Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Spark solutions for discovering fuzzy association rules in Big Data

Carlos Fernandez-Basso, M. Dolores Ruiz [*], Maria J. Martin-Bautista

Department of Computer Science and A.I. and CITIC-UGR, University of Granada, Spain

ARTICLE INFO

Article history:
Received 2 September 2020
Received in revised form 7 May 2021
Accepted 5 July 2021
Available online 24 July 2021

Keywords:
Big Data
Fuzzy frequent itemset
Fuzzy association rules
Spark

ABSTRACT

The high computational impact when mining fuzzy association rules grows significantly when managing very large data sets, triggering in many cases a memory overflow error and leading to the experiment failure without its conclusion. It is in these cases when the application of Big Data techniques can help to achieve the experiment completion. Therefore, in this paper several Spark algorithms are proposed to handle with massive fuzzy data and discover interesting association rules. For that, we based on a decomposition of interestingness measures in terms of α-cuts, and we experimentally demonstrate that it is sufficient to consider only 10 equidistributed α-cuts in order to mine all significant fuzzy association rules. Additionally, all the proposals are compared and analysed in terms of efficiency and speed up, in several datasets, including a real dataset comprised of sensor measurements from an office building.

Nowadays, the increasing data generation in the majority of companies, social media, etc. has given rise to the Big Data phenomenon. Moreover, every day more buildings sum to the "smart sensored" fashion by incorporating sensor devices to collect data in order to use them for improving their energy usage and therefore save not only energy but also money and natural resources. The constantly necessity of information discovery from these huge amounts of data is the key in this competitive world. In this regard, Big Data philosophy, based on the MapReduce framework, helps in that task. In particular, many Data Mining techniques had been developed using Big Data techniques [1–4].
Association rules can be helpful in this ambit to uncover hidden relationships in gathered data without supervising. Association rules are often represented by implications of the type $A \rightarrow B$, representing the joint co-occurrence of A and B in a high percentage of transactions. However, numerical data coming e.g. from sensors may contain very granulated values that difficult their analysis. At this respect, some works have employed discretization of these numerical values. But this may bias the obtained results since final results can vary a lot depending on how the attribute values are divided. This problematic has been pointed out by several authors like in [5]. Fuzzy Sets theory [6] has been proven to be a good solution to discretize numerical values in a soften way and represent them in a understandable way to users. As a result of this fuzzy discretization process, fuzzy databases are created where fuzzy association rules can be extracted [7]. Additionally, there are fields where data can be affected by some kind of uncertainty or imprecision, and they are gathered following a fuzzy description of attributes [8].

In this paper, we study which can be the best procedure to analyse Big Data by means of fuzzy association rules. Non-distributive proposals can be found to perform such analysis, but they run into problems when the volume of data increases, becoming less efficient and leading, in many cases, to memory overflow errors. To this end, we have proposed

* Corresponding author.
E-mail address: cjferba@decsai.ugr.es (C. Fernandez-Basso).

https://doi.org/10.1016/j.ijar.2021.07.004

different Spark proposals for mining fuzzy association rules, enabling the distributed processing of massive datasets. Spark implementations [9] enables faster memory operations than other frameworks like Hadoop [10] because in-memory computations are allowed. In [11] there is a very complete comparison between Hadoop and Spark frameworks for different Machine Learning algorithms, obtaining that Spark outperforms Hadoop in the majority of cases.

Fuzzy association rules in the Big Data environment have not attracted yet much attention, and to the best of our knowledge, there does not exist any implementation in distributive environments for fuzzy association rules. Therefore, the main contribution of this work is then the proposal, implementation and comparison of different algorithms for fuzzy association rules mining in Spark, improving the existing implementations of crisp association rules algorithms, by enabling the uncovering of fuzzy association rules with no restrictions in either the number of items present in the antecedent or the consequent. The results of our experiments show that Big Data approach improves the efficiency of algorithms, with respect to non-distributed techniques. For that we have compared their performance in terms of execution time and also in memory consumption, attending to the number of transactions and the number of items to be analysed. However, the growth in the number of items, due to the exponential combination among them, the different proposals may not achieve a significant improvement in time, but thanks to the distribution capability of Big Data, substantial improvements are achieved in terms of memory problems, enabling to finish the analysis, in contrast to the memory overflow problems that sometimes appear in traditional approaches.

Additionally, our proposals are based on a decomposition of interestingness measures in terms of α-cuts which facilitates their implementation to other interestingness measures different to that of support and confidence using the formal model developed in [12]. In order to facilitate the decision of what set of α-cuts it is necessary to employ when mining fuzzy association rules, we experimentally demonstrate that it is sufficient to consider only 10 equidistributed α-cuts in order to uncover all significant fuzzy association rules.
These proposals employ elements of efficient search and seek to maximize the use of data cluster resources to improve computational efficiency and data processing capabilities.
The paper organization is the following: Section 1 makes a literature revision in the ambit of Big Data technologies and association rules mining algorithms. Section 2 introduces the definitions and concepts related to Big Data and fuzzy association rules mining necessary for the comprehension of the paper. Section 3 describes the different algorithms proposed and developed for fuzzy association rules mining using Spark. Section 4 presents the experiments performed and results obtained, before concluding the paper in Section 5.

1. Related work

Data mining techniques can be classified into two main types: supervised techniques such as classification methods and non-supervised techniques such as clustering or association rules. This work is focused in association rule mining, but there are also many works which restrict to frequent itemsets mining, the first step in the process of discovering association rules.

1.1. Algorithms review for frequent itemsets mining

In this section we review the existent algorithms in the literature for frequent itemset extraction. Once frequent itemsets are discovered, association rules can be extracted by assessing the strength of the relation (e.g. by means of confidence).

1.1.1. Apriori based algorithms

Apriori algorithm was proposed in the mid-nineties [13]. It consists on finding the set of frequent itemsets L, in a given database D. This algorithm searches the items in the transactions structure, which is consulted in each iteration to check if larger itemsets are satisfied. In this first work [13], the authors identified a fundamental property: the downward closure property. This property assures that any subset of a frequent itemset must be also frequent. This gave rise to divide the process in two main steps that are repeated to find frequent itemsets of length k: first one is known as the candidate generation step, in which the support of the corresponding k-itemsets is computed by scanning the transactional database, and second one is known as the large itemset generation, in which frequent $k+1$-itemsets are generated by pruning the candidate itemsets that do not exceed the minimum support threshold. These steps are then repeated until no more frequent itemsets can be found. Afterwards, an interestingness measure like the confidence is employed to discover the association rules, using the frequent itemsets extracted in the first step.
There are two limitations of this algorithm: one limitation comes from the complex itemset generation process which is very costly in time and memory, and the second limitation comes from the excessive number of scans necessary in the candidate generation step.
An improvement of Apriori is the algorithm called Apriori-TID [13]. This version of Apriori algorithm improves the performance of traditional Apriori algorithm with large databases. This approach is based on an itemset reduction by removing non-frequent itemsets in the database in each step, and has some changes with respect to the Apriori algorithm. The first change consists of sorting the transactions by item frequency, and removing the non-frequent items. The second change is that it uses the previous k candidate itemsets which were calculated in the previous phase to generate the itemsets of $k+1$ size.
This version improves the Apriori algorithm in small datasets, but for large datasets the performance of Apriori-TID is similar to Apriori although the use of memory is improved [14].

1.1.2. ECLAT TID-list based algorithm

Another way to improve the frequent itemset mining is by means of intermediate storage of data. In this case, a TID-list[1] is employed [15], where a binary list is created for each itemset, containing 1 in position j, if the item is satisfied in the transaction j and 0 otherwise.[2] This structure allows the computation of item and itemset supports using the boolean operators. The use of these operations improves the performance of the algorithm because boolean operations are performed very efficiently. One of the most known algorithms using this approach is the ECLAT [14][16] algorithm.
The main problem of this algorithm is that for a large number of items it would be necessary to store very large lists and the memory consumption would grow a lot. Additionally, the extension of this process in the distributed environment is not straightforward because each TID-List has a lot of dependencies with other TID-lists. However, the performance of ECLAT is faster in comparison with Apriori in the classical framework [17].

1.1.3. Frequent pattern FP-Growth algorithm

Another kind of algorithm for frequent itemset extraction is that using the FP-tree structure [18]. This algorithm called FP-Growth [19] uses the divide-and-conquer technique. In the FP-Growth algorithm the search space is decomposed on length-1 suffixes, reducing the number of searches in the database by using the representation of the data in a FP-tree structure.
This algorithm was originally designed for query recommendation where only top-k frequent itemsets are extracted. This issue is a drawback for association rules mining, because frequent itemsets of every length are needed to extract the association rules, unless we are interested in only the association rules with higher support. This means that FP-Growth is not exhaustive.

1.1.4. Algorithm comparison

In the literature different comparative studies of frequent itemsets algorithms can be found [20]. Amongst them we can emphasize works comparing the Apriori algorithm most widely used and known, to the rest of approaches. In [17] the Apriori algorithm is compared with ECLAT and it was appreciated how the use of the TID-lists improves the performance of the Apriori, although the use of memory is higher. Regarding the FP-Growth algorithm it scans the database of transactions only once, having thus a faster algorithm than Apriori [20]. However, one of the problems of FP-Growth is that, for very large datasets, the FP-tree may not fit in memory. Another feature to take into account, is that, it is not exhaustive, that is, FP-Growth does not obtain all the possible association rules since it does not generate all the possible frequent itemsets.
Moreover, in [21] there is a study comparing the most employed approaches, namely Apriori, ECLAT and FP-Growth. The experimental evaluation concludes that FP-Growth is more scalable and outperforms the others. In [22], a comparative study in the Big Data paradigm in a crisp framework is presented. As a conclusion, it shows that the distributed adaptation of FP-Growth is not always convenient to extract association rules, since it only provides the most frequent itemsets as well as their support. In our case, the algorithms of our proposal are exhaustive, finding all possible frequent itemsets. Hence, we propose the implementation and comparison of the Apriori, AprioriTID and ECLAT based algorithms in Spark to extract both frequent itemsets and fuzzy association rules in an exhaustive way.

