM directly to the original database.

higher limit by taking the upper bound set by our approach processing DBPedia 2016-10 times (see Table 6).

Both Tk-means and Tx-means were discarded for this experiment due to their performance: 1) Tk-means did not manage to get the clusterings for DBpedia 3.6 in 24h, and 2) Tx-means managed to finish for DBpedia 3.6 in 21h¹⁹ achieving a compression ratio value of 0.444; but when dealing with the other versions of DBpedia it did not manage to get the clusters within the established 24h limit. For these experiments, the desktop computer used is the same one we used in [6], with a configuration similar to the one used for the rest of the experiments.

Table 6. Processing times to obtain the partitions for the different versions of DBpedia: obtaining the item embeddings, partitioning the databases, and the whole processing time including the mining step (1h per partition).

Version	Item Embed. (s)	Partition (s)		Total Time (s)	
		k=8	k=16	k=8	k=16
DB 3.6	883.46	482.02	464.34	4965.48	4947.8
DB 2014	3892.86	1231.84	1234.64	8724.7	8727.5
DB 2016-10	5552.78	4670.87	4809.98	13823.65	13962.76

In Figure 7, we can see the compression ratios achieved by our approach compared to the reported ones and compared to a random partitioning of the database. We can see how, for 8 clusters, our approach: 1) beats random partitioning with 1h processing limit, 2) achieves similar results to random

¹⁹The clustering step took 18h and 30 min, resulting in 276 clusters which were mined within 2 hours and a half.

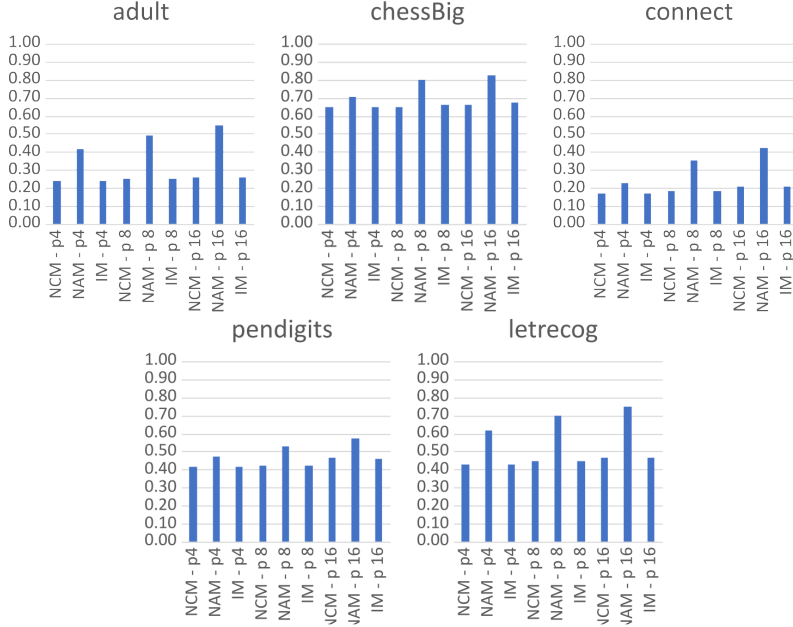


Fig. 6. Average Compression Ratio for the different databases applying the different strategies: NCM (Naive Conservative Method), NAM (Naive Aggressive Method), and IM (Informed Method).

partitioning with 4h, and 3) even it only beats the reported SWJ values for DBpedia 3.6 (bottom row in the table), we have to bear in mind the good results achieved given the difference in real-time thanks to the parallelization. Our approach shows further its benefits as we leave more freedom to the partitioning step, steadily outperforming the random partitioning and the reported values SWJ for all the versions with 16 partitions, and showing that our approach allows us to partition the large and sparse database in an informative way. In terms of the overall speed improvement achieved through parallelization, we would encounter a significant reduction in processing time: 24 hours versus 83 minutes for DBpedia 3.6, 48 hours versus 2.5 hours for DBpedia 2014, and 3.9 hours for DBpedia 2016-10, respectively.

