Cluster Computing
https://doi.org/10.1007/s10586-023-04014-w

New Spark solutions for distributed frequent itemset and association rule mining algorithms

Carlos Fernandez-Basso[1,2] · M. Dolores Ruiz[1] · Maria J. Martin-Bautista[1]

Received: 9 June 2022 / Revised: 18 July 2022 / Accepted: 18 April 2023

Abstract
The large amount of data generated every day makes necessary the re-implementation of new methods capable of handle with massive data efficiently. This is the case of Association Rules, an unsupervised data mining tool capable of extracting information in the form of IF-THEN patterns. Although several methods have been proposed for the extraction of frequent itemsets (previous phase before mining association rules) in very large databases, the high computational cost and lack of memory remains a major problem to be solved when processing large data. Therefore, the aim of this paper is three fold: (1) to review existent algorithms for frequent itemset and association rule mining, (2)to develop new efficient frequent itemset Big Data algorithms using distributive computation, as well as a new association rule mining algorithm in Spark, and (3) to compare the proposed algorithms with the existent proposals varying the number of transactions and the number of items. To this purpose, we have used the Spark platform which has been demonstrated to outperform existing distributive algorithmic implementations.

Keywords Big Data · Data Mining · Association Rule · Frequent Itemset · Distributed computing · Spark

1 Introduction

The enormous amounts of data stored by companies, social networks and internet of things like sensors in buildings and/or cities, are producing new challenges when applying Data Mining techniques [1]. Businesses and society have increased their interest on knowledge and information discovery from these huge quantities of data by means of different procedures. The problem arise when the majority of the classic knowledge extraction techniques are not capable of working with these amounts of data, triggering to the undesired memory overflows. For this reason new techniques have appeared in order to store and process data, facilitating the programmer to abstract the complexity of data management in large clusters. The Big Data framework gives a new perspective to store and process large amounts of data, enabling the management and processing of a large variety of data like streaming data (e.g. audio, image, video, etc.).

Data mining covers a wide range of techniques for knowledge discovery which can be classified into two main types: supervised techniques such as classification methods [2] and non-supervised such as clustering [3] or association rule mining. In this paper, we focus on Association Rule Mining (ARM) which discovers patterns in the data in the form of IF-THEN rules. ARM is usually based on two phases: (1) the extraction of frequent itemset by means of algorithms such as Apriori [4], Eclat [5] or FP-Growth [6] and (2) the extraction of association rules using previously extracted frequent itemsets using some measure of interest (e.g. Confidence, Lift or Certainty Factor) [7]. Association rules are specially interesting to discover relationships between items, that can be market or bank transactions [8], student records [9], or sensor meters [10], but often this kind of datasets are too large to analyse them with classic techniques. Additionally, frequent itemsets mining is an

M. Dolores Ruiz and Maria J. Martin-Bautista have contributed equally to this work.

✉ Carlos Fernandez-Basso
cjferba@decsai.ugr.es; carlos.basso@ucl.ac.uk

M. Dolores Ruiz
mdruiz@decsai.ugr.es

Maria J. Martin-Bautista
mbautis@decsai.ugr.es

[1] Dept of Computer Science and A.I., University of Granada, Granada, Spain

[2] Causal Cognition lab, University College London, London, United Kingdom

Published online: 30 April 2023

important phase in ARM that can also be employed for discovering other types of patterns such as sequential patterns [11], gradual dependencies [12] or exception and anomalous rules [13] to discover meaningful information in the data.

For being capable of processing such massive data we are going to focus on distributive computing based on MapReduce framework since it is transparent to the programmer and the distribution process is automatically managed by the system. For that, two main platforms have arisen: Hadoop and Spark. The former is a great solution to manage data and store them in large clusters, but its parallel processing is done on disk and its implementation of MapReduce only allows a phase Map and Reduce without being able to use iterative processing. On the contrary, Spark performs its processing in main memory increasing thus the computing speed significantly (up to 100 times faster) [14, 15] and allows global shared variables among clusters, important feature for our proposals.

Moreover, other of the limitations of Hadoop is that it has to query the dataset after executing each job, thus incurring in a large disk input/output, while Spark directly passes the data without writing to a persistent storage [16, 17]. In addition, the implementation of Spark [3] enables faster memory operations than Hadoop [18] since it allows in-memory computations (see the complete comparison made in [19] where Hadoop and Spark frameworks are compared in several Machine Learning algorithms).

There are few available implementations using the Spark framework for mining frequent itemsets and association rules. Although some research has been published in this direction, mainly in proceedings papers (examples of this are [20–23]) some of them are parallel versions but are not thought to work in distributed systems. Other available proposals only include frequent itemset mining and are designed using the pure MapReduce framework and implemented in Hadoop. These works cannot be directly compared to Spark proposals due to the limitations previously mentioned of MapReduce in Hadoop. Frequent itemset mining (FIM) proposals have been also proposed in Spark [23–25], although their implementations are not available. The only algorithm available in Spark is that of PFP presented in [20] where a distributed version of FP-Growth is presented to extract top Q frequent itemsets to support query recommendations. Therefore, it does not discover all frequent itemsets exceeding a support threshold. This is a clear disadvantage when discovering association rules, since itemsets with not such a high support may co-occur with high confidence with other items and therefore we might miss the information captured with the omitted rules.

In the light of previous observations, the aim of this paper is threefold. Firstly, we review existent parallel and distributive algorithms for frequent itemset and association rule mining. Secondly, three new exhaustive algorithms have been designed inspired on the sequential versions of the Apriori, Apriori-TID and ECLAT algorithms. These novel and original designs allow data to be processed using a Spark distributive framework that improves efficiency and performance for processing massive data. In addition, as far as we know, we have proposed the first Association Rule mining algorithm in Spark. Lastly, we compare these three new proposals with a Spark version of the YAFIM [25] (implemented by us), and the PFP [20] algorithm available in the Spark library, by studying speed up, efficiency and memory consumption, in terms of the amount of transactions and items in different sets of data, as well as the number of obtained association rules (since the PFP is not exhaustive). For that, we have employed very large datasets that fail when processing them with non-Big Data versions of the algorithms. The different algorithms are also compared in terms of the obtained results when they are applied to discover frequent itemsets and frequent and confident association rules.

The paper is structured as follows: Next section summarizes some needed concepts and reviews the existent works for association rule and frequent itemset mining. It also contains a brief explanation of the different approaches and highlights their main differences. Section 3 presents three new Big Data algorithms for frequent itemset mining. Section 4 develops a new proposal for association rule mining in Spark. In Section 5 we compare the new implemented algorithms with YAFIM and the PFP algorithm available in the Spark library. Finally in last Section the reader can find the main conclusions of the comparative analysis and some lines for future research.

2 Previous research and related work

In this section we first summarize the notions of support and confidence and afterwards we review the existent literature about association rule discovery algorithms [26] and frequent itemset mining, since FIM is in many cases the first phase of association rule mining algorithms. We focus this review on distributive proposals, classifying them according to the framework employed (Hadoop of Spark).

2.1 Definitions

Association rules were formally first defined by Agrawal et al. [4] as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ a set of items and $D = \{t_1, t_2, \ldots, t_N\}$ be a set of N transactions containing a subset of items. In this ambient an association rule can be defined as follows:

$$X \rightarrow Y, \text{ where } X, Y \subseteq I \text{ and } X \cap Y = \emptyset. \quad (1)$$

X is referred as the antecedent (or left-hand side of the rule) and Y as the consequent (or right-hand side of the rule). The problem of uncovering association rules is usually developed in two steps:

- Step 1: Finding all the itemsets above the minimum support threshold. These itemsets are known as frequent itemsets.
- Step 2: Using the frequent itemsets, association rules are discovered by imposing a minimum threshold for an assessment measure such as confidence.

The most commonly used measures to extract frequent itemsets and association rules are:

- The *support* [27] measures the frequency of appearance of an itemset in the database. In general, the most interesting association rules are those with a high support value.

$$Supp_D(X) = \frac{t_i \in D : X \subseteq t_i}{|D|} \tag{2}$$

- Given the itemsets X and Y, and the database D, the confidence of the rule $X \rightarrow Y$ [27], represented as $Conf_D(X \rightarrow Y)$, is the conditional probability of Y appearing in those transactions of D that also contain X.

$$Conf_D(X \rightarrow Y) = \frac{Supp_D(X \cup Y)}{Supp_D(X)} \tag{3}$$

In the literature one can find diverse kinds of algorithms which use different data structures to represent the transactions. Depending on these structures it is necessary to use different approaches to distribute the algorithms and extract association rules. For this reason we first analyse traditional sequential algorithms and we move a step forward by describing parallel and distributed approaches using Big Data models.

2.2 Related works

In this section we review the existent algorithms for association rule and frequent itemset mining focusing on those that are distributive.

2.2.1 Sequential and parallel approaches

Apriori was first proposed by Agrawal and Srikant in the mid-nineties [4] for finding the frequent set of itemsets and afterwords mining association rules using the downward closure property. Since then, other proposals have been developed such as Apriori-TID, ECLAT or FP-Growth. The main idea of the Apriori-TID algorithm is to reduce the itemsets to be analysed by sorting the transactions by item frequency, and removing non-frequent ones in each step [4]. The ECLAT algorithm [5] uses the structure of TD-list [28] to improve computations using Boolean operators. FP-growth employs an FP-tree structure for using the divide-and-conquer technique and consulting the database of transactions only once [6]. This results in a very fast algorithm. These algorithms have been analysed and compared in several works [29–32] concluding that although the Apriori algorithm is the most widely used and known, the FP-Growth outperforms the others with respect to time consumption, but, Apriori and Apriori-TID need less resources in terms of memory.

A different type of proposals are those considering a parallelization of the frequent itemset extraction process. In this regard, we can highlight the following works: parallel versions of Apriori, with some variations, can be found in [33, 34], ParEclat (Parallel Eclat) is described in [35], and Parallel FP-Growth with Sampling is presented in [36].

2.2.2 Distributive approaches

Distributive algorithms are capturing more attention due to the new philosophy introduced around Big Data using the MapReduce framework. In this regard, two different environments arise: Hadoop [37], which follows pure MapReduce philosophy, and Spark [38], which also enables in-memory computations.

2.2.2.1 Hadoop approaches Among the proposals using Hadoop, we can highlight the Dist-Eclat and BigFIM algorithms presented in [21] for the extraction of frequent itemsets. These proposals employed a load balancing scheme for the Dist-Eclat algorithm, and for the BigFIM proposal, a hybrid approach following an Apriori variant which distributes the mappers using the sequential ECLAT algorithm. In [39–41] the authors proposed different Apriori implementations for FIM using Hadoop called SPC, FPC and DPC. More Apriori-based proposals in Hadoop where proposed in [42] with and without pruning strategy,[1] called AprioriMR, iterative AprioriMR, pruning AprioriMR and top AprioriMR. Additionally, the authors also proposed a maximal AprioriMR algorithm for mining condensed representations of frequent itemsets. In [43] the FIMMR algorithm is proposed for FIM in Hadoop and was compared with PFP (parallel FP-growth available in Mahout) and SPC in two datasets with very good time speedup performance. Authors in [44] developed the BIGMiner algorithm for FIM and compared it with the following implementations of Hadoop: SPC, BigFIM, FIMMR and PFP in Mahout, obtaining that BIGMiner improved the other MapReduce versions accelerating the support counting and reducing the network communication overhead.

[1] The pruning strategy is the well known anti-monotone property employed in Apriori.

Regarding Hadoop implementations of association rule mining algorithms, as far as we know, there are two different proposals. The proposal in [45] is based on genetic programming and was compared with 14 sequential versions of ARM algorithms including Apriori and ECLAT, and other multi-objective proposals. And the work in [46] developed an algorithm to discover quantitative association rules, which is a special type of association rule where the attribute values lie in a numerical range.

Nevertheless, as was pointed in the introduction, Spark offers some advantages enabling faster memory operations than Hadoop since it allows in-memory computations, increasing thus the computing speed significantly (up to 100 times faster) [19].

2.2.2.2 Spark approaches We can distinguish among two types of approaches: exhaustive and non-exhaustive ones, based on their capacity to extract all the frequent itemsets or part of them. In addition we also describe the main differences between batch data processing algorithms and stream data ones.

- Exhaustive approaches: Among the proposals using the Spark framework we highlight the YAFIM algorithm presented in [25] and the R-Apriori developed in [23] which are Spark approaches of Apriori similar to our proposal. The main difference is the ordering of the MapReduce phase. They make the loop to search k-itemsets inside the distributed process using a hash tree, meanwhile we make the MapReduce for every k-itemset using a hash table. The problem of the implementations in [23, 25] is that it is very difficult to adapt it for the Apriori-TID, since in every step of the loop the YAFIM and R-Apriori algorithms do not know if a k-itemset is frequent or not, information that is used in the TID list. In addition, posterior analysis made in [47] 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).
- Non-exhaustive approaches: Another proposal using the Spark framework, but not exhaustive, is the PFP which is a distributed adaptation of FP-Growth algorithm for mining most frequent itemsets. It is based on a different structure which does not coincide with the traditional FP-Tree, because there does not exist efficient Spark implementations of distributed trees. The PFP algorithm sorts and divides data in several groups and counts itemsets in each group using MapReduce paradigm. This algorithm has different phases:
  - Parallel counting of the number of repetitions of each item using MapReduce.
  - Grouping Items: Dividing all the items into k groups. The algorithm obtains a list of groups, where each group is given by a unique groupID.
  - MapReduce phase: Per each transaction it extracts the groups containing the items in it. Afterwards, they are reduced by groupID.
  - Aggregation of results. It aggregates the results obtained in previous steps giving as a final result the top Q frequent itemsets.

  This implementation only returns the frequent itemsets of higher level exceeding the minimum support threshold (i.e. if ABC is a frequent itemset, A, B, C, AB, AC and BC are not retrieved). Note that the PFP also depends on a parameter k set at the beginning of the algorithm. This may be an inconvenient, for instance when mining association rules, since itemsets of different granularity are necessary during the extraction process. Following with non exhaustive algorithms, we can highlight the proposals developed in [48–50] whose efficiency is improved through the use of some pruning and reduction techniques in the search of candidates. This leads to a reduction on the number of obtained frequent itemsets in the results. As occurred with PFP, these algorithms are often used in recommendation systems [20, 49] where it is not require to be exhaustive in their search space, since they are more interested in retrieving only the most frequent itemsets (not all of them exceeding the *MinSupp* threshold).
- Batch vs stream data algorithms: It is also worth to distinguish two different types of algorithms. On the one hand we can differentiate among those proposals oriented to the extraction of frequent itemsets or association rules in batch data [20, 25, 44, 48–50]. And, on the other hand, those oriented to mine streaming data as [51] and [52]. In these cases the data is analysed in sliding windows along the time to be analysed. A summary of the proposals can be found in Table 1, where some of their main characteristics are shown.

From this regard, our proposal is based on three new Spark implementations inspired on Apriori, Apriori-TID and ECLAT to discover frequent itemsets, and another new algorithm to extract association rules. Our experimentation focuses in those available algorithms that are exhaustive with batch data processing, incorporating the PFP (Parallel FP-Growth) [20] approach to compare with a non-exhaustive algorithm, and the YAFIM algorithm whose implementations are available in the Spark library. These algorithms have been selected because the vast majority of proposals do not have the implementation available, so it is not possible to use them in the experimental comparison.

To this purpose, next section goes into a detailed description of developed algorithms DApriori, DATID and DECLAT (where the first "D" stands for distributive) for frequent itemset mining, and proposes a distributive algorithm for Association Rule Mining, part in common for every algorithm.

Table 1 Distributive frequent itemset (FIM) and association rule mining (ARM) algorithms

Cite	Name	Type	Year	Comments	Availability
[20]	PFP	FIM	2008	Non-exhaustive algorithm	✓
[25]	YAFIM	FIM	2014	Exhaustive algorithm	✓
[50]	DFIMA	FIM	2015	Non-exhaustive algorithm	×
[23]	R-Apriori	FIM	2015	Exhaustive algorithm	×
[48]	HFIM	FIM	2017	Non-exhaustive algorithm	×
[42]	AprioriMR	FIM	2018	Non-exhaustive algorithm	×
[44]	BIGMiner	FIM	2018	Non-exhaustive algorithm	×
[49]	Adaptive-Miner	ARM	2018	Non-exhaustive algorithm for a recommendation system	×
[51]	Distributed FIMoTS	FIM	2019	Streaming	×
[52]	SWEclat	FIM	2020	Streaming	×

3 New frequent itemset mining Spark algorithms

In this section, we present three new algorithms for frequent itemset mining using Big Data techniques. These proposals are inspired by the operation of the traditional sequential Apriori, Apriori-TID and ECLAT algorithms, which justify the given names for them: DApriori, DATID and DECLAT. These algorithms are all implemented using the Spark framework. Before delve into the details, next section recalls the necessary concepts for the good understanding of the algorithms.

3.1 Preliminary concepts

The most famous framework for Big Data is *MapReduce* designed by Google in 2003 [53]. It has become one of the most relevant tools for processing large datasets with parallel and distributed algorithms on a cluster. The MapReduce framework manages all data transfers and communications between the systems. It also provides redundancy, fault-tolerance and job scheduling. In this programming paradigm we usually have two phases. Firstly the *Map*() function, which makes the processing of data and returns the data transformed into key value pairs depending on our necessities. For instance, we may use a Map function applied to $<key, value>$ pairs that gives transformed pairs. In our case, the Map function is used to retrieve lists of itemsets:

$$Map(<item, value_i>) \rightarrow list(<itemset, value_j>) \quad (4)$$

Secondly, the *Reduce*() function aggregates the lists of $<key, value>$ pairs sharing the same key to obtain a piece of processed data. For example, to compute the frequency of appearance of an item we may use a Reduce function in the following way:

$$Reduce(<item, list(value)>) \rightarrow <item, value_{aggregated}> \quad (5)$$

Apache Hadoop [37] is an open source software platform for distributed storage and distributed processing of very large data. Hadoop allows distribution and processing of data comfortably for the programmer since it also provides services such as security, data access, data governance, operations, data storage and data processing. Later, Apache Spark appeared as an open-source framework built around speed, ease of use, and sophisticated analytics [38]. The most important feature of Spark is that it allows in-memory computing, and, as a consequence more complex algorithms can be developed. This is because Spark supports an advanced Directed Acyclic Graph (DAG) execution engine that enables cyclic data flow. In Apache Spark there exists an implementation of a data structure to abstract the concept of data partition. This structure is called Resilient distributed dataset (RDD) [54]. The RDD concept means that data collections are distributed across the clusters. The RDD has two different types of operations. First type of transformations converts the RDD structure in a different RDD, which are called *Transformation* operations. The second type are evaluation *Actions* performed over the above transformations which return a final value for each RDD partition. The programmer has to take into account, that evaluations are not executed until a specific Action operation is specified in the code. This is due to the "lazy" evaluation of Spark that strongly distinguishes between *Transformations* and *Actions*.

To design the distributed algorithms inspired on Apriori, Apriori-TID and ECLAT for frequent itemsets mining it would be necessary some primitive Spark functions. Here, we explain them:

- *Map*: Applies a transformation function to each element of RDD and returns a transformed RDD.
- *FlatMap*: Similar to Map, but each input item can be mapped to 0 or more output items.
- *Reduce*: Aggregates the elements of the dataset using an aggregation function.

Additionally the algorithm uses 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. The shared

variables are stored in a hash tree format, allowing direct and fast access to the query and insertion of the frequencies of each item or itemsets.

For a better understanding of the running algorithms we use the example database depicted in Table 2. In next sections, we explain the three approaches to extract frequent itemsets using Spark and after that we compare these algorithms with our implementation of YAFIM and the available Spark implementation of PFP.

function is employed (see line 5 of Algorithm 1 and first and second rows of Fig. 1). Right after, the Map function is applied returning pairs of the form $<item, 1>$ when item has appeared in a transaction (line 6 of Algorithm 1 and third row of Fig. 1). Finally the Reduce phase is in charge of merging items with the same key (line 7) filtering only those items exceeding the MinSupp threshold. Figure 1 describes this whole stage using the example database of Table 2.

Algorithm 1 Main Spark procedure for DApriori algorithm

1: **Input:** *Data:* RDD transactions: $\{t_1, \ldots, t_n\}$
2: **Input:** *MinSupp:* minimum support threshold
3: **Output:** $Global\_FreqItemset$: frequent itemsets exceeding $MinSupp$

Phase 1: FreqItems()

4: Distributive computing in q chunks of transactions: $\{S_1, \ldots, S_q\}$
5: $\{< it_1 >, \ldots, < it_m >\} \leftarrow S_i.\mathbf{FlatMap}()$
6: $\{< it_1, 1 >, \ldots, < it_m, 1 >\} \leftarrow \mathbf{Map}(< it_k >)$
7: $\{< it_1, card_1 >, \ldots, < it_m, card_m >\} \leftarrow \mathbf{Reduce}(|it_k|)$
8: $ItemSupport \leftarrow \mathbf{Support}(< it_k >)$
9: $DicFreqItemset \leftarrow \mathbf{Filter}(ItemSupport \geq MinSupp)$
10: End distributive computation

Phase 2: Candidate generation

11: $Candidate \leftarrow DicFreqItemset$ #Candidates of Length =1
12: $Length = 2$
13: $Global\_FreqItemset \leftarrow Candidate$
14: **broadcast**($Global\_FreqItemset$) #Creation of a broadcast variable available across all the cores in the cluster
15: **while** $|DicFreqItemset| > 1$ **do**
16: Distributive computing in q chunks of transactions: $\{S_1, \ldots, S_q\}$
17: $\{< its_1 >, \ldots, < its_m >\} \leftarrow S_i.\mathbf{FlatMap}()$
18: $\{< its_1, 1 >, \ldots, < its_m, 1 >\} \leftarrow \mathbf{Map}(< its_k >)$
19: $\{< its_1, card_1 >, \ldots, < its_m, card_m >\} \leftarrow \mathbf{Reduce}(|its_k|)$
20: $ItemsetSupport \leftarrow \mathbf{Support}(< its_k >)$
21: $DicFreqItemset = \mathbf{Filter}(ItemsetSupport \geq MinSupp)$
22: End distributive computation
23: $Length++$
24: $Candidate \leftarrow CandidateGen(DicFreqItemset, Length)$
25: $Global\_FreqItemset.\mathbf{Append}(Candidate)$
26: **end while**
27: **return** $Global\_FreqItemset$

3.2 DApriori: Apriori big data approach

DApriori has two main steps. The foremost involves loading the dataset and calculating how often each item appears in the set of transactions using Map and Reduce functions. The second step involves the calculation of frequent itemsets having atomic frequent items as input.

In the first stage of the algorithm we must have the appearing items in each transaction separately in order to be processed by the Map function. To that aim a FlatMap

During the second phase, the list of frequent items resulting from the previous stage are utilised to form the candidate itemsets of superior length using the downward closure property of frequent itemsets. For the new candidates the same process followed in the first stage is applied, i.e. a FlatMap followed by a MapReduce process plus a filtering step by their support. In Fig. 2 we can see the whole process for the second stage of the algorithm. This phase is executed repeatedly until no more candidates of higher length can be formed (see lines 15-26 of Algorithm 1).

Table 2 Dataset example

ID	Items
1	A,B,C
2	B,D
3	A,C,D
4	B,C
5	A,C
6	B,D
7	A,B,C
8	B,C,D
9	A,B,D
10	B,C,D

In particular, Fig. 3 depicts the second phase for the example database of Table 2 to obtain the frequent itemsets of length 2. In this second stage, a global broadcast variable is employed, as a global variable in sequential programming, to store the candidate items in order to have them available in every partition of the cluster.

3.3 DATID: Apriori-TID approach

DATID algorithm follows the same filosophy of Apriori-TID Algorithm. In Algorithm 2 can be seen the details, with two main differences with respect to DApriori. The first change occurs at the end of first phase. In this case, the algorithm sorts by support the frequent items obtained in the first stage and then removes the non-frequent items from the set (see Fig. 4 and line 10 of Algorithm 2). For this reason, the second stage differs from DApriori, since the computation of frequent itemsets is not performed over the original data. Instead, this computation is made using the transformed data, i.e data which only contain frequent items. Thanks to this modification, the second phase that retrieves itemsets of length 2, and subsequent $k+1$-itemsets, is faster and decreases memory consumption.

Algorithm 2 Main Spark procedure for DATID algorithm

1: **Input:** *Data:* RDD transactions: $\{t_1, \ldots, t_n\}$
2: **Input:** *MinSupp:* minimum support threshold
3: **Output:** $Global\_FreqItemset$: frequent itemsets exceeding $MinSupp$

Phase 1: FreqItems()

4: Distributive computing in q chunks of transactions: $\{S_1, \ldots, S_q\}$
5: $\{< it_1 >, \ldots, < it_m >\} \leftarrow S_i.\mathbf{FlatMap}()$
6: $\{< it_1, 1 >, \ldots, < it_m, 1 >\} \leftarrow \mathbf{Map}(< it_k >)$
7: $\{< it_1, card_1 >, \ldots, < it_m, card_m >\} \leftarrow \mathbf{Reduce}(|it_k|)$
8: $ItemSupport \leftarrow \mathbf{Support}(< it_k >)$
9: $DicFreqItemset \leftarrow \mathbf{Filter}(ItemSupport \geq MinSupp)$
10: $ArrayData \leftarrow S_i.\mathbf{Map}(\mathbf{Remove}(DicFreqItemset)).\mathbf{Map}(\mathbf{Sort}(DicFreqItemset)).collect()$
11: End distributive computation

Phase 2: Candidate generation

12: $Candidate \leftarrow DicFreqItemset$ #Candidates of Length =1
13: $Length = 2$
14: $Global\_FreqItemset \leftarrow Candidate$
15: **broadcast**($Global\_FreqItemset$) #Creation of a broadcast variable available across all the cores in the cluster
16: **while** $|DicFreqItemset| > 1$ **do**
17: Distributive computing in q chunks of transactions: $\{S_1, \ldots, S_q\}$
18: $\{< its_1 >, \ldots, < its_m >\} \leftarrow S_i.\mathbf{FlatMap}()$
19: $\{< its_1, 1 >, \ldots, < its_m, 1 >\} \leftarrow \mathbf{Map}(< its_k >)$
20: $\{< its_1, card_1 >, \ldots, < its_m, card_m >\} \leftarrow \mathbf{Reduce}(|its_k|)$
21: $ItemsetSupport \leftarrow \mathbf{Support}(< its_k >)$
22: $DicFreqItemset = \mathbf{Filter}(ItemsetSupport \geq MinSupp)$
23: $ArrayData \leftarrow S_i.\mathbf{Map}(\mathbf{Remove}(DicFreqItemset))$
24: End distributive computation
25: $Length++$
26: $Candidate \leftarrow CandidateGen(DicFreqItemset, Length)$
27: $Global\_FreqItemset.\mathbf{Append}(Candidate)$
28: **end while**
29: **return** $Global\_FreqItemset$

Figure 5 illustrates the second stage of DATID algorithm for the example dataset of Table 2. Second row in Fig. 5 contains ordered frequent items for their posterior processing. Let us emphasize, that for implementation of DATID, two broadcast variables are employed. The first variable called *Global_FreqItemset* contains the ordered list of frequent items and the second variable called *Candidate* which stores the candidate list of itemsets for every iteration in the second phase.

3.4 DECLAT: ECLAT approach

We have also developed DECLAT algorithm following the philosophy of the sequential ECLAT algorithm, but using the MapReduce paradigm in the Spark framework (see Algorithm 3). In this case, the principal difference resides in the data distribution by itemsets, instead of distributing the process by set of transactions like in DApriori or DATID algorithms Fig. 6.

For this reason, DECLAT has a preprocessing phase, because the database has to be changed to consider items as transactions. This process is developed in lines 4-7 of Algorithm 3 where we can see the two main distributive functions for preprocessing the data. First step uses FlatMap, returning for each transaction a pair $<TransactionID, Item>$ if the item appears in the transaction. After that, the function *GroupByKey* aggregates the pairs and returns pairs $<Item, [ListTransaction]>$ where the list transaction contains 1 or 0 depending whether the item is in the transaction or not respectively. Figure 7 depicts this process.

DECLAT is also comprised of two phases like previous algorithms. In the first phase, the algorithm counts the number of appearances of every item in every transaction. After preprocessing, the *Reduce* function calculates frequent items and generates the itemsets candidates for the next step. For the computation of the itemsets support, the algorithm employs the *FlatMap*() function and the itemsets lists like broadcast variables to generate the pairs $<Itemset, support>$.

Afterwards, in the second phase, the *FlatMap*() function returns a pair comprised of an itemset and a list with 1 or 0 if the itemset appear in the transaction or not, and the algorithm follows with the computation of the support

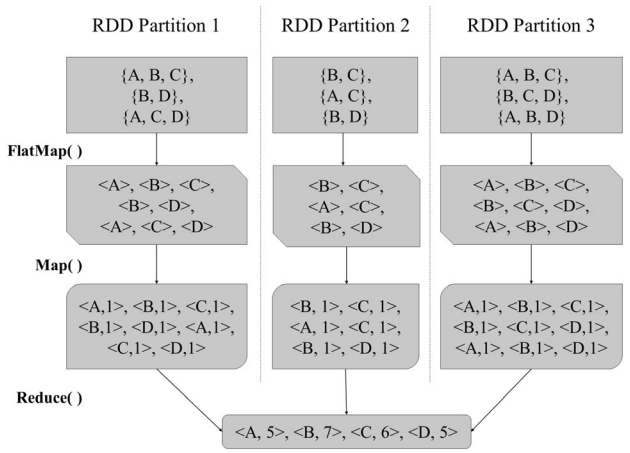


Fig. 1 Example of Phase 1 using the example dataset for DApriori and DATID algorithms

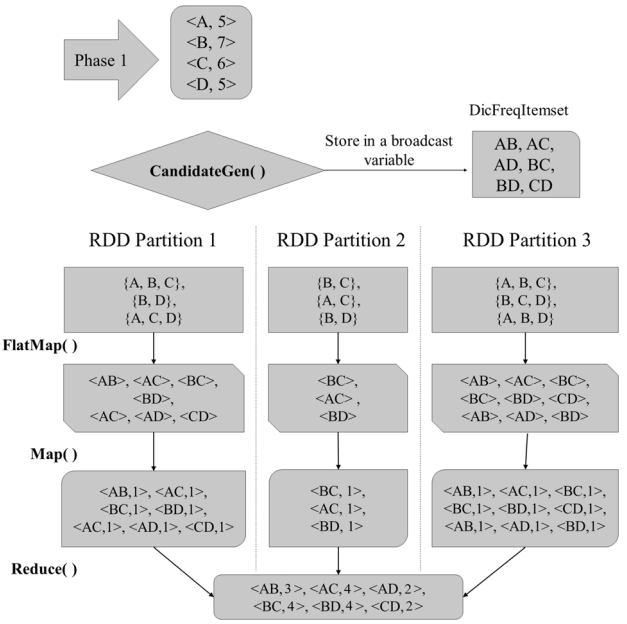
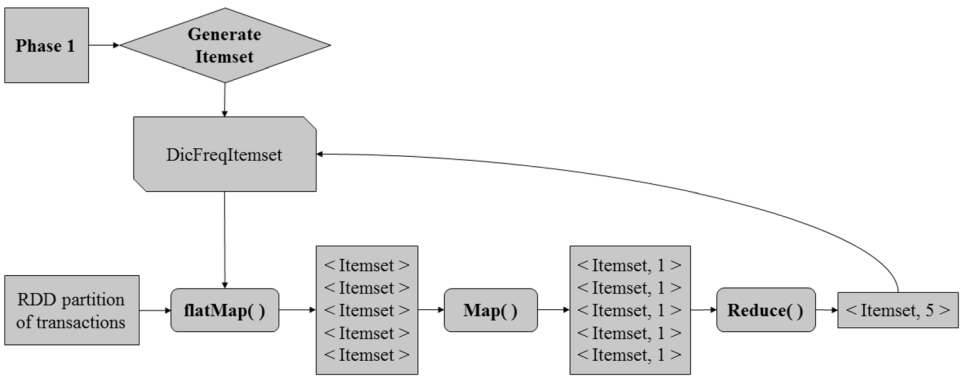


Fig. 3 Example of Phase 2 using the example dataset for DApriori algorithm

Fig. 2 Phase 2 of DApriori algorithm using Spark



using the *Reduce* function (see the Fig. 8 and lines 20-21 of Algorithm 3).

list of frequent itemsets and it also calculates the confidence of each candidate rule (see lines 4-5 of Algorithm 4).

Algorithm 3 Main Spark procedure for DECLAT algorithm

1: **Input:** *Data:* RDD transactions: $\{t_1, \ldots, t_n\}$
2: **Input:** *MinSupp:* minimum support threshold
3: **Output:** $Global\_FreqItemset$: frequent itemsets exceeding $MