Modern Management based on Big Data II and Machine Learning and Intelligent Systems III
A.J. Tallón-Ballesteros (Ed.)

doi:10.3233/FAIA210299

Dist Frequent Next Neighbours: A Distributed Galois Lattice Algorithm for Frequent Closed Itemsets Extraction

Naomie Sandra NOUMI SANDJI [a,1], Djamal Abdoul Nasser SECK [a]
[a] *Department of Mathematics and Computer Science, Faculty of Science and Technology, Cheikh Anta Diop University of Dakar, Senegal*

Abstract. The general purpose of this paper is to propose a distributed version of frequent closed itemsets extraction in the context of big data. The goal is to have good performances of frequent closed itemsets extraction as frequent closed itemsets are bases for frequent itemsets. To achieve this goal, we have extended the Galois lattice technique (or concept lattice) in this context. Indeed, Galois lattices are an efficient alternative for extracting closed itemsets which are interesting approaches for generating frequent itemsets. Thus we proposed Dist Frequent Next Neighbour which is a distributed version of the Frequent Next Neighbour concept lattice construction algorithm, which considerably reduces the extraction time by parallelizing the computation of frequent concepts (closed itemsets).

Keywords. frequent itemset, association rules, closed itemset, Galois lattice, big data, data mining.

1. Introduction

Association rule mining is a data mining task introduced by Agrawal et al. in 1993 [1] that discovers meaningful relationships between attributes according to their associations in databases. It consists of two steps: the discovery of frequent itemsets and the generation of association rules. The discovery of frequent itemsets, which is the most important step, consists in discovering sets of attributes that appear very often with the same groups of values on a large part of the data. There are several areas of application. In the **commercial sector [1], [2]**, the analysis of the transaction databases allows to determine the items that are often bought together and thus to highlight the causal relationships between them, which then allows to elaborate a good commercial planning as to the location of the items in the shelves.In the **telecommunication sector**, they are used for example for the prediction of incidents in remote maintenance processes, in order to limit the costs of manual interventions and to improve the quality of service[3]. In the **medical field [4]** ,organizations (laboratories, hospitals, etc.) can search frequent

[1]Corresponding Author: Naomie Sandra NOUMI SANDJI, Cheikh Anta DIop University of Dakar, Senegal, B.P.:5005 Dakar-Senegal; E-mail:noumisandji03@gmail.com.

sets in patient databases to determine symptoms that precede an illness, or a treatment to be provided.Several algorithms for frequent itemsets extraction have been proposed, including the Apriori algorithm [5], which is one of the first algorithms proposed, and its improved variants such as ECLAT [6], Fp-growth [7], etc. These algorithms sequentially analyze the databases to extract frequent itemsets. With the explosion of data favored by the development of the internet and data collection and storage tools, the size of transaction databases becomes very important and the extraction task becomes a complex and computationally expensive task, as it requires several scans of the transaction database for the calculation of the support of the candidat itemsets in order to generate the frequent ones. Next to these algorithms there are others based on frequent closed itemsets such as CLOSE [8], CHARM [9], CLOSET [10], etc. Frequent closed itemsets constitute reduced bases allowing to find all frequent itemsets, therefore reducing the set of candidates to be processed and the number of scans in the databases. Thus the extraction of frequent closed itemsets is more advantageous in the context of large volumes of data, as they allow the improvement of the computation time. In this paper we propose **Dist Frequent Next Neighbour**, which is a distributed version of the Frequent Next Neighbour algorithm for extracting frequent closed itemsets from Galois (or concept) lattices. We implemented this algorithm with the Apache Spark framework. This proposed algorithm allows the efficient extraction of frequent closed itemsets by optimizing the closure computation and the support computation of closed itemsets. The plan of the paper is organized as follows: after presenting some definitions in section 2, a literature review is presented in section 3. Then section 4 presents in detail the Dist frequent Next Neighbour algorithm; section 5 evaluates the performances of the Dist Frequent Next Neighbour algorithm and compares them to those of the Spark-Apriori algorithm [11] and finally section 6 to conclude the paper.