Thus, to answer Q6, we can state that our approach achieves an informed partitioning of the database which is especially beneficial for large and sparse vocabulary databases.

6 RELATED WORK

Given the nature of our proposal, in this section, we consider two main lines of related works: Transactional Data Clustering, and Efficient Pattern Mining.

Transactional Data Clustering. To the best of our knowledge, *large items* [51] is the first clustering algorithm specially designed to cluster transactional data. Instead of defining a distance between transactions, *large items* looks for a partition minimizing two competitive measures: the number of *large* items in a cluster (i.e., those whose support in the cluster is above a threshold, inter-cluster cost), and the number of *small* ones (not large items, intra-cluster cost). This work is superseded by *clope* [55] where the intra and inter-cluster costs are based on the *slope* of a cluster C : the sum of the lengths of all transactions in C (the size of a cluster) divided by the size of the vocabulary of C (the width of a cluster) to the power of a parameter, called *repulsion parameter*, that is difficult

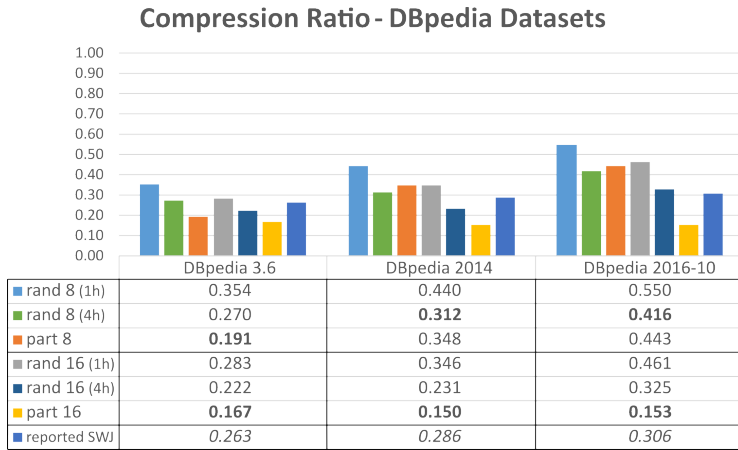


Fig. 7. Compression Ratio for the different versions of DBpedia datasets. The comparison is done against the results reported in [6].

to interpret. More recently, we found the works we compared to in Section 5.1. *tk-means* [24] is a generalization of the *k-means* using the Jaccard distance between transactions; and *tx-means* [25] algorithm also uses the Jaccard distance to obtain the clusters, but it uses a bisection method to avoid as input parameter the number of clusters. According to [24, 25], these two latter methods improve the clusterization quality results of the previous ones. While our proposal was not aimed at transaction clustering, we have shown that our approach behaves better for our purposes.

Efficient Pattern Mining. The state of the art in parallelizing Data Mining techniques involves the use of distributed computing frameworks and GPUs, and the partitioning of the databases. Most of the current efforts have been focused on proposing parallel versions of well-known pattern mining algorithms, such as Apriori [1], Eclat [57] or FP-growth [57]. In [30, 58], the authors describe the GPAApriori and CU-apriori algorithms, a GPU-accelerated implementation of the Apriori algorithm. Some improvements of these algorithms are obtained in [16, 17, 36]. Combining ideas from Apriori and Eclat, the BigFIM algorithm has been developed to work under the MapReduce paradigm [42]. In [53, 54], FIDoop and FIDoop-DP algorithms are presented as an extension of the FP-growth algorithm based on an implementation of the Hadoop MapReduce framework. For a more detailed discussion, the interested reader can consult the survey [27]. However, these approaches inherit the problem of large and sparse vocabularies, where traditional algorithms struggle. Our approach reduces the impact of such situations by partitioning the database initially reducing the complexity boundaries in an informed way.