1.2. Big Data algorithms for frequent itemset mining

The *MapReduce* framework employed in Big Data was designed by Google in 2003. Its foundation lies on two different functions, as its name indicates, to distribute the computation. In particular, the *Map()* function is in charge of transforming data into pairs of the type $(key, value)$ attending to some criteria that should be specified. And the *Reduce()* function is employed to aggregate those $(key, value)$ pairs containing the same key to finally obtain a piece of processed data according to the specified criteria.
When implementing MapReduce algorithms, two different frameworks arise as the most employed: Hadoop and Spark. These platforms have been improved in the last few years by incorporating diverse functions to fully take advantage of the capacity processing of a cluster, enabling to obtain thus more scalable algorithms and improving substantially traditional ways of cluster programming. In particular, for the case of association rules, within the Spark library, the PFP (Parallel FP-Growth) is included. This is a distributed adaptation of known FP-Growth algorithm that can be employed to extract higher level itemsets exceeding the minimum support threshold [23].
Other proposals for frequent itemsets mining that employ MapReduce techniques can be found in the literature for the non-fuzzy case. There exist some approaches which present Apriori-based algorithms using Hadoop: [24–26]. Note that these implementations are made in Hadoop, and according to the analysis made in [11], Spark accelerates executions since it enables in memory computations. Moreover, algorithms proposed in [24–26] do not employ data structures like tree, hash tree or hash table that can help to decrease execution times (see the study made in [27]).
In [28] and [29] the authors proposed the R-Apriori and YAFIM algorithms respectively, which are Apriori-based algorithms using Spark. These proposals can be compared to the non-fuzzy phase of our approach (made for each α-cut), but their

[1] TID stands for Transaction IDentifier.
[2] Another variant of TID-list consists on a list for each item containing the ids of the transactions where the item is satisfied.

proposal bases on a vectorial processing to obtain the itemsets of length k in the distributed process which uses a hash tree, while our proposal uses a hash table when applying MapReduce for every k-itemset. In addition, posterior analysis made in [27] which compares MapReduce implementations for different data structures concluded that using the hash table accelerates the algorithm performance versus using hash trees and tries (prefix trees).
Beside this, parallel adaptations of most used algorithms have been also developed for frequent itemsets mining [30] such as: ParEclat (Parallel Eclat) [31], Par-FP (Parallel FP-Growth with Sampling) [32], HPA (Hash Partitioned Apriori) [33]. All these algorithms use different data structures to improve the performance and take advantage of the potential of multi-processors. But the problem is that the multiprocessor it is not enough for all cases. When data exponentially increase, algorithms need more processing capability. For this reason distributed algorithms arise as a new option for frequent itemsets extraction.

1.3. Distributed algorithms for association rules mining

As described previously, most of the reviewed algorithms focus only on frequent itemset data extraction. Besides, in the literature as far as we know, no significant improvements have been described in the association rules creation phase, since the most time/memory consuming part is that of obtaining frequent itemsets. We can highlight the work of [34] where PEAR (Parallel Efficient Association Rules) algorithm is developed to improve the association rule mining step, but the global complexity is not decreased because the cost of extracting the frequent itemsets is much higher than in other algorithms. In [23] it is developed a distributed algorithm which extracts association rules but enabling only one item in the consequent part of the rule.
Conversely, in our approach, described in Section 3, it is presented an improvement of this part that enables the uncovering of association rules with no restrictions in either the number of items in the antecedent or the consequent.

1.4. Distributed algorithms for fuzzy association rule mining

The best to our knowledge, the only one work using the MapReduce framework for mining fuzzy association rules can be found in [35]. This proposal is based on an extension of the Count Distribution algorithm [36,37] to the fuzzy case. This algorithm uses similar procedure than R-Apriori [28] algorithm where in the second phase, in charge of computing the itemset support, Hadoop is employed instead of Spark. This approach differs from our proposal since we employ the resilient distributed dataset structure that enables across cluster computation.

1.5. Discussion

We have reviewed several types of frequent itemset extraction algorithms and association rules. As it can be seen, there are different options to improve the efficiency and computational capacity of these algorithms, on the one hand parallelisation which does not allow the full usage of computational power of modern processors to improve the efficiency of the algorithms. On the other hand, the new distributed computing tools, such as Spark, get profit of the whole processing capability to efficiently apply the algorithms.
This paper, and the proposals presented in it, focuses on the latter technology, by designing new algorithms based on distributed computing in large clusters to improve the capabilities of current algorithms for the extraction of fuzzy association rules. In this way, these new algorithms can be applied to massive data sets, as we will see in Section 4.

2. Preliminaries

2.1. Fuzzy association rules

Agrawal et al. [38] formally defined association rules for the first time, although in Observational Calculi [39,40] it was also investigated the analysis of associations.

In general, the Association Rule discovery problem consists in uncovering implications of the form $A \to B$ where A, B are itemsets from $I = \{i_1, i_2, \dots, i_m\}$ such that $A \cap B = \emptyset$ in $D = \{t_1, t_2, \dots, t_n\}$, a database formed by a set of n transactions each of them containing subsets of items (i.e. itemsets) from I. The right part of the rule, A, is often referred as the antecedent of the rule and the right part, B, as the consequent of the rule.

The problem of association rules discovery has two differentiated sub-tasks consisting on

- discovering all the itemsets exceeding the imposed threshold for the support, where support is defined as the percentage of transactions satisfying an itemset. These are known as frequent itemsets.
- Once frequent itemsets are obtained, association rules are those which exceed the minimum confidence threshold or another established assessment measurement (e.g. Lift).

However, as it was mentioned in the introduction, the data to be analysed can be diverse and can be numerical, categorical, imprecise, etc. For continuous numerical attributes, it is often applied a categorization process, for instance, the price of an object may be divided in different intervals indicating its belonging, like for example [100, 200]. However, depending on the definition of the intervals, the obtained associations can be very different. To prevent this, fuzzy linguistic labels appear as a solution to overcome this problem. In the previous example, a label like "expensive", that can be represented by a fuzzy set, could be a good option for the representation of the price of an object, and offering at the same time a meaningful and understandable semantic to the user [5]. Moreover, there can be occasions where crisp methods for describing data cannot be directly applied (see for example [41]).

In all these cases, the data has to be represented by a gradual value, leading to the concepts of fuzzy transaction and fuzzy association rule. Definitions of these concepts are taken from [42,7].

Definition 1. *A fuzzy transaction, t, is a non-empty fuzzy subset of I, where I is a set of items. That is, the membership degree of an item $i \in I$ in t is represented by a number in the range [0, 1] and denoted by $t(i)$.*

Note that this definition generalizes the idea of crisp transaction to the special case of fuzzy transaction. From now on, $\tilde{D}$ will denote a fuzzy transactional database (see for instance Table 1).

Definition 2. *Let A denote an itemset, i.e. a subset of items in I. The degree of membership of A in a fuzzy transaction $t \in \tilde{D}$ is defined as follows:*

$$t(A) = \min_{i \in A} t(i). \tag{1}$$

This means that $t(A)$ is the minimum of the membership degree of all its items.

Definition 3. *Let $A, B \subset I$ be itemsets in a fuzzy database $\tilde{D}$. Then, a fuzzy association rule $A \to B$ is satisfied in $\tilde{D}$ if and only if $t(A) \leq t(B)$ $\forall t \in \tilde{D}$, that is, the degree of satisfiability of B in $\tilde{D}$ is greater than or equal to the degree of satisfiability of A for all fuzzy transactions t in $\tilde{D}$.*

Different assessment measures have been proposed and analysed from different perspectives for fuzzy association rules (a good review can be found in [43]). The cardinality based generalization proposed in [7] and also generalized in works like [44,45], is a good option due to their good properties (see [7] and [46]). However, other measures arise using of the combination of particular inclusion and cardinality operators. The study made in [47] focuses in studying the suitable operators yielding a fuzzy Ruspini's partition in close relation with negated items in rules. For a deeper discussion on the possible frameworks to assess fuzzy association rules we recommend the review made in [43].

The support and confidence measures employed in this work are based on a semantic approach for the evaluation of quantified sentences [42]. This approach uses the GD-method [7] and the quantifier $Q_M(x) = x$, which represents the quantifier *"the majority"*. The following definitions can be found in [42,7].

Definition 4. *The support of a fuzzy itemset A is defined as:*

$$FSupp(A) = \sum_{\alpha_i \in \Lambda} (\alpha_i - \alpha_{i+1}) \frac{|\{t \in \tilde{D} : t(A) \geq \alpha_i\}|}{|\tilde{D}|} \tag{2}$$

where $\Lambda = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ with $\alpha_i > \alpha_{i+1}$ and $\alpha_{p+1} = 0$ is a set of α-cuts.

Definition 5. *The support of a fuzzy association rule $A \to B$ is defined as:*

$$FSupp(A \to B) = \sum_{\alpha_i \in \Lambda} (\alpha_i - \alpha_{i+1}) \frac{|\{t \in \tilde{D} : t(A) \geq \alpha_i \text{ and } t(B) \geq \alpha_i\}|}{|\tilde{D}|} \tag{3}$$

where $\Lambda = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ with $\alpha_i > \alpha_{i+1}$ and $\alpha_{p+1} = 0$ is a set of α-cuts.

Definition 6. *The confidence of a fuzzy association rule $A \to B$ is defined as:*

$$FConf(A \to B) = \sum_{\alpha_i \in \Lambda} (\alpha_i - \alpha_{i+1}) \frac{|\{t \in \tilde{D} : t(A) \geq \alpha_i \text{ and } t(B) \geq \alpha_i\}|}{|\{t \in \tilde{D} : t(A) \geq \alpha_i\}|} \tag{4}$$

where $\Lambda = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ with $\alpha_i > \alpha_{i+1}$ and $\alpha_{p+1} = 0$ is a set of α-cuts.

From these definitions leads that fuzzy association rules can be discovered by fixing a set of predefined α-cuts [44] to compute the support and confidence measures. Note that the computed measures will be closer to the real measure when considering every $\alpha \in [0, 1]$ appearing in the dataset if we fix a sufficiently dense set of α-cuts in the unit interval. This idea is behind of our proposal using MapReduce for fuzzy association rules mining. And we will see in the experiments that it is enough to consider 10 equidistributed α-cuts to obtain the whole set of fuzzy association rules.

Table 1
Fuzzy database example.

ID	A	B	C	D
1	0.75	0.15	0.35	0.1
2	0	0.5	0.2	0
3	0.8	0.4	0	0.45
4	1	0.25	0.85	1
5	0.5	1	0	0.8
6	0.3	0	0.75	0

3. Fuzzy association rule mining algorithms using Spark

In this section, we present three new algorithms for frequent itemsets extraction and the common part of these algorithms for fuzzy association rules (from now on FAR) mining, all of them following the Big Data paradigm for distributed-mode algorithms. These proposals are inspired by the operation of the traditional sequential Apriori, Apriori-TID and ECLAT algorithms and are all implemented using the Spark Framework, which has several facilities for developing Big Data algorithms based on MapReduce and improvements like the use of main memory or advanced DAG. In this regard, the implemented data structure in Apache Spark, called Resilient Distributed Dataset (RDD), abstracts the concept of data partition and will be employed through all our proposals, meaning that data are distributed across the clusters [48].
Before presenting the algorithms it is necessary to describe the following primitive Spark functions:

- *Map*: Applies a transformation function to each RDD and returns a transformed RDD. For instance, a Map function applied to $< key, value >$ pairs will give transformed pairs:

$$Map(< item, value_i >) \rightarrow < item, transformed\_value_i > \quad (5)$$

- *FlatMap*: Similar to *Map*, but each input item can be mapped to 0 (Equation (6) A), or to a pair (Equation (6) B) or different output items by means of auxiliary functions (Equation (6) C).

$$FlatMap(< item, value_i >) \rightarrow \emptyset \quad (6)$$

$$FlatMap(< item, value_i >) \rightarrow < itemset, value_j > \quad (7)$$

$$FlatMap(< item, value_i >) \rightarrow set(< itemset, value_j >) \quad (8)$$

- *Reduce*: Aggregates the elements of the dataset using an aggregation function. For example, the frequency of appearance of an item is computed by applying a *Reduce* function in the following way:

$$Reduce(< item, list(value) >) \rightarrow < item, value_{aggregated} > \quad (9)$$

- *Filter*: This function allows to filter the distributed data according to a condition.