2. Some Definitions of the Term

In this section we present some basic notions for the good understanding of the document.

A **Formal context** also called transaction base, is a triplet **K = (O,A,I)**, where O is the set of objects or transactions, A is the set of attributes and I is the incidence relation between O and A (O and A are two disjoint sets). We write o$I a_i$ or (o,a_i) $\in$ I to mean that the object o has for attribute a_i. It is represented in the form of an array in which the objects are in row and the attributes in column, we use 1 or X to represent the incidence relation o$I a_i$. If an object o does not have an attribute a_i we leave the box empty or we put 0 in the box. Let O = {1,2,3,4,5,6,7} the set of objects and A = {a,b,c,d,e,f} the set of attributes, the table 1 below presents an example of formal context.

Table 1. Example of a formal context[12]

Objects \ Attributes	a	b	c	d	e	f
1	X	X	X	X	X	
2	X		X			
3		X	X			
4	X	X		X		
5			X	X		
6		X		X		
7				X		X

A **item** or **item** or **attribute**, is any element belonging to a finite set of distinct elements. $\boldsymbol{I} = \{x_1, x_2, ..., x_m\}$.

Example: In the table 1, the finite set of elements $\boldsymbol{I} = \{a, b, c, d, e, f\}$ contains 5 items a, b, c, d, e, f.

An **itemset** is a set of items or any subset of items of I . The number of items of an itemset constitutes its length, an itemset containing **k** items is called a **k-itemsets**

The **support** of an itemset is the percentage of transaction instances that contain the itemset. It allows to measure the interest of the itemset, mathematically the support of an itemset X is defined by :

Support(X) = Card $(\{t_i / X \subseteq t_i, t_i \in \boldsymbol{T})$. Where Card(A) is the cardinal of the set A.

A **frequent itemset** is an itemset whose support verifies the fixed support threshold y . In other words the itemset X is frequent if and only if support(X) $\geq$ y.

A **Galois correspondence** [13] is a **derivation operation** which allows to establish the link (the correspondence) between objects and attributes of a context **K**. It can be defined from two functions **f** and **g** :

• Soit X $\subseteq$ O, $f : P(O) \rightarrow P(A)$, $f(X) = \{ a \in A / \forall o \in X, oIa \}$

• Soit Y $\subseteq$ A, $g : P(A) \rightarrow P(O)$, $g(Y) = \{ o \in O / \forall a \in Y, oIa \}$

Example in the table 1, f({3,5}) = {c} and g({a,c}) = {1,2} which means that the set of attributes {a,c} has as a set of common objects {1,2}. In the same way, the set of objects {3,5} has in common the attribute {c}.

The **Closure of sets [13]** of a set S is the smallest superset containing this set. The two functions f and g will be used to compute the closure of X (X $\subseteq$ O) and the closure of Y (Y $\subseteq$ A) representing respectively a subset of objects and a subset of attributes. To do this we compose the functions f and g as follows:

• X'' =g(f(X)) : closing operator on objects

• Y'' = f(g(Y)) : closing operator on attributes.

A set is said to be closed if it is equal to its closure. Thus X is closed if X = X'' (respectively Y is closed if Y = Y'').

A closed itemset is said to be **frequent** if its support verifies the minimum fixed support .

A **Formal concept** is a couple (O_1,A_1), with $O_1 \subseteq$ O, $A_1 \subseteq$ A, $O_1 = A'_1$, and $A_1 = O'_1$. O_1 is the extension (object) of A_1 and A_1 is the intention (attribute) of O_1.

O'_1, A'_1 correspond respectively to the derivation operations f (O_1) and g(A_1)

Example : ({1,2,3,5},{c}) and ({1,2},{a,c}) are both formal concepts.