On the use of parallel resources, it is worth mentioning the different works on *Widening* [5, 29], where the authors propose to explore in parallel different solutions aiming at improving the accuracy of different mining techniques. Although their aim is not to reduce the complexity of the problem, it might be interesting to include these approaches when mining the different local solutions. Recently, Fisher and Vreeken [21] proposed BinaPs, a neural approach to extract patterns adopting a binarized encoder architecture. BinaPs treats the forward and backward steps differently: the forward pass is binarized, thus allowing to capture of the patterns as activations of the intermediate neurons; while the backward pass is continuous to be able to learn using gradient backward propagation. However, while the loss function tries to promote patterns that exist in the database, the patterns that BinaPs mines are not assured to be present in the database (it depends on the number of neurons of the hidden layer). Thus, while the patterns might be useful for exploration, they could

lead to unusable results in a real setting. Finally, Djenouri et al. [18] propose a framework called Clustering-based Pattern Mining (CBPM), which involves the partitioning of a large transaction base with a clustering algorithm before the application of a pattern mining algorithm in parallel. In this case, instead of working directly with the transaction database, our method transforms the database into a continuous embedding space, exploiting richer information about the relationships among items within transactions. Moreover, while their approach uses the clustering of the database to parallelize pattern mining algorithms to obtain *all* relevant patterns using the GPU (such as, Apriori or FP-growth), ours focuses on obtaining only the most representative ones.

To the best of our knowledge, our approach is the only proposal that exploits the current language modeling techniques to partition the database in an informed way, enabling parallel processing over a horizontal partition of the database which retains vertical information. Our approach is directly aimed at improving the expressivity of the propositionalizations of knowledge graphs, enabling the mining of different graph aspects in an explainable way.

7 CONCLUSIONS AND FUTURE WORK

Frequent Itemset Mining remains an open problem when we face large and sparse vocabularies. In this paper, we have proposed a novel approach to deal with these scenarios. By adopting NLP techniques, we can get a continuous version of the database which enables us to partition it horizontally, reducing the complexity of the underlying mining problems in an informed way. Given that we obtained different local solutions, we have explored different possibilities to reconstruct the global pattern set. We have shown that adopting the union of each partition's patterns is Pareto optimal under MDL criteria and the time required to process the database. Finally, we have shown the benefits of our approach on real and synthetic datasets of different sizes. Our approach shows especially strong results in the datasets with large and sparse vocabularies such as the ones obtained from the propositionalization of different versions of the DBpedia.

In future work, we plan to explore the use of other language models (i.e., contextual word embeddings) to capture the interaction between items, learning if necessary a distance out from the data. We will also further extend our analysis of the merging of the local solutions into better global ones exploiting the item's distances. Finally, we will apply our approach to improve graph summarization and structural comparison via propositionalization.

LIMITATIONS

The limitations of our proposal arise from the embedding technique that we use, which is focused on distributional semantics. This imposes the limitation to our current study to be focused on pattern mining techniques which rely on item frequency as the main relevance measure of the itemsets. If other important metrics should be included in the algorithm (e.g., high-utility itemset mining [12]), the embedding technique should be revised to take into account such information to perform the informed partition of the database. Moreover, we would like to remind that, as it happens with data mining algorithms, obtaining the embedding requires to have enough information (i.e., large enough databases) to be meaningful enough to obtain an informed partition.

ACKNOWLEDGMENTS

The work of Carlos Bobed and Jorge Bernad was supported by the I+D+i project PID2020-113903RB-I00 (funded by MCIN/AEI/10.13039/501100011033) and the project T42_23R (funded by Gobierno de Aragón). The work of Pierre Maillot was supported by the ANR DeKaloG (Decentralized Knowledge Graphs) project, ANR-19-CE23-0014. We also thank Jonas Fischer for helping us and making their databases available for our experiments, and the anonymous reviewers for their helpful comments to improve the paper.