Additionally, the algorithms use broadcast variables to enable access to global variables in every node of the cluster, i.e. broadcast variables are available in every partition performed by the Map functions.

In the following, we explain the different approaches proposed to extract frequent itemsets and fuzzy association rules using Big Data technologies.
Traditional algorithms problems when dealing with large amounts of data are mainly due to the multiple scans made of the whole database. This often raise in an increasing of the execution time along with the number of transactions. In our proposals, Spark is employed to improve the Apriori and ECLAT algorithms for mining fuzzy association rules. In both cases the data is stored using the HDFS (Hadoop Distributed File System), which permits replication and enables distributed processing.
In the following sections we present the different algorithms designed for extracting fuzzy association rules using Spark.

3.1. BDFARE-Apriori

This section is devoted to present the new algorithm, BDFARE-Apriori (Big Data Fuzzy Association Rules Extraction), which is comprised of different phases: preprocessing, phase 1, phase 2 and fuzzy association rule extraction.

The overall process is depicted in Algorithm 1. In that, we have employed the acronym DCS (Distribute Computing using Spark) to represent each chunk of data automatically created by Spark when distributing the data among the clusters. Each chunk is noted by S_i. This notation is also used in the subsequent algorithms.
For a better understanding of the running algorithms we will use the fuzzy database example in Table 1.

Algorithm 1 Main Spark procedure for BDFARE-Apriori algorithm.

```
1: Input: Data: Fuzzy RDD transactions: ${t_1, ..., t_n}$
2: Input: AlphaCuts: List of alpha cuts: ${\alpha_1, ..., \alpha_p}$
3: Input: MinSupp: minimum support threshold.
4: Output: Frequent itemsets exceeding $MinSupp$ (FreqItemset)
                              Preprocessing
5: DCS in $q$ chunks of Data: $\{S_1, ..., S_q\}$
6:     $BitArray_{S_i} \leftarrow S_i$.Map ($FuzzyToArray(t_k \in S_i)$) # Map function computes independently each transaction in $S_i$
                              Phase 1: FreqItems()
7: DCS in $q$ chunks of BitArray: $\{BA_1, ..., BA_q\}$
8:     $\{< it_1, bit\_list_1 >, ..., < it_m, bit\_list_m >\} \leftarrow BA_j$.FlatMap()
9:     $\{< it_1, card_1 >, ..., < it_m, card_m >\} \leftarrow$ ReduceByKey($|bit\_list_{it_k}|$)
10:    ItemSupport $\leftarrow$ Support($it_k$)
11:    DicFreqItemset $\leftarrow$ Filter($ItemSupport \geq MinSupp$)

                              Phase 2: Candidate generation
12: $Candidate \leftarrow DicFreqItemset$ #Candidates of $Length = 1$
13: $Length = 2$
14: $Global\_FreqItemset \leftarrow Candidate$
15: $broadcast(Global\_FreqItemset)$ #Creates a broadcast variable for its use across the cluster
16: do
17:    DCS in $q$ chunks of BitArray: $\{BA_1, ..., BA_q\}$
18:        $\{< itemset_1, bit\_list_1 >, ..., < itemset_m, bit\_list_m >\} \leftarrow$
           $BA_j$.FlatMap($BitListComputation(Global\_FreqItemset)$)
           #see Algorithm 3
19:        $\{< itemset_1, card_1 >, ..., < itemset_t, card_t >\} \leftarrow$
           ReduceByKey($|bit\_list_{itemset_k}|$)
20:        ItemsetSupport $\leftarrow$ Support($itemset_k$)
21:        $DicFreqItemset =$ Filter($ItemsetSupport \geq MinSupp$)
22:    end DCS computation
23:    $Length + +$
24:    $Candidate \leftarrow CandidateGen(DicFreqItemset, Length)$
25:    $Global\_FreqItemset.append(Candidate)$
26: while $|DicFreqItemset| > 1$
27: return $Candidate$
```

3.1.1. Preprocessing

The different algorithms proposed for fuzzy association rule mining algorithms have a first step to pre-process the data transforming them into an array of BitSets. Other works like [49] and [50], which use this kind of representation, have obtained very good results in terms of execution time and memory used. The BitSet representation has the advantage of accelerating logical operations such as conjunction or cardinality. This is an important part of the algorithm when calculating item conjunctions and their frequencies.

Therefore, for fuzzy association rules mining, the BDFARE-Apriori algorithm processes the data and stores them into an array of bit-lists whose size will depend on the number of transactions contained in the chunk, obtaining for each transaction an array of itemsets with their corresponding bit-list. This is described in lines 6-10 of Algorithm 1. Note that all this process is made in a distributive manner, i.e. for each chunk of data, in order to accelerate the computation. Then, for each transaction and for each item a bit-list will be created containing 1 in position j if the membership value of the item in that transaction is higher or equal than α_j and 0 otherwise. In this way, each item is represented by its bit-list depending on its value in the transaction. For that, we implement a procedure called $FuzzyToArray$, whose pseudocode is in Algorithm 2. For instance, if the itemset X is satisfied with degree 0.25 in a transaction t_i, i.e. $\mu_X(t_i) = 0.25$, and the set of α-cuts is $\{1, 0.75, 0.5, 0.25\}$, then the associated bit-list of X in that transaction will be $[0, 0, 0, 1]$.

The implementation of this type of bit-lists has been made using the *numpy* library available in python, which enables concurrent management of lists without going through every list element, enabling the distribution of computations along different clusters. Regarding the use of BitSets, it is necessary to emphasize the low memory resources needed to manage the array of bit-lists created during the preprocessing phase for big amounts of data (see also [12,49,50]). In addition, this structure improves the performance of the algorithm since it accelerates conjunction and cardinality computations.

3.1.2. Phase 1

The first step involves the dataset load and computation of each item appearance in the set of transactions. Fot that, it uses the Map and Reduce functions. In lines 7-11 of Algorithm 1 we can see how is the process of counting the items using the MapReduce paradigm. Firstly, we use a FlatMap function (see line 8) to transform the lists $[item_1\_bit\_list], \ldots, [item_m\_bit\_list]$ to lists of pairs of the form $< item_i, bit\_list_i >$. We perform this transformation for being able to use a $ReduceByKey$ function that adds all the Bitlists of the same key (i.e. of the same item) to obtain the frequencies of each item. In Fig. 1 we can see an example for the value of minimum support threshold of 0.5. In this example, the $FlatMap()$ function, returns the pairs $< item_i, bit\_list_i >$. Afterwards, the items are grouped using the $ReduceByKey$

Algorithm 2 Pseudocode of FuzzyToArray function.

```
1: Input: Data: a fuzzy transaction $t_k$.
2: Input: AlphaCuts: List of alpha cuts: $\{\alpha_1, \dots, \alpha_p\}$
3: Output: An array of Bit-lists for each item
                              FuzzyToArray()
4: For every item $it \in t_k$
5:     $j = 0$
6:     $Array = [\ ]$
7:     Do
8:         If($\mu_{it}(t_k) \geq \alpha_j$)
9:             Array.append(1)
10:        Else
11:            Array.append(0)
12:        $j++$
13:    While $j \leq p$
14: end for
15: return $\{Array\}$ # For each $t_k$ the Array contains lists of the type $<$item,bit_list$>$
```

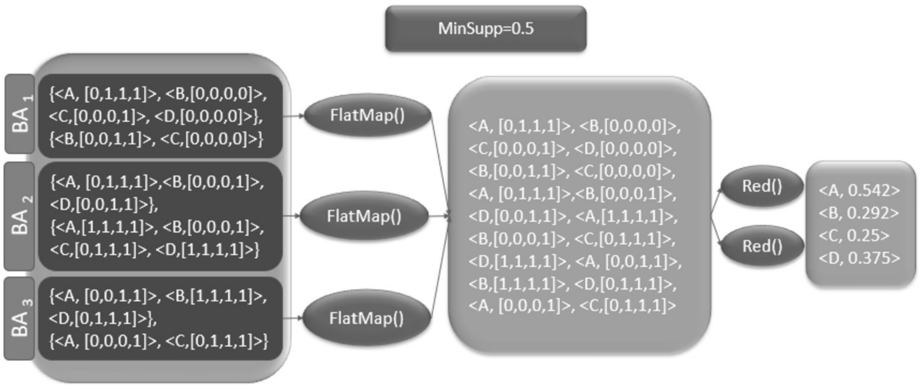
MinSupp=0.5
BA 1
{<A, [0,1,1,1]>, <B,[0,0,0,0]>, <C,[0,0,0,1]>, <D,[0,0,0,0]>}, {<B,[0,0,1,1]>, <C,[0,0,0,0]>}
BA 2
{<A, [0,1,1,1]>,<B,[0,0,0,1]>, <D,[0,0,1,1]>}, {<A,[1,1,1,1]>, <B,[0,0,0,1]>, <C,[0,1,1,1]>, <D,[1,1,1,1]>}
BA 3
{<A, [0,0,1,1]>, <B,[1,1,1,1]>, <D,[0,1,1,1]>}, {<A, [0,0,0,1]>, <C,[0,1,1,1]>}
FlatMap()
FlatMap()
FlatMap()
<A, [0,1,1,1]>, <B,[0,0,0,0]>, <C,[0,0,0,1]>, <D,[0,0,0,0]>, <B,[0,0,1,1]>, <C,[0,0,0,0]>, <A, [0,1,1,1]>,<B,[0,0,0,1]>, <D,[0,0,1,1]>, <A,[1,1,1,1]>, <B,[0,0,0,1]>, <C,[0,1,1,1]>, <D,[1,1,1,1]>, <A, [0,0,1,1]>, <B,[1,1,1,1]>, <D,[0,1,1,1]>, <A, [0,0,0,1]>, <C,[0,1,1,1]>
Red()
Red()
<A, 0.542>
<B, 0.292>
<C, 0.25>
<D, 0.375>

Fig. 1. Example of first phase of BDFARE algorithms applied to the fuzzy items in Table 1.

function using the operator of *numpy* library explained previously, and their support is calculated using the formula in equation (2). The last step is in charge of selecting those items with support greater than threshold ($MinSupp = 0.5$). In the example provided in Fig. 1 only item A fulfils that condition.

3.1.3. Phase 2

The second phase is in charge of extracting the size-k frequent itemsets (lines 12-26 of Algorithm 1). For this task we use different functions, for example the $CandidateGen()$ function generates the itemsets combinations for next k iteration of the algorithm. On the other hand, the function $BitListComputation()$ (see Algorithm 3) performs in a distributed way, the AND combination in each transaction of the candidate itemsets stored in the global variable *Global_FreqItemset* through its use in a FlatMap() function (see line 18 of Algorithm 1). The input of this FlatMap() is a chunk of the dataset transformed into bit-lists and the variable *Global_FreqItemset* containing the frequent itemsets of length k. Its output will be a list of pairs comprised of the key of the itemset and its corresponding *bit_list* for each transaction. The way to perform this task is to go through all the itemsets in *Global_FreqItemset* and search in the transaction each one of the items contained in the itemset (see lines 5-7 Algorithm 3). This returns a list with the obtained itemsets and their corresponding bit-list (where the AND operation has been applied to the bit-lists (see lines 8-10 Algorithm 3).
Afterwards, these pairs will be grouped by means of a function $ReduceByKey$ which calculates the cardinal of the bit-lists. This function will return a list with the items and their cardinal with which we will calculate the support of each one (see lines 19-21 of Algorithm 1). Then in line 21 the $DicFreqItemset$ variable is overwritten with the new candidates and the new frequent itemsets are added to the *Candidate* variable and if there is more than one new candidate a new iteration is made. These tasks are repeated until no new itemsets of length k can be found (see line 26).

As mentioned in Section 1, one of the main differences with other crisp approaches developed in Spark is the use of a hash table to accelerate the searching of itemsets. When other structures, like a linear search at each node, are employed, it results in an increase of time. A comparative study among these structures can be found in [27] where the experiments confirmed that the hash table surpassed the other data structures considered (hash tree, trie and hash table) for both real and synthetic datasets.

3.1.4. Fuzzy association rule mining

Once frequent itemsets are extracted, the final step is to uncover the fuzzy association rules that exceed the predetermined thresholds for support and confidence. This procedure is described in Algorithm 4. The result of previous phases

Algorithm 3 Pseudocode of BitListComputation function.

```
1: Input: Data: a fuzzy transaction $t_r$.
2: Input: Global_FreqItemset: List of $k$-Itemsets
3: Output: ItemsetList: List of pairs $< itemset, bit\_list >$.
                         BitListComputation()
4: ItemsetList=[ ]
5: For every $k$-itemset $A \in$ Global_FreqItemset
6:     $\{it_1, \dots, it_m\} =$ Split($A$)
7:     ArrayOfBitList: $\{bit\_list_1, \dots, bit\_list_m\}$ =SeekItems($\{it_1, \dots, it_m\}, t_r$)
8:     $bit\_list_A \leftarrow bit\_list_1 \wedge \cdots \wedge \cdots \wedge bit\_list_m$
9:     ItemsetList.append($< A, bit\_list_A >$)
10: end for
11: return ItemsetList
```

Algorithm 4 Main Spark procedure for extracting fuzzy association rules in BDFARE type algorithms.

```
1: Input: Candidate: Candidate list of frequent itemsets.
2: Input: MinConf minimum threshold for confidence.
3: Output: Fuzzy Association Rules exceeding $MinConf$
4: DCS in $r$ chunks of Candidate: $\{C_1, \dots, C_r\}$
5:     $\{< Rule_1, Conf_1 >, \dots, < Rule_s, Conf_s >\} \leftarrow$
       $C_j$.FlatMap(GenerateRules(), Conf()) # The FlatMap generate the
       rules using the list of candidates and computes their confidence using
       the frequency information in $Global\_FreqItemset$ variable
6:     $Rules \leftarrow$ ReduceByKey($RuleConf \geq MinConf$)
7:     return Rules
8: end DCS computation
```

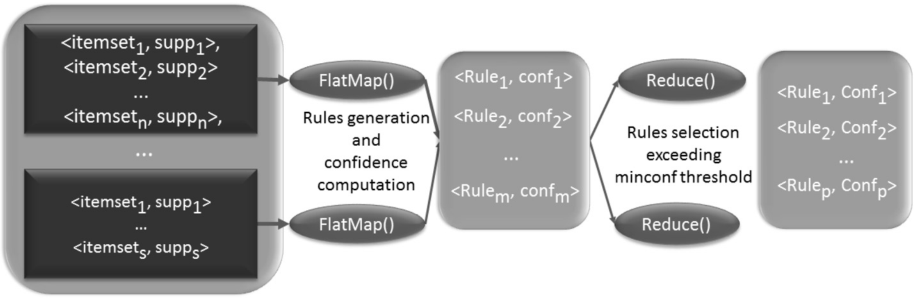

Fig. 2. General association rules mining procedure using MapReduce.

is a list of frequent itemsets that will be in RDD format used by Spark. The idea of this last part of the algorithm is to use a $FlatMap$ function and afterwords a $Reduce$ function as follows. For each partition of frequent itemsets the $FlatMap$ function generates the possible rules (see Fig. 2 and lines 4-5 of pseudocode of Algorithm 4). This task is performed by the $GenerateRules()$ function which generates rules from an itemset sequentially [51]. Therefore the distributed execution of $GenerateRules$ through $FlatMap$ returns pairs of the form $< Rule, Confidence >$, and finally, through a $ReduceByKey$ operation we filter those pairs keeping only those above $MinConf$ threshold (see lines 6-7 of Algorithm 4).

3.2. BDFARE-Apriori-TID

Another developed algorithm is BDFARE-Apriori-TID, design following the philosophy of Apriori-TID. In this case, the structure similar to the BDFARE-Apriori algorithm explained in the previous section.

3.2.1. Preprocesing

The preprocessing phase is equal to the BDFARE-Apriori algorithm, because the structure of data for distributive processing across the cluster is the same.

3.2.2. Phase 1

The first phase of Algorithm 5 is similar to Algorithm 1. The main difference is that those items with support value lower than the $MinSupp$ threshold are removed from the bit-list vector storage in the *Bit-Array* and sorted after the counting (see line 11). The ordering followed is from the highest to the lowest according to the number of occurrences of the item in the database.

Algorithm 5 Main Spark procedure for BDFARE-Apriori-TID algorithm.

```
1: Input: Data: Fuzzy RDD transactions: ${t_1, ..., t_n}$
2: Input: AlphaCuts: List of alpha cuts: ${\alpha_1, ..., \alpha_p}$
3: Input: MinSupp: minimum support threshold.
4: Output: Frequent itemsets exceeding $MinSupp$ (FreqItemset)
                                   Preprocessing
5: \\ Equal to preprocessing in Algorithm 1
                                   Phase 1: FreqItems()
6: DCS in $q$ chunks of BitArray: ${BA_1, ..., BA_q}$
7:     ${<it_1, bit\_list_1>, ..., <it_m, bit\_list_m>} \leftarrow BA_j$.FlatMap()
8:     ${<it_1, card_1>, ..., <it_m, card_m>} \leftarrow$ ReduceByKey($|bit\_list_{it_k}|$)
9:     ItemSupport $\leftarrow$ Support($it_k$)
10:    DicFreqItemset $\leftarrow$ Filter($ItemSupport \geq MinSupp$)
11:    $BitArray \leftarrow BA_j$.Map(Remove(DicFreqItemset)).
                 .Map(Sort(DicFreqItemset)).collect()
                                   Phase 2: Candidate generation
12: \\ Equal to lines 12-16 of Algorithm 1
13: do
14:    DCS in $q$ chunks of BitArray: ${BA_1, ..., BA_q}$
15:        ${<itemset_1, bit\_list_1>, ..., <itemset_m, bit\_list_m>} \leftarrow$
           $BA_j$.FlatMap($BitListComputation(Global\_FreqItemset)$)
           # see Algorithm 3
16:        ${<itemset_1, card_1>, ..., <itemset_t, card_t>} \leftarrow$
           ReduceByKey($|bit\_list_{itemset_k}|$)
17:        ItemsetSupport $\leftarrow$ Support($itemset_k$)
18:        $DicFreqItemset$ = Filter($ItemsetSupport \geq MinSupp$)
19:        $BitArray \leftarrow BA_j$.Map(Remove(DicFreqItemset))
20:    end DCS computation
21:    $Length++$
22:    $Candidate \leftarrow CandidateGen(DicFreqItemset, Length)$
23:    $Global\_FreqItemset.append(Candidate)$
24: while $|DicFreqItemset| > 1$
25: return $Candidate$
```

3.2.3. Phase 2

The principal difference with BDFARE-Apriori is that in each iteration of the frequent itemsets searching process all infrequent items are removed (see line 22 Algorithm 5).

3.2.4. Fuzzy association rule mining

The extraction rules phase is equal to the BDFARE-Apriori algorithm explained in Section 3.1.4. We use the hash table obtained with the frequent itemsets for extracting the association rules and calculate the measures like the confidence, lift or certainty factor of each rule.

3.3. BDFARE-ECLAT

We have also implemented the BDFARE algorithm following the philosophy of the sequential ECLAT algorithm, but using the MapReduce paradigm in the Spark framework. In this case, the principal difference resides in the data distribution by itemset, instead of process distribution by set of transactions like in Apriori.

3.3.1. Preprocesing

The preprocessing in BDFARE-ECLAT algorithm is different from the previous explained algorithms. In this way, BDFARE-ECLAT needs the data grouped by item joint with their membership degree (in the format of bit-list) (see Algorithm 6). Therefore the main difference is the aggregation by item in the MapReduce phase depicted in Fig. 3, where there is an example of the aggregation function and output data of BDFARE-ECLAT preprocessing.

Firstly, the algorithm transforms each transaction into pairs $< item_i, bit\_list_i >$ as explained in the preprocessing phase of BDFARE-Apriori algorithm. Then, the algorithm aggregates the different lists by items using a $GroupByKey$ function (see line 6 of Algorithm 6), generating pairs with an item and a large list of lists containing a bit-list per transaction.

3.3.2. Phase 1

In the first phase, the algorithm counts the number of appearances of every item in every transaction depending on the established set of α-cuts. After that, the algorithm calculates frequent items and generates the itemsets for the next step.
The main difference with BDFARE-Apriori resides in how to compute the frequency of items using the list data format. In this case the $FlatMap$ function (see line 8 of Algorithm 6) returns pairs comprised of items and their corresponding *bit-lists* per transaction. Then, the $ReduceByKey$ function calculates the cardinal per item with this data, and afterwards the support is computed (see lines 9-11 of Algorithm 6).