A **Concept lattice** is a pair **(S,**$\leq$**)**, where: $\leq$ is an **order relation** on the set **S**. That is to say a binary relation which verifiy the following properties:

1. reflexivity : for all $x \in S$ we have $x\Re x$
2. antisymmetry: for all $x, y \in S$ we have ($x\Re y$ and $y\Re x$) involves x = y
3. transitivity: for all $x, y, z \in S$ we have ($x\Re y$ and $y\Re z$) involves $x\Re z$

3. State of Art

Several algorithms have been proposed to solve the problem of itemset extraction in large volumes of data [14,15,16,17,18,19]. The majority of these algorithms use techniques, tools and strategies to solve the problems of generating a large number of candidates in the candidate itemset generation phase, efficient support counting in order to improve retrieval performance. They are also much more based on frequent itemsets extraction algorithms such as Apriori [5], ECLAT [6] and Fp-Growth [7]. In this paper we propose the Dist Frequent Next Neighbours algorithm for frequent closed itemsets extraction which improves and speeds up the frequent itemsets extraction process as frequent closed itemsets form a basis for frequent itemsets.

4. The Dist Frequent Next Neighbours Algorithm

In this section we will first present the Frequent Next Neighbours algorithm, then the design of the Dist Frequent Next Neighbours distributed algorithm followed by its architecture and implementation.

4.1. The Frequent Next Neighbours Algorithm

The **Frequent Next Neighbours [20]** algorithm used to address the problem of determining only frequent concepts, rather than all concepts. We would like to recall that a **Frequent Concept** is a tuple **(extension, intention)** with the intention being a **Frequent Closed Topic**. The Next Frequent Neighbour algorithm is a top-down breadth-first search algorithm (we start with the parent concept and generate its descendants (direct successors). It is formalized by the function **Find Frequent Lower Neighbours** which computes and/or lists all the frequent lower neighbors (successors/children) of each concept starting with the parent concept. The Frequent Next Neighbours sequential algorithm operates in five steps that we have identified ourselves, namely:
step 1: The combinations of the intentions. In this step, combinations (union) are made between the attribute (intention) of the current concept and each attribute of the set of attributes of the context, excluding the attribute of the current concept. The objective is to form potential intentions (or candidate intentions) **Y1**, for the calculation of the lower neighboring concepts of the current concept.
Step 2: The support check. In this step, the support of each of the combinations obtained in step 1 is computed and we retain those which have a support higher than or equal to the fixed minimum support.
Step 3: The calculation of the neighboring concept. For each combination Y1 retained in step 2 we compute the couple ($Y1'$, $Y1''$), where $Y1'$ and $Y1''$ are the first and second derivatives which correspond respectively to the Galois connection functions g and fog, **g** which associates the attributes with their objects and **fog** which is the closure operator on the attributes.

Step 4 : The maximality check. This step makes it possible to check the relation of coverage (direct link) between the parent concept and the calculated neighbour concept (child concept) .
Step 5: The existence check. This last step allows to be reassured of the uniqueness of a generated frequent concept. Generally, the first three steps are grouped into a single step called the generation of frequent neighbour concepts of a concept.

4.2. Design of the Dist Frequent Next Neighbours Algorithm

In order to be able to design a parallel/distributed architecture of the Frequent Next Neighbour concept generation algorithm, we first identified the independent actions of the algorithm that can participate to the reduction of the execution time. We thus identified the step of generating the frequent neighbour concepts of a concept as an independent step, because it can be carried out in a completely independent way. This step which is the most complex step can be executed by several nodes simultaneously. And the maximality and existence checks are delegated to a main node, because they require the generation of all concepts by each node before being performed. Thus our distributed approach is carried out in two phases:
Stage 1 : Parallel computation of frequent neighbour concepts. It is carried out in three stages including the combinations of the intentions, the support control and the calculation of the frequent neighboring concepts of a concept.
Stage 2 : The generation of frequent formal concepts (frequent closed itemsets) This is done in two steps, the maximality check and the existence check.