REFERENCES

- [1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487–499.
- [2] L.N.F. Ana and A.K. Jain. 2003. Robust data clustering. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, Vol. 2. IEEE, II–II. <https://doi.org/10.1109/CVPR.2003.1211462>
- [3] Daniel Barbará, Yi Li, and Julia Couto. 2002. COOLCAT: An Entropy-Based Algorithm for Categorical Clustering. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management (McLean, Virginia, USA) (CIKM '02)*. Association for Computing Machinery, New York, NY, USA, 582–589. <https://doi.org/10.1145/584792.584888>
- [4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3 (mar 2003), 1137–1155.
- [5] Michael R. Berthold, Alexander Fillbrunn, and Arno Siebes. 2021. Widening: using parallel resources to improve model quality. *Data Mining and Knowledge Discovery* 35 (2021), 1258–1286.
- [6] Carlos Bobed, Pierre Maillot, Peggy Cellier, and Sébastien Ferré. 2020. Data-driven assessment of structural evolution of RDF graphs. *Semantic Web* 11, 5 (2020), 831–853. <https://doi.org/10.3233/SW-200368>
- [7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146. https://doi.org/10.1162/tacl_a_00051
- [8] Christian Borgelt. 2012. Frequent item set mining. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 2, 6 (2012), 437–456.
- [9] Mohamed Bouguessa. 2011. A Practical Approach for Clustering Transaction Data. In *Proceedings of the 7th International Conference on Machine Learning and Data Mining in Pattern Recognition (New York, NY) (MLDM'11)*. Springer-Verlag, Berlin, Heidelberg, 265–279.
- [10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. <http://arxiv.org/abs/2005.14165>
- [11] Eugenio Cesario, Giuseppe Manco, and Riccardo Ortale. 2007. Top-Down Parameter-Free Clustering of High-Dimensional Categorical Data. *IEEE Transactions on Knowledge and Data Engineering* 19, 12 (2007), 1607–1624. <https://doi.org/10.1109/TKDE.2007.190649>
- [12] Raymond Chan, Qiang Yang, and Yi-Dong Shen. 2003. Mining high utility itemsets. In *Third IEEE International Conference on Data Mining*. 19–26. <https://doi.org/10.1109/ICDM.2003.1250893>
- [13] Chen Chen, Cindy Xide Lin, Matt Fredrikson, Mihai Christodorescu, Xifeng Yan, and Jiawei Han. 2010. *Mining Large Information Networks by Graph Summarization*. Springer New York, 475–501. https://doi.org/10.1007/978-1-4419-6515-8_18
- [14] F Coenen. 2003. *The LUCS–KDD discretised/normalised ARM and CARM data library*. Department of Computer Science, The University of Liverpool, UK. <https://cgi.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html>
- [15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. <https://doi.org/10.18653/v1/N19-1423>
- [16] Youcef Djenouri, Ahcene Bendjoudi, Zineb Habbas, Malika Mehdi, and Djamel Djenouri. 2017. Reducing thread divergence in GPU-based bees swarm optimization applied to association rule mining. *Concurrency and Computation: Practice and Experience* 29, 9 (2017), e3836. <https://doi.org/10.1002/cpe.3836> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.3836> e3836 cpe.3836.
- [17] Youcef Djenouri, Djamel Djenouri, Asma Belhadi, Philippe Fournier-Viger, Jerry Chun-Wei Lin, and Ahcene Bendjoudi. 2019. Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases. *Information Sciences* 496 (2019), 326–342. <https://doi.org/10.1016/j.ins.2018.06.060>
- [18] Youcef Djenouri, Jerry Chun-Wei Lin, Kjetil Nørnvåg, Heri Ramampiaro, and Philip S. Yu. 2021. Exploring Decomposition for Solving Pattern Mining Problems. *ACM Transactions on Management Information Systems* 12, 2 (2021), 1–36. <https://doi.org/10.1145/3439771>
- [19] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (Portland, Oregon) (KDD'96)*. AAAI Press, 226–231.
- [20] J. Firth. 1957. *Studies in Linguistic Analysis*. Oxford: Blackwell, Chapter A Synopsis of Linguistic Theory, 1930-1955, 1–32.