Algorithm 6 Main Spark procedure for BDFARE-ECLAT algorithm.

```
1: Input: Data: Fuzzy RDD transactions: $\{t_1, \dots, t_n\}$
2: Input: AlphaCuts: List of alpha cuts: $\{\alpha_1, \dots, \alpha_p\}$
3: Input: MinSupp: Minimum support threshold.
4: Output: Frequent itemsets exceeding $MinSupp$
                                   Preprocessing
5: DCS in $q$ chunks of Data: $\{S_1, \dots, S_q\}$
6:     $BitArray_{S_i} \leftarrow S_i$.Map(FuzzyToArray($t_k \in S_i$)).GroupByKey()
       # Map function computes independently each transaction in $S_i$
       # $BitArray_{S_i}$ contains lists of the form $<item,$
       $[bit\_list_1, \dots, bit\_list_n]>$, in this way, they can be distributed in the
       cluster by item (see Fig. 3)
                                   Phase 1: FreqItems()
7: DCS in $q$ chunks of BitArray: $\{BA_1, \dots, BA_q\}$
8:     $\{<it_1, [bit\_list_1, \dots, bit\_list_n]>, \dots, <it_m, [bit\_list_1, \dots, bit\_list_n]>\} \leftarrow$
       $BA_j$.FlatMap()
9:     $\{<it_1, card_1>, \dots, <it_m, card_m>\} \leftarrow$ ReduceByKey($|bit\_list_{it_k}|$)
10:    ItemSupport $\leftarrow$ Support($it_k$)
11:    DicFreqItemset $\leftarrow$ Filter($ItemSupport \geq MinSupp$)
                                   Phase 2: Candidate generation
12: $Candidate \leftarrow DicFreqItemset$ #Candidates of $Length = 1$
13: $Length = 2$
14: $Global\_FreqItemset \leftarrow Candidate$
15: $broadcast(Global\_FreqItemset)$ #Creates a broadcast variable for using across the cluster
16: do
17:    DCS in $q$ chunks of items list of BitArray: $\{IBA_1, \dots, IBA_q\}$
18:        $\{<itemset_1, bit\_list_1>, \dots, <itemset_m, bit\_list_m>\} \leftarrow$
           $IBA_j$.FlatMap($BitListComputation(Global\_FreqItemset)$)
           # see Algorithm 3
19:        $\{<itemset_1, card_1>, \dots, <itemset_t, card_t>\} \leftarrow$
           ReduceByKey($|bit\_list_{itemset_k}|$)
20:        ItemsetSupport $\leftarrow$ Support($itemset_k$)
21:        $DicFreqItemset =$ Filter($ItemsetSupport \geq MinSupp$)
22:    end DCS computation
23:    $Length++$
24:    $Candidate \leftarrow CandidateGen(DicFreqItemset, Length)$
25:    $Global\_FreqItemset.append(Candidate)$
26: while $|DicFreqItemset| > 1$
27: return $Candidate$
```

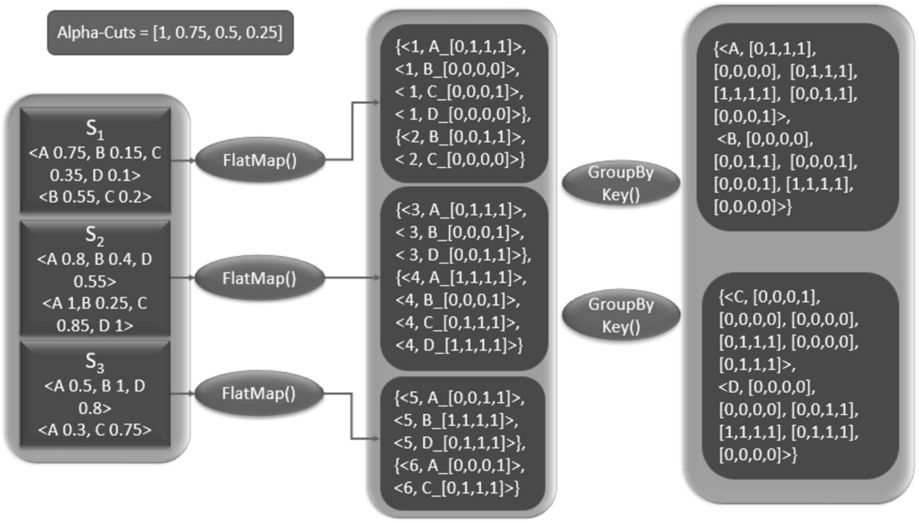
Alpha-Cuts = [1, 0.75, 0.5, 0.25]
S1
<A 0.75, B 0.15, C 0.35, D 0.1>
<B 0.55, C 0.2>
S2
<A 0.8, B 0.4, D 0.55>
<A 1,B 0.25, C 0.85, D 1>
S3
<A 0.5, B 1, D 0.8>
<A 0.3, C 0.75>
FlatMap()
FlatMap()
FlatMap()
{<1, A_[0,1,1,1]>, <1, B_[0,0,0,0]>, < 1, C_[0,0,0,1]>, < 1, D_[0,0,0,0]>}, {<2, B_[0,0,1,1]>, < 2, C_[0,0,0,0]>}
{<3, A_[0,1,1,1]>, < 3, B_[0,0,0,1]>, < 3, D_[0,0,1,1]>}, {<4, A_[1,1,1,1]>, <4, B_[0,0,0,1]>, <4, C_[0,1,1,1]>, <4, D_[1,1,1,1]>}
{<5, A_[0,0,1,1]>, <5, B_[1,1,1,1]>, <5, D_[0,1,1,1]>}, {<6, A_[0,0,0,1]>, <6, C_[0,1,1,1]>}
GroupBy Key()
GroupBy Key()
{<A, [0,1,1,1], [0,0,0,0], [0,1,1,1], [1,1,1,1], [0,0,1,1], [0,0,0,1]>, <B, [0,0,0,0], [0,0,1,1], [0,0,0,1], [0,0,0,1], [1,1,1,1], [0,0,0,0]>}
{<C, [0,0,0,1], [0,0,0,0], [0,0,0,0], [0,1,1,1], [0,0,0,0], [0,1,1,1]>, <D, [0,0,0,0], [0,0,0,0], [0,0,1,1], [1,1,1,1], [0,1,1,1], [0,0,0,0]>}

Fig. 3. Preprocessing data phase which transforms fuzzy items into bit-lists in BDFARE-ECLAT.

3.3.3. Phase 2

For the calculation of itemsets support, the algorithm uses the $FlatMap()$ function in the same way as in the Apriori case. The main difference is how the pair lists are computed. In this case, the $FlatMap()$ function returns a pair comprised of an itemset and a list for each transaction with the *bit-list* using the *numpy* array format (see Fig. 4 and lines 18-19 of Algorithm 6). Therefore, the algorithm calculates all the cardinalities needed for the computation of the cardinality of the

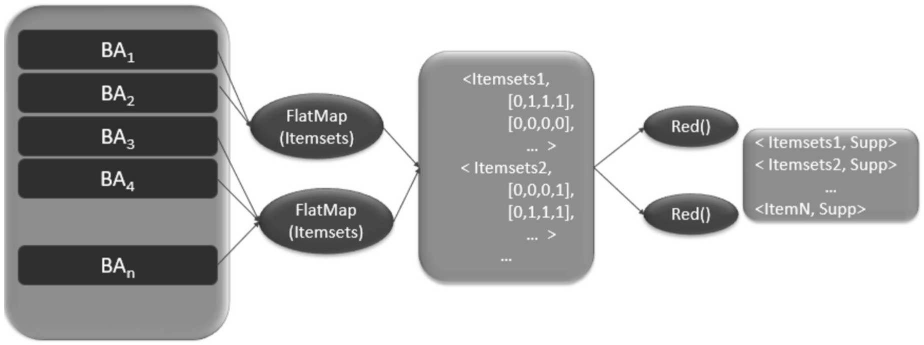

Fig. 4. Example of second phase of BDFARE-ECLAT applied to the fuzzy items in Table 2.

obtained itemsets using the $ReduceByKey()$ function, and afterwards computes their support. This step will be repeated until there are not new frequent k-itemsets.

3.3.4. Fuzzy association rule mining

The extraction rules phase is equal to BDFARE-Apriori and BDFARE-Apriori-TID previously explained.

4. Experiments and results

Our aim is to study the behaviour of the new algorithms designed for distributed computation using Spark framework. The following experiments have been designed to analyze the following different aspects:

- Compare proposed algorithms among them taking into account the execution time along different configurations for the experiments, and towards different datasets.
- Compare the proposed algorithms for the calculation of measures like support and confidence when the algorithms use $numpy$ vector or traditional method.
- Analyse proposed algorithms measuring the speed up and the efficiency achieved by increasing the number of cores in the procedure.
- Study the performance of proposed algorithms with respect to different sets of α-cuts.

To this end, the algorithms have been tested in several fuzzy transactional datasets to analyse their execution time attending to different parameters: the number of items, the length of datasets, the number of transactions, and the number of α-cuts. Nine different datasets have been considered. Four of them from the UCI Machine Learning repository[3] where some continuous attributes have been conveniently fuzzyfied using trapezoidal labels according to the semantics of the attributes, as described in [45] and in Fig. 5. The `German` dataset is comprised of transactions about credits offered by a german bank. Three variables were fuzzyfied: amount of the credit, its duration and the age of the person who owns the credit. The `Autompg` dataset comprises several attributes about cars. In this case, the continuous attributes were fuzzified using the following linguistic labels: low, medium and high. The `Bank`[4] dataset contains marketing data from different campaigns of a Portuguese banking institution. The `Higgs` [52] dataset is comprised of several attributes about Higgs Boson produced during several Monte Carlo simulations. It contains 28 continuous attributes plus the class attribute. The continuous attributes have been fuzzified using three different labels: low, medium and high as it can be seen in Fig. 5, and the class attribute remains the same containing the values 1 or 0.

The `Forest-equidepth` database is originated from the database used in [53] where binary attributes were excluded and we have fuzzified the remaining attributes using equi-depth intervals explained in [54].

A new dataset obtained from social networks has been also used, more precisely a dataset obtained from twitter about the 2016 American elections. The content of the tweets has been fuzzified using the TF-IDF[5] that gives values in the unit interval for those words appearing in the tweets.

For the reminder of datasets, a different strategy for fuzzification of attributes has been followed, using the automatic process described in [55]. The `Energy ICPE` dataset, comprises data from an office building in Romania located in Bucharest. It comprises 273 sensors containing different metering data that were collected during a year (from September 2016 to September 2017) with a total of 3,649,678 transactions. Other dataset with energy related data is `FARO`. This dataset is comprised of sensors for heating, air conditioning, lights and power consumption, taken in the FARO airport, useful to analyse the behaviour of the building in terms of energy consumption. This dataset contains data from 64 sensors that have

[3] http://archive.ics.uci.edu/ml/.
[4] http://ugritlab.ugr.es:82/cjferba/datasets.
[5] TF-IDF stands for term frequency–inverse document frequency. It is a measure often used when analysing the frequency of terms in text documents.

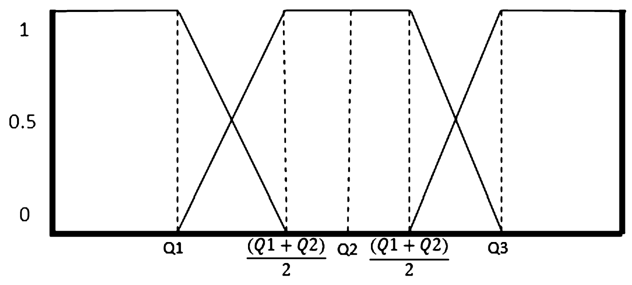

Fig. 5. Fuzzyfication of Higgs features using quartiles.

Table 2
Datasets.

Fuzzy Database	Transactions	Fuzzy Items	Supp	Conf
`Bank`	1 446 752(45 211)	112	0.2	0.8
`Higgs`	11 000 000	86	0.2	0.8
`Forest-equidepth`	581 012	37	0.2	0.8
`Energy ICPE`	3 649 678	1 121	0.4	0.8
`Energy FARO`	5 693 452	246	0.05	0.8
`Tweets for EEUU elections`	100 000	1 478	0.1	0.8
`ASHRAE`	20 200 000	109	0.1	0.8
`Autompg`	1 630 208(398)	39	0.05	0.8
`German`	1 000 000(1000)	79	0.05	0.8
`Soildb`	1 000 000(542)	33	0.1	0.8

been fuzzified with the same methodology followed in the previous dataset. It is composed of 5 693 452 transactions corresponding to the summer season (from 1st of June to 15th of September) of 2017. The `ASHRAE` energy predictor dataset can be obtained from the public Kaggle repository.[6] This dataset contains different datasets involving time, consumption and building type. We have merged them using the location and timestamp, as well as removing some of the features that were not going to be of interest such as the building ID or some of the weather features. Then the variables have been fuzzified. After this process, the resulting dataset contains 20 million records and 109 fuzzified items.

In order to test the behaviour of the algorithms with more datasets, those datasets that were not sufficiently big have been replicated in order to have a more extensive experimentation. This is propitiated due to the lack of massive datasets with fuzzy data in the literature. These are `Bank`, `German`, `Soildb` and `Autompg`, whose original number of transactions is the one framed in parentheses (see Table 2). The experimental evaluation has been made in a cluster comprised of 4 servers with 102 cores and 420 Gb of RAM where intel's hyperthreading functionality was disabled for testing. The Spark version employed was 2.2 which uses a fully distributed mode with Ambari Server.

We have made several experiments with different threshold values $Minsupp \in \{0.03, 0.05, 0.1, 0.2, 0.4\}$ and $Minconf \in \{0.8\}$. The high values fixed for the minimum support have been chosen to reduce the number of obtained rules, because some of the fuzzy items are very frequent in some of the databases, leading to a huge amount of discovered rules. The idea of these thresholds is to have a comparable set of obtained fuzzy rules (in terms of number of rules).

Additionaly, 10 alpha cuts have been selected in all the experiments because the results of the algorithms using 10, 50 and 100 are practically the same, changing only a little the precision of the computations for support and confidence.

Fig. 7 shows the number of fuzzy association rules obtained for the different datasets in Table 2. These values differ from dataset to dataset due to the nature and composition of the datasets. Minimum support values have been chosen to obtain a "similar" number of rules in each of the examples, taking into account their characteristics. For example, in the case of the Energy ICPE dataset there is a high frequency of repeated values in the sensors as well as a large number of items, so the combination of these items is much higher, leading to a higher number of association rules. In the particular case of the Energy FARO dataset, where the sensor values have very low frequencies, the support is fixed lower.

We are interested in analysing the performance of every algorithm according to the variation of the following: the number of transactions, the number of items and the set of α-cuts. In addition to this, note that the proposed algorithms extract fuzzy association rules without restricting the number of items appearing in the consequent or the antecedent of the rules.

We have also analysed the behaviour of the algorithms when the number of resources (cores and memory) increases. Figs. 8 to 16 show the behaviour of the algorithms when the resources (1 core means sequential and 24, 48, 72, 102 is equivalent to 1, 2, 3, 4 nodes where each node has 100 Gb of RAM) are increased in each of the datasets. It can be observed that BDFARE-Apriori-TID is the most efficient, and BDFARE-Apriori performed better than BDFARE-ECLAT.

[6] https://www.kaggle.com/c/ashrae-energy-prediction.

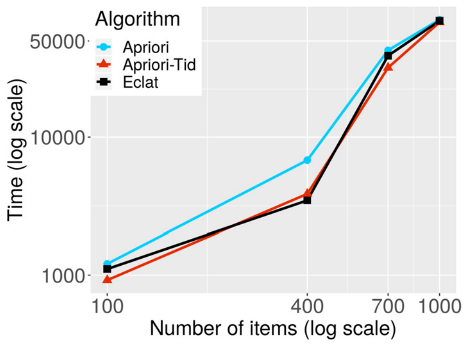

Fig. 6. Performance in logarithmic scale of BDFARE algorithms for `Energy ICPE` dataset when the quantity of items increases.

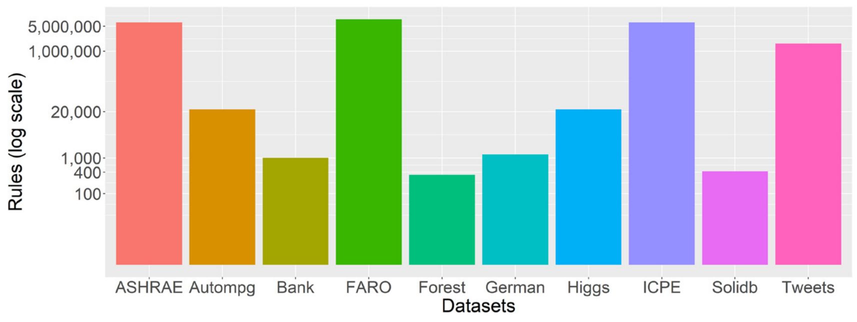

Fig. 7. Number of fuzzy association rules extracted for each dataset.

Table 3
Table of results (in seconds) obtained by the algorithms in their sequential and distributed version.

Dataset	sequential Apriori	BDFARE Apriori	sequential Apriori-TID	BDFARE Apriori-TID	sequential Eclat	BDFARE ECLAT
German	983	598	908	294	1 652	978
Bank	1 542	492	1 050	280	1 780	890
Autompg	1 309	393	1 220	212	1 509	726
Forest	2 307	300	2 107	240	2 207	270
Higgs	8 986	546	7 829	440	-	1 892
ASHRAE	-	30 172	-	22 819	-	-
FARO	-	61 029	-	47 829	-	77 281
ICPE	-	71 400	-	68 400	-	69 600
Tweets	-	128 514	-	118 179	-	119 028

Moreover, BDFARE-ECLAT has some memory problems in `Higgs`, `ASHRAE` (see Fig. 11) and `Energy ICPE` dataset, because these datasets have a lot of transactions and items, and therefore the lists that BDFARE-ECLAT has to create are very big. For this reason it needs to use a big amount of memory. By contrast, BDFARE-Apriori and BDFARE-Apriori-TID algorithms completed all the executions.

To be exhaustive, a detailed description of running times can be found in Table 3. It can be seen that in those datasets with 1 million and 11 million of transactions, the algorithms achieved in average time savings of 40% and 60% respectively if they are compared to the non-distributed case (1 core). In addition, in the sequential versions of several of the algorithms we encountered memory problems such as for `Higgs`, `Twiter`, `FARO`, `ICPE` and `ASHRAE` datasets. This is caused by the large number of transactions or items that require more memory and computational capacity to deal with the dataset. In the table of results (see Table 3) we can see marked with the sign "-" those executions that could not finished.

Fig. 6 shows the time spent by the algorithms when the number of items increases for `Energy ICPE` dataset. In this graph it can be observed that BDFARE does not offer substantial improvements in terms of time, because the increment of time is exponential by the number of items. This is due to the items combinatorial explosion of Apriori-based algorithms. Additionally, BDFARE does not achieve always more efficient executions, because during the jobs planning, necessary when distributing data, there is also a waste of time. However the performance of the BDFARE algorithm tends to improve when the number of transactions increases. This is because the distribution of data across the clusters is made by transaction. Moreover we can see that the BDFARE-ECLAT algorithm, although it uses a list of items, the performance is similar to BDFARE-Apriori. This is because the use of long lists of elements in each node does not work efficiently.

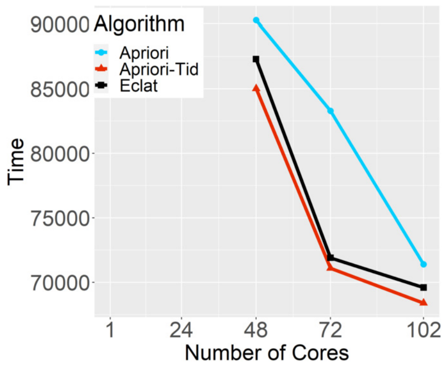

Fig. 8. Time for different number of clusters for `Energy ICPE` dataset.

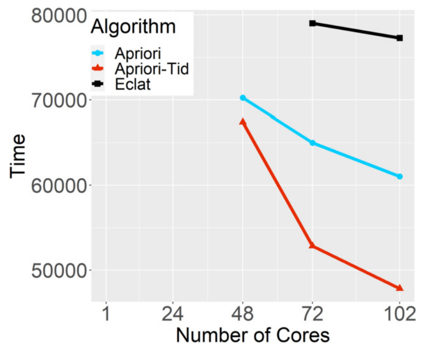

Fig. 9. Time for different number of clusters for `Energy FARO` dataset.

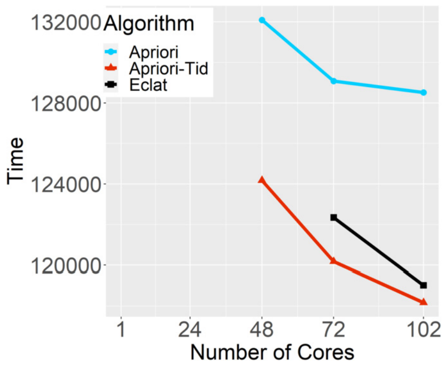

Fig. 10. Time for different number of clusters for `Twiter` dataset.

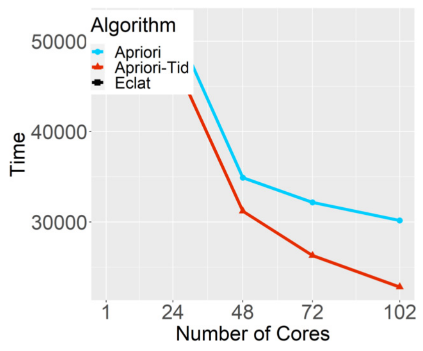

Fig. 11. Time for different number of clusters for `ASHRAE` dataset.

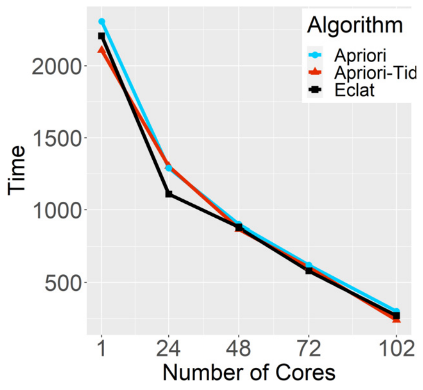

Fig. 12. Time for different number of clusters for `Forest-equidepth` dataset.

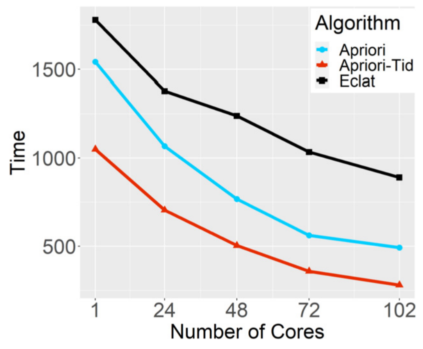

Fig. 13. Time for different number of clusters for `Bank` dataset.

For the analysis of the *speed up* and the *efficiency* [56–58] when different number of cores are used, the known speed up measure has been employed, defined as [58,59]

$$S_n = T_1/T_n \tag{10}$$

where T_1 represents the time spent by the sequential algorithm and T_n the time of the distributed algorithm with several cores. To complement the speed up, the efficiency measure [56–58] is defined as

$$E_n = S_n/n = T_1/(n \cdot T_n) \tag{11}$$

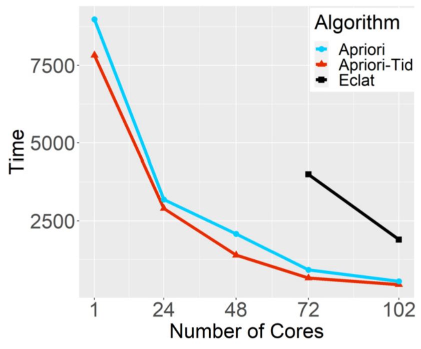

Fig. 14. Time for different number of clusters for `Higgs` dataset.

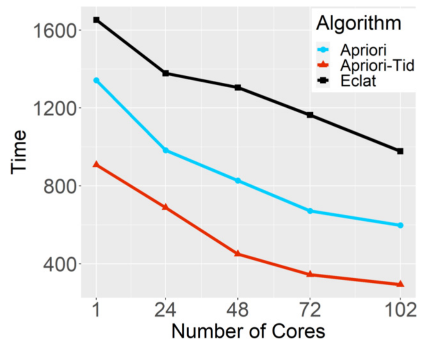

Fig. 15. Time for different number of clusters for `German` dataset.

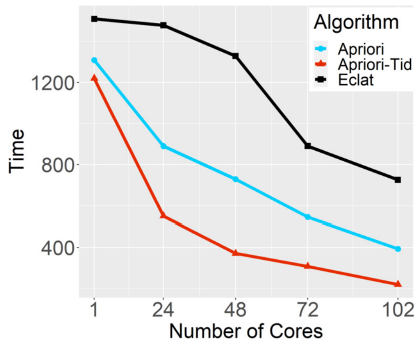

Fig. 16. Time for different number of clusters for `Autompg` dataset.

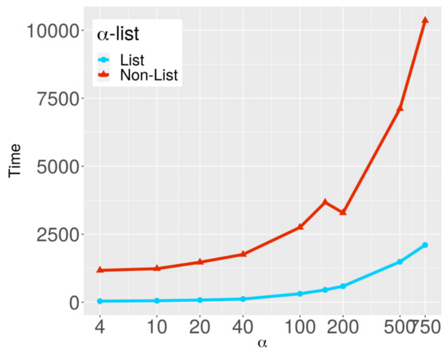

Fig. 17. Performance in logarithmic scale of BDFARE-Apriori-TID algorithm when the quantity of α-cuts increases in `forest-equidepth` dataset.

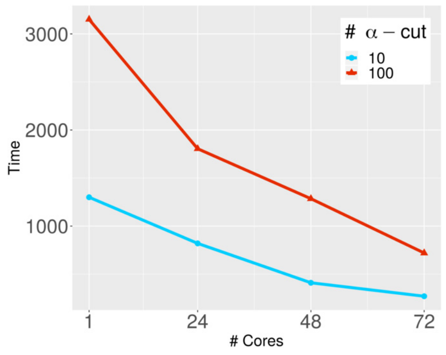

Fig. 18. Performance in logarithmic scale of BDFARE-Apriori-TID algorithm with 10 and 100 α-cuts when the quantity of cores increases (from 1 core=sequential to 32 cores) in `Bank` dataset.

In Figs. 19 and 20 it can be seen that as the numbers of cores increase, the efficiency and speed up are improved, even they are not optimal. This factor is influenced by the cores workloads and also by the network congestion caused by the communication among the cores.

Fig. 19 shows the speed up of the different proposals. In this figure, it can be observed the evolution of the execution times having the greatest reduction when the number of processors is higher. Although the speed up obtained increased along the number of processors used, we can see that it moves away from proportional speed up as resources expand. This is because increased resources may not be used as efficiently with respect to the same amount of data using fewer resources. Note also that there are parts of the algorithm that are iterative (e.g. calculation of the itemset of size $k+1$ needs to execute those of length k), so certain parts are not totally distributed and this actually affects efficiency. As far as we can see in Fig. 20 the efficiency does not increase proportionally. This same behaviour of efficiency and speed up has been observed in other studies of distributed algorithms where the efficiency does not increase proportionality with more processors [60,58].

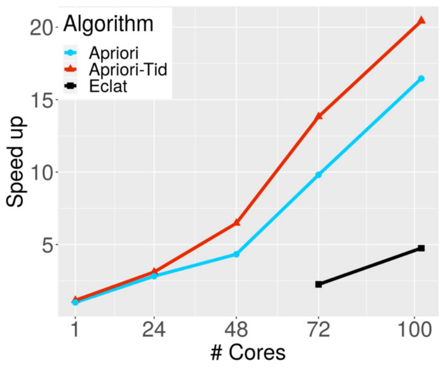

Fig. 19. Speed up of BDFARE-Apriori-TID algorithm for `Higgs` dataset.

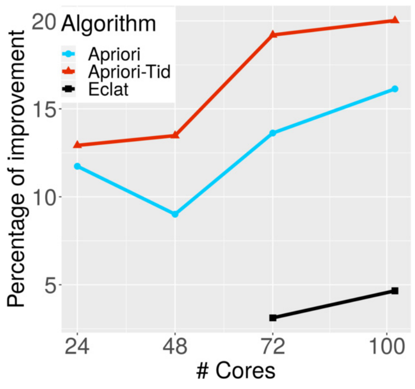

Fig. 20. Efficiency of BDFARE-Apriori-TID algorithm for `Higgs` dataset measured by percentage of improvement.

Regarding the performance of BDFARE algorithms with respect to the set of α-cuts we have performed different experiments. Fig. 18 describes how BDFARE-Apriori-TID algorithm performs when it uses lists to represent α-cuts and the number of α-cuts is increased, using from 1 to 16 cores (1, 24, 8, 16 cores). The difference of the performance when the algorithm uses lists or not for different number of α-cuts is noticeable, concluding that using lists improves the efficiency.

Moreover, in Fig. 17 it can be observed the performance when the quantity of α-cuts increases. In that experiment, we have observed that the number of rules found considering 100, 50 α-cuts and 10 α-cuts is the same (in all the experiments). Then, it seems that using only 10 α-cuts it is sufficient to achieve the precision of computations to retrieve all the fuzzy association rules, which will decrease the number of computations and therefore, the efficiency of the BDFARE algorithms.

5. Conclusions and future research

This paper has proposed different fuzzy association rules mining algorithms in the ambit of Big Data for the discovery of co-occurrence patterns from fuzzy datasets. It has been shown, that non-distributed algorithms proposed for mining fuzzy association rules can fail when handling massive datasets due to the memory overflow errors and their efficiency is affected when the dataset grows.

To this end, the proposals presented, which uncovers fuzzy association rules using the Spark framework, are capable of analysing massive data. To prove this, the different proposals have been compared and analysed obtaining improvements not only in the running time, but also in their memory, enhancing their processing capacity (note that some of the experiments could not finish their execution in the non-distributed cases). An additional advantage of Big Data proposals is that their performance can be easily improved just by expanding the system by adding more computation nodes or clusters. This makes easier to improve the scalability of our proposals, enabling also their execution in external cloud systems like for instance Amazon Web Services.

These advantages are obtained not only in terms of execution times and efficiency, but also in terms of the ability to apply these algorithms to large data sets. For example, applying it to large sets of sensors, social network data or cybercrime advertisements on the darknet. All of these types of datasets contain large amounts of transactions that are impossible to tackle with traditional algorithms.

Additionally, our proposal is based on a decomposition of interestingness measures in terms of α-cuts which facilitates their implementation to other interestingness measures different to that of support and confidence using the formal model developed in [12], and we have experimentally demonstrated that it is sufficient to consider only 10 equidistributed α-cuts in order to mine all significant fuzzy association rules.

Currently, there is not any large fuzzy dataset available in open data repositories. For this reason, as for this article we have created different datasets by fuzzifying the items, as future work we intend to create a repository to share collections of large data sets containing fuzzy data.

As regards future research, we plan the application of these proposals to real world problems such as in the analysis of social media or in the energy field, by the combination of other Data Mining and Machine Learning techniques in order to obtain more valuable knowledge from the data, utilising association rules as a first exploratory step in the knowledge discovery process.

Lastly, we also intend to generalise these Big Data procedures to other techniques that are formulated in terms of association rules such as gradual dependencies [61,62] or exception and anomalous rules [63].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The research reported in this paper was partially supported the COPKIT project from the 8th Programme Framework (H2020) research and innovation programme (grant agreement No 786687) and from the BIGDATAMED projects with references B-TIC-145-UGR18 and P18-RT-2947.

References

[1] S. del Río, V. López, J.M. Benítez, F. Herrera, On the use of MapReduce for imbalanced big data using random forest, Inf. Sci. 285 (2014) 112–137, Processing and Mining Complex Data Streams.
[2] D. Anastasiu, J. Iverson, S. Smith, G. Karypis, Big data frequent pattern mining, in: Frequent Pattern Mining, Springer International Publishing, Switzerland, 2014, pp. 225–259.
[3] A. Fernández, C. Carmona, M. del Jesus, F. Herrera, A view on fuzzy systems for Big Data: progress and opportunities, Int. J. Comput. Intell. Syst. 9 (2016) 69–80.
[4] C. Fernandez-Basso, M. Ruiz, M. Martin-Bautista, Extraction of association rules using Big Data technologies, Int. J. Des. Nat. Ecodyn. 11 (3) (2016) 178–185.
[5] E. Hüllermeier, Y. Yi, In defense of fuzzy association analysis, IEEE Trans. Syst. Man Cybern., Part B, Cybern. 37 (4) (2007) 1039–1043.
[6] L. Zadeh, Fuzzy sets, Inf. Control 8 (1965) 338–353.
[7] M. Delgado, N. Marín, D. Sánchez, M. Vila, Fuzzy association rules: general model and applications, IEEE Trans. Fuzzy Syst. 11 (2) (2003) 214–225.
[8] J. Calero, G. Delgado, M. Sánchez-Marañón, D. Sánchez, J. Serrano, M.A.V. Miranda, Helping user to discover association rules: a case in soil color as aggregation of other soil properties, in: ICEIS 2003, Proceedings of the 5th International Conference on Enterprise Information Systems, Angers, France, April 22-26, 2003, 2003, pp. 533–540.
[9] X. Meng, et al., MLlib: machine learning in Apache Spark, J. Mach. Learn. Res. 17 (2016) 1–7, http://arxiv.org/abs/1505.06807.
[10] T. White, Hadoop: The Definitive Guide, fourth edition, O'Reilly, 2015.
[11] L. Liu, Performance comparison by running benchmarks on Hadoop, Spark and Harm, Ph.D. thesis, University of Delaware, 2016, http://udspace.udel.edu/bitstream/handle/19716/17628/2015_LiuLu_MS.pdf?sequence=1.
[12] M. Delgado, M.D. Ruiz, D. Sánchez, J.-M. Serrano, A formal model for mining fuzzy rules using the RL representation theory, Inf. Sci. 181 (23) (2011) 5194–5213.
[13] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proc. of the Twentieth Inter. Conf. on Very Large Databases, Santiago, Chile, 1994, pp. 487–499.
[14] J. Hipp, U. Güntzer, G. Nakhaeizadeh, Algorithms for association rule mining - a general survey and comparison, ACM SIGKDD Explor. Newsl. 2 (1) (2000) 58–64.
[15] M.J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, et al., New algorithms for fast discovery of association rules, in: KDD, vol. 97, 1997, pp. 283–286.
[16] M.J. Zaki, Scalable algorithms for association mining, IEEE Trans. Knowl. Data Eng. 12 (3) (2000) 372–390.
[17] C. Borgelt, Efficient implementations of Apriori and Eclat, in: FIMI'03: Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.
[18] C.R.J. Li, Z.H. Deng, Mining frequent ordered patterns without candidate generation, in: FSKD 2007, ACM Press, New York, 2007, pp. 402–406, arXiv:cond-mat/0611061v2, http://portal.acm.org/citation.cfm?doid=342009.335372.
[19] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: SIGMOD'00 Proc. of the 2000 ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
[20] Z. Zheng, R. Kohavi, L. Mason, Real world performance of association rule algorithms, in: Proc. of the Seventh ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM, 2001, pp. 401–406.
[21] K. Garg, D. Kumar, Comparing the performance of frequent pattern mining algorithms, Int. J. Comput. Appl. 69 (25) (2013).
[22] C. Fernandez-Basso, M. Ruiz, M. Martin-Bautista, A comparative analysis of Spark frequent itemsets and association rule mining algorithms, Knowl.-Based Syst. (2020), submitted for publication.
[23] H. Li, Y. Wang, D. Zhang, M. Zhang, E.Y. Chang, PFP: parallel FP-growth for query recommendation, in: Proc. of the 2008 ACM Conference on Recommender Systems, ACM, 2008, pp. 107–114.
[24] N. Li, L. Zeng, Q. He, Z. Shi, Parallel implementation of Apriori algorithm based on MapReduce, in: Proc. of the 2012 13th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 236–241.
[25] Z. Farzanyar, N. Cercone, Efficient mining of frequent itemsets in social network data based on MapReduce framework, in: Proc. in ASONAM 2013, 2013, pp. 1183–1188.
[26] Z. Farzanyar, N. Cercone, Accelerating frequent itemset mining on the cloud: a MapReduce-based approach, in: IEEE 13th Int. Conf. on Data Mining Workshops, 2013, pp. 592–598.
[27] S. Singh, R. Garg, P. Mishra, Performance analysis of Apriori algorithm with different data structures on Hadoop cluster, Int. J. Comput. Appl. 128 (9) (2015) 45–51.
[28] S. Rathee, M. Kaul, A. Kashyap, R-Apriori: an efficient Apriori based algorithm on Spark, in: Proc. of the PIKM'15, ACM, Melbourne, VIC, Australia, 2015.
[29] H. Qiu, R. Gu, C. Yuan, Y. Huang, YAFIM: a parallel frequent itemset mining algorithm with Spark, in: Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, IEEE, 2014, pp. 1664–1671.
[30] R. Agrawal, J.C. Shafer, Parallel mining of association rules, IEEE Trans. Knowl. Data Eng. 8 (6) (1996) 962–969.
[31] M.J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, Parallel algorithms for discovery of association rules, Data Min. Knowl. Discov. 1 (4) (1997) 343–373.
[32] S. Cong, J. Han, J. Hoeflinger, D. Padua, A sampling-based framework for parallel data mining, in: Proc. of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2005, pp. 255–265.
[33] T. Shintani, M. Kitsuregawa, Hash based parallel algorithms for mining association rules, in: Parallel and Distributed Information Systems, 1996, Fourth Int. Conf. on, IEEE, 1996, pp. 19–30.
[34] A. Mueller, Fast sequential and parallel algorithms for association rule mining: a comparison, Tech. Rep., University of Maryland at College Park, College Park, MD, USA, 1998.

[35] M. Gabroveanu, M. Cosulschi, F. Slabu, Mining fuzzy association rules using MapReduce technique, in: Int. Symposium on INnovations in Intelligent SysTems and Applications, INISTA, 2016, pp. 1–8.
[36] M. Gabroveanu, I. Iancu, M. Cosulschi, N. Constantinescu, Towards using grid services for mining fuzzy association rules, in: Proc. of the 1st East European Workshop on Rule-Based Applications, RuleApps, 2007, pp. 507–513.
[37] M. Gabroveanu, M. Cosulschi, N. Constantinescu, A New Approach to Mining Fuzzy Association Rules from Distributed Databases, Annals of the University of Bucharest LIV, 2005, pp. 3–16.
[38] R. Agrawal, T. Imielinski, A. Swami, Mining associations between sets of items in large databases, in: ACM-SIGMOD Int. Conf. on Data, 1993, pp. 207–216.
[39] P. Hájek, The question of a general concept of the GUHA method, Kybernetika 4 (1968) 505–515.
[40] P. Hájek, T. Havranek, Mechanizing Hypothesis Formation, Springer Verlag, Berlin, 1978.
[41] J. Calero, G. Delgado, M. Sánchez-Marañón, D. Sánchez, M.A.V. Miranda, J. Serrano, An experience in management of imprecise soil databases by means of fuzzy association rules and fuzzy approximate dependencies, in: ICEIS 2004, Porto, Portugal, April 14-17, 2004, 2004, pp. 138–146.
[42] F. Berzal, M. Delgado, D. Sánchez, M. Vila, Measuring accuracy and interest of association rules: a new framework, Intell. Data Anal. 6 (3) (2002) 221–235.
[43] N. Marín, M. Ruiz, D. Sánchez, Fuzzy frameworks for mining data associations: fuzzy association rules and beyond, Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 6 (2) (2016) 50–69.
[44] M. Delgado, M. Ruiz, D. Sánchez, J. Serrano, A formal model for mining fuzzy rules using the RL representation theory, Inf. Sci. 181 (23) (2011) 5194–5213.
[45] M.D. Ruiz, D. Sánchez, M. Delgado, M.J. Martin-Bautista, Discovering fuzzy exception and anomalous rules, IEEE Trans. Fuzzy Syst. 24 (4) (2016) 930–944.
[46] M. Delgado, M.D. Ruiz, D. Sánchez, Studying interest measures for association rules through a logical model, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 18 (1) (2010) 87–106, https://doi.org/10.1142/S0218488510006404.
[47] D. Dubois, E. Hüllermeier, H. Prade, A systematic approach to the assessment of fuzzy association rules, Data Min. Knowl. Discov. 13 (2) (2006) 167–192.
[48] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proc. of the 9th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, 2012.
[49] E. Louie, T. Young, Finding association rules using fast bit computation: machine-oriented modeling, in: International Symposium on Methodologies for Intelligent Systems, Springer, 2000, pp. 486–494.
[50] J. Rauch, M. Šimůnek, An alternative approach to mining association rules, in: Foundations of Data Mining and Knowledge Discovery, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 211–231.
[51] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo, et al., Fast discovery of association rules, Adv. Knowl. Discov. Data Min. 12 (1) (1996) 307–328.
[52] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nat. Commun. 5 (2014) 4308.
[53] H. Liu, F. Hussain, C.L. Tan, M. Dash, Discretization: an enabling technique, Data Min. Knowl. Discov. 6 (4) (2002) 393–423.
[54] M. Calvo-Flores, M. Ruiz, D. Sánchez, J. Serrano, A fuzzy rule mining approach involving absent items, in: Proc. of the EUSFLAT'2011, 2011, pp. 275–282.
[55] C. Fernandez-Basso, M. Ruiz, M. Martin-Bautista, A fuzzy mining approach for energy efficiency in a Big Data framework, IEEE Trans. Fuzzy Syst. (2020), https://doi.org/10.1109/TFUZZ.2020.2992180.
[56] V.P. Kumar, A. Gupta, Analyzing scalability of parallel algorithms and architectures, J. Parallel Distrib. Comput. 22 (3) (1994) 379–391.
[57] A.Y. Grama, A. Gupta, V. Kumar, Isoefficiency: measuring the scalability of parallel algorithms and architectures, IEEE Parallel Distrib. Technol. 1 (3) (1993) 12–21.
[58] C. Barba-González, J. García-Nieto, A. Benítez-Hidalgo, A.J. Nebro, J.F. Aldana-Montes, Scalable inference of Gene Regulatory Networks with the Spark distributed computing platform, in: J. Del Ser, E. Osaba, M.N. Bilbao, J.J. Sanchez-Medina, M. Vecchio, X.-S. Yang (Eds.), Intelligent Distributed Computing XII, Springer International Publishing, Cham, 2018, pp. 61–70.
[59] F.J. Baldán, J.M. Benítez, Distributed FastShapelet Transform: a Big Data time series classification algorithm, Inf. Sci. (2018).
[60] C. Barba-Gonzaléz, J. García-Nieto, A.J. Nebro, J.F. Aldana-Montes, Multi-objective Big Data optimization with jMetal and Spark, in: H. Trautmann, G. Rudolph, K. Klamroth, O. Schütze, M. Wiecek, Y. Jin, C. Grimme (Eds.), Evolutionary Multi-Criterion Optimization, Springer International Publishing, Cham, 2017, pp. 16–30.
[61] E. Hüllermeier, Association rules for expressing gradual dependencies, in: Proc. PKDD 2002, in: Lecture Notes in Computer Science, vol. 2431, 2002, pp. 200–211.
[62] F. Berzal, J. Cubero, D. Sánchez, M. Vila, An alternative approach to discover gradual dependencies, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 15 (5) (2007) 559–570.
[63] M. Delgado, M. Ruiz, D. Sánchez, New approaches for discovering exception and anomalous rules, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 19 (2) (2011) 361–399.