4.3. Architecture of the Dist Frequent Next Neighbours Algorithm

To improve the performance of the frequent closed itemset extraction process, the Dist Frequent Next Neighbours algorithm is based on the logical distribution of the search space and a parallel execution of the generation of frequent neighbour concepts of a frequent concept which are frequent closed itemset. Indeed, we make a distribution of the set of attributes and on each distribution we have distinct combinations of attributes that constitute subsets of the search space. Thus the architecture of our distributed approach is given by the figure 1 below:

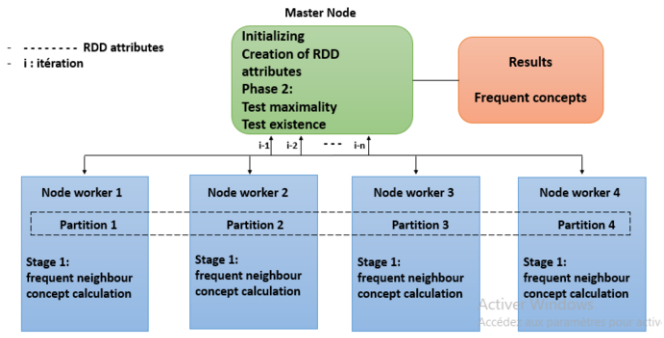

Figure 1. Architecture of the Frequent Next Neighbour distributed algorithm.

4.4. Implementation on Spark

This subsection presents in detail the different steps of the implementation of the distributed algorithm Dist Frequent Next Neighbours in the Spark environment via the **pyspark** python API.

Stage I: Parallel calculation of frequent neighbour concepts

The algorithm starts by creating the object **SparkContext**, whose role is to coordinate Spark applications, running as independent sets of processes on a cluster. It takes as input the formal context file, the structure of this file is such that objects and attributes are separated by the **space** character . The input file is initially loaded into a Spark RDD. The use of RDDs makes it possible to make the most of the resources available at the cluster level and also take advantage of fault tolerance. To facilitate access to these data sets, broadcast variables (command **broadcast()**) were used to store the data dictionaries in the cache of each node-slave. Next we start by creating our parent concept; we create an RDD that contains the set of all context attributes, excluding the attributes of the current concept. Initially our current concept is the parent concept. This rdd will allow us to perform the parallel calculation of the frequent neighbourconcepts of a concept.

Stage II: The generation of frequent formal concepts (frequent closed itemset)

After each iteration of generating the frequent neighbour concepts of a concept, the results are sent to the master node by the spark action **collect()** for maximality and existence testing to generate the relevant and unique frequent concepts (frequent closed itemset). stages I and II are repeated for all these generated frequent concepts until there are no more frequent concepts.

5. Experimental Assessment

In this section, we analyze the performance of our algorithm by analyzing the extraction time of frequent (closed) itemset compared to those of the distributed Apriori algorithm on Spark (Spark-Apriori [11]). For this purpose, a series of several tests is performed by varying the value of the minimal support.

5.1. Data Sets

We have used for our experiments the following databases : Mushrooms, Retails, T40I10D100K. These databases are available on the internet. They are distinguished by their type. We have the **dense databases** which better reflect the real transaction databases, they produce a significant number of frequent itemset of rather large size, and this even for high values of the support. They are also characterized by a large number of attributes per object (transaction), and a limited number of distinct attributes. We also have the **sparse databases** which most often reflect the synthetic transaction bases, characterized by a few items per object and a large number of distinct attributes. The Mushrooms databases contain information about mushrooms, they contain 8124 transactions with an average size of 23 attributes per object and 115 items corresponding to the characteristics of mushrooms in total. Retails and T40I10D100K are synthetic databases built according to the properties of sales data. They contain respectively 88162 and 100 000 transactions with an average size of 15 attributes per object for Retails and 40 attributes

for T40I10D100K. The table 2 lists the characteristics of these databases we used for our tests.

Table 2. Characteristics of transactional test databasest

Basics	Type	Number of items	Number of transactions
Mushrooms	Dense	115	8124
T40I10D100K	Sparse	1000	100 000
Retails	Sparse	16469	88 162

For the performance study we used a cluster on amazon's cloud with 3 nodes whose characteristics are: 8 virtual cores, 32GB of RAM and 128GB SSD.

5.2. *Performance Evaluation*

The results obtained are represented by the figures 2, 3 and 4.

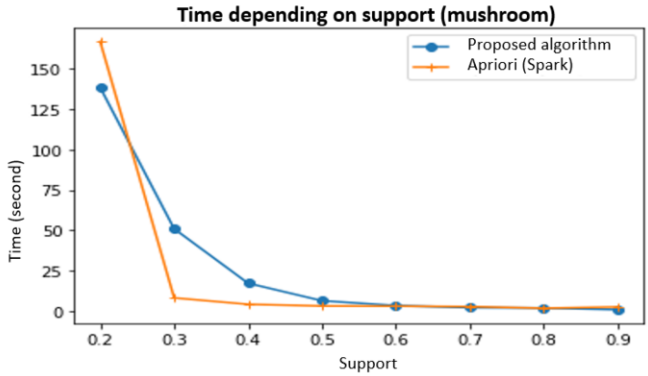

Figure 2. Variation of the extraction time of the Spark-Apriori algorithm and the proposed algorithm for the Mushrooms database as a function of the support thresholds.

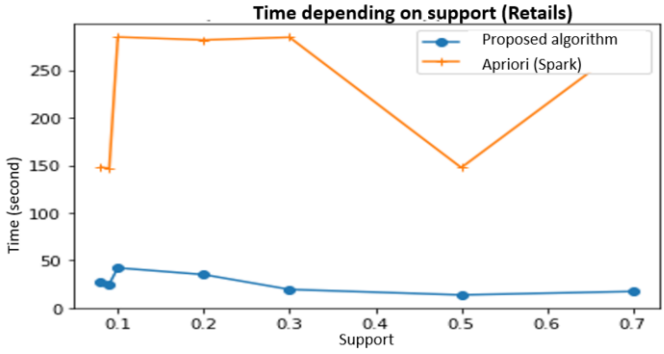

Figure 3. Variation of the extraction time of the Spark-Apriori algorithm and the proposed algorithm for the database Retails as a function of support thresholds.

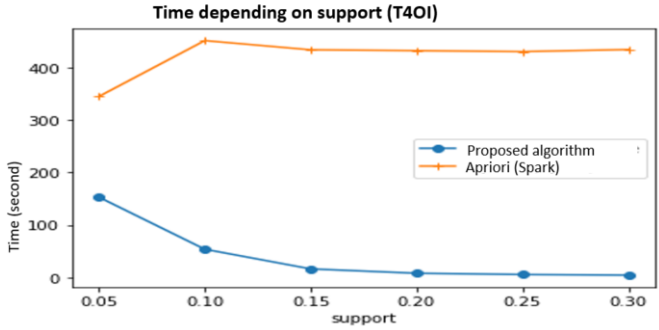

Figure 4. Variation of the extraction time of the Spark-Apriori algorithm and the proposed algorithm for the T40I10D100K base as a function of the support thresholds.

Interpretations:
Reading of the figures 2-4. We could know that :

- For the Mushrooms dataset, the response times of the Apriori distributed algorithm are much lower than those of the Dist Frequent Next Neighbours algorithm for support thresholds of 0.3, 0.4, and 0.5. This can be explained by the fact that the Mushrooms dataset is dense and correlated. Then, we have a large number of frequent itemsets generated during iterations, which is less advantageous for our proposed algorithm, as it performs more operations than the distributed Apriori algorithm. And for support thresholds of 0.7, 0.8 and 0.9, the response times of the two algorithms tend to be identical. This is due to the fact that for these minimum support values the number of frequent itemsets generated at each iteration is approximately the same as the number of closed itemsets generated. As a result, the search spaces are almost identical.
- For the Retails and T40I10D100K datasets, the response times of the Dist Frequent Next Neighbours algorithm are much lower than those of the Apriori distributed algorithm whatever the value of the minimum support set. We observe a degradation of the performance of the distributed Apriori algorithm.

6. Conclusion and Future Works

We presented a new approach for frequent itemsets extraction based on closed itemsets. The proposed algorithm Dist Frequent Next Neighbour is an extension of the Frequent Next Neighbour algorithm to big data. The fact that frequent closed itemsets are by nature reduced sets of frequent itemsets has allowed to obtain a significant reduction of the search space to be explored (reduction of the number of candidates to be considered at each iteration). The experimental results show an efficiency compared to the Spark-Apriori algorithm. However, we note that the concept computation phase is the most computationally intensive phase of our algorithm; this is explained by the intersection operations performed between the sets of objects and attributes which contain more and more elements. In our future work, we plan to improve the concept computation by optimizing or dispensing with the intersection computation between the sets of objects and attributes which is very consuming in terms of memory space.

References

[1] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. In : Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data;1993. p. 207-216.

[2] White EN. The stock market boom and crash of 1929 revisited. Journal of Economic Perspectives.1990;4(2):67-83.

[3] Ali K, Manganaris S, Srikant R. Partial Classification Using Association Rules. In : KDD; 1997; p. 115-118.

[4] Pazzani M, Mani S, Shankle RW. Comprehensible knowledge discovery in databases. In : Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society; 1997; p. 596-601.

[5] Agrawal R, Srikant R. Fast algorithms for mining association rules. In : Proc. 20th Int. Conf. Very Large Data Bases, VLDB; 1994; p. 487-499.

[6] Zaki MJ, Parthasarathy S, Ogihara M, Li W. Parallel algorithms for discovery of association rules. Data Mning and Knowledge Discovery. 1997;1(4):343-373.

[7] Han J, Pei J, Yin Y, Mao R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery. 2004;8(1):53-87.

[8] Pasquier N, Bastide Y, Taouil R, Lakhal L. Efficient mining of association rules using closed itemset lattices. Information Systems. 1999;24(1):25-49.

[9] Zaki MJ, Hsiao CJ. CHARM: An efficient algorithm for closed itemset mining. In : Proceedings of the 2002 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics; 2002; p. 457-473.

[10] Pei J, HAN J, MAO R. Closet: An efficient algorithm for mining frequent closed itemsets. In : ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery; 2000; p. 21-30.

[11] Zhang943. Spark apriori [Online]. 2018. Available from: https://github.com/zhang943/Spark-Apriori.git.

[12] Jatteau G. APPROXIMATION DU TREILLIS DE CONCEPTS POUR LA FOUILLE DE DONNÉES. 2005.

[13] Ganter B, Wille R. Contextual attribute logic. In : International Conference on Conceptual Structures. Springer, Berlin, Heidelberg; 1999; p. 377-388.

[14] Li H, Wang Y, Zhang D, Chang EY. Pfp: parallel fp-growth for query recommendation. In : Proceedings of the 2008 ACM Conference on Recommender Systems; 2008; p. 107-114.

[15] Yahya O, Hegazy O, Ezat E. An efficient implementation of apriori algorithm based on hadoop-mapreduce model. International Journal of Reviews in Computing. 2012;12:59-67.

[16] Moens S, Aksehirli E, Goethals B. Frequent itemset mining for big data. In : 2013 IEEE International Conference on Big Data. IEEE; 2013; p. 111-118.

[17] Qiu H, Gu R, Yuan C, Huang Y. Yafim: a parallel frequent itemset mining algorithm with spark. In : 2014 IEEE International Parallel & Distributed Processing Symposium Workshops. IEEE; 2014; p. 1664-1671.

[18] Zhang F, Liu M, Gui F, Shen W, Shami A, Ma Y. A distributed frequent itemset mining algorithm using Spark for Big Data analytics. Cluster Computing. 2015;18(4):1493-1501.

[19] Rathee S, Kashyap A. Adaptive-Miner: an efficient distributed association rule mining algorithm on Spark. Journal of Big Data. 2018;5(1):1-17.

[20] Carpineto C, Romano G. Concept data analysis: Theory and applications: John Wiley & Sons; 2004.