- [21] Jonas Fischer and Jilles Vreeken. 2021. Differentiable Pattern Set Mining. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2021)*. ACM, 383–392. https://doi.org/10.1007/978-3-540-68279-0_8
- [22] Evelyn Fix and J. L. Hodges. 1989. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique* 57, 3 (1989), 238–247. <http://www.jstor.org/stable/1403797>
- [23] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin Truong Chi, Ji Zhang, and Hoai Bac Le. 2017. A survey of itemset mining. *WIREs Data Mining and Knowledge Discovery* 7, 4 (2017), e1207. <https://doi.org/10.1002/widm.1207>
- [24] Fosca Giannotti, Cristian Gozzi, and Giuseppe Manco. 2002. Clustering Transactional Data. In *Principles of Data Mining and Knowledge Discovery*, Tapio Elomaa, Heikki Mannila, and Hannu Toivonen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 175–187.
- [25] Riccardo Guidotti, Anna Monreale, Mirco Nanni, Fosca Giannotti, and Dino Pedreschi. 2017. Clustering Individual Transactional Data for Masses of Users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Halifax, NS, Canada) (KDD '17)*. Association for Computing Machinery, New York, NY, USA, 195–204. <https://doi.org/10.1145/3097983.3098034>
- [26] Zellig S. Harris. 1954. Distributional Structure. *WORD* 10 (8 1954), 146–162. Issue 2-3. <https://doi.org/10.1080/00437956.1954.11659520>
- [27] Z. R. Hesabi, Z. Tari, A. Goscinski, A. Fahad, I. Khalil, and C. Queiroz. 2015. *Data Summarization Techniques for Big Data—A Survey*. Springer New York, New York, NY, 1109–1152. https://doi.org/10.1007/978-1-4939-2092-1_38
- [28] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *Comput. Surveys* 54, 4 (2021), 71:1–71:37. <https://doi.org/10.1145/3447772>
- [29] Violeta N Ivanova and Michael R Berthold. 2013. Diversity-driven widening. In *International Symposium on Intelligent Data Analysis*. Springer, 223–236. https://doi.org/10.1007/978-3-642-41398-8_20
- [30] Liheng Jian, Cheng Wang, Ying Liu, Shenshen Liang, Weidong Yi, and Yong Shi. 2013. Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). *The Journal of Supercomputing* 64 (06 2013), 942–967. <https://doi.org/10.1007/s11227-011-0672-7>
- [31] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
- [32] Richard M. Karp. 1972. *Reducibility among Combinatorial Problems*. Springer US, Boston, MA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9
- [33] Stefan Kramer, Nada Lavrač, and Peter Flach. 2001. *Propositionalization Approaches to Relational Data Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg, 262–291. https://doi.org/10.1007/978-3-662-04599-2_11
- [34] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web* 6, 2 (2015), 167–195.
- [35] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (Montreal, Canada) (NIPS'14)*. MIT Press, Cambridge, MA, USA, 2177–2185.
- [36] Yun Li, Jie Xu, Yun-Hao Yuan, and Ling Chen. 2017. A new closed frequent itemset mining algorithm based on GPU and improved vertical structure. *Concurrency and Computation: Practice and Experience* 29, 6 (2017), e3904. <https://doi.org/10.1002/cpe.3904> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.3904> e3904 cpe.3904.
- [37] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. <http://arxiv.org/abs/1907.11692>
- [38] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.
- [39] Pierre Maillot and Carlos Bobed. 2018. Measuring Structural Similarity Between RDF Graphs. In *Proceedings of the 33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018), SWA track, Pau (France)*. ACM, 1960–1967.
- [40] Marios Meimaris, George Papastefanatos, Nikos Mamoulis, and Ioannis Anagnostopoulos. 2017. Extended characteristic sets: graph indexing for SPARQL query optimization. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 497–508.
- [41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, [ICLR] 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. <https://doi.org/10.1162/153244303322533223>

- [42] Sandy Moens, Emin Aksehirli, and Bart Goethals. 2013. Frequent Itemset Mining for Big Data. In *2013 IEEE International Conference on Big Data*. 111–118. <https://doi.org/10.1109/BigData.2013.6691742>
- [43] George Papastefanatos, Marios Meimaris, and Panos Vassiliadis. 2022. Relational schema optimization for RDF-based knowledge graphs. *Information Systems* 104 (2022), 101754. <https://doi.org/10.1016/j.is.2021.101754>
- [44] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543. <https://doi.org/10.3115/v1/D14-1162>
- [45] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. <http://is.muni.cz/publication/884893/en>.
- [46] Jorma Rissanen. 2014. Minimum Description Length Principle. In *Wiley StatsRef: Statistics Reference Online*. John Wiley & Sons, Inc.
- [47] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. 2019. RDF2Vec: RDF graph embeddings and their applications. *Semantic Web* 10, 4 (2019), 721–752. <https://doi.org/10.3233/SW-180317>
- [48] Gideon Schwarz. 1978. Estimating the Dimension of a Model. *The Annals of Statistics* 6, 2 (1978), 461 – 464. <https://doi.org/10.1214/aos/1176344136>
- [49] Koen Smets and Jilles Vreeken. 2012. Slim: Directly mining descriptive patterns. In *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 236–247.
- [50] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. 2011. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery* 23, 1 (2011), 169–214.
- [51] Ke Wang, Chu Xu, and Bing Liu. 1999. Clustering Transactions Using Large Items. In *Proceedings of the Eighth International Conference on Information and Knowledge Management (Kansas City, Missouri, USA) (CIKM '99)*. Association for Computing Machinery, New York, NY, USA, 483–490. <https://doi.org/10.1145/319950.320054>
- [52] Dongkuan Xu and Ying jie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science* 2 (2015), 165–193. <https://doi.org/10.1007/s40745-015-0040-1>
- [53] Yaling Xun, Jifu Zhang, and Xiao Qin. 2016. FiDooP: Parallel Mining of Frequent Itemsets Using MapReduce. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46, 3 (2016), 313–325. <https://doi.org/10.1109/TSMC.2015.2437327>
- [54] Yaling Xun, Jifu Zhang, Xiao Qin, and Xujun Zhao. 2017. FiDooP-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2017), 101–114. <https://doi.org/10.1109/TPDS.2016.2560176>
- [55] Yiling Yang, Xudong Guan, and Jinyuan You. 2002. CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Edmonton, Alberta, Canada) (KDD '02)*. Association for Computing Machinery, New York, NY, USA, 682–687. <https://doi.org/10.1145/775047.775149>
- [56] Ching-Huang Yun, Kun-Ta Chuang, and Ming-Syan Chen. 2006. Adherence clustering: an efficient method for mining market-basket clusters. *Information Systems* 31, 3 (2006), 170–186. <https://doi.org/10.1016/j.is.2004.11.008>
- [57] M.J. Zaki. 2000. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering* 12, 3 (2000), 372–390. <https://doi.org/10.1109/69.846291>
- [58] Fan Zhang, Yan Zhang, and Jason Bakos. 2011. GPAPriori: GPU-Accelerated Frequent Itemset Mining. In *2011 IEEE International Conference on Cluster Computing*. 590–594. <https://doi.org/10.1109/CLUSTER.2011.61>

Received October 2023; revised January 2024; accepted March 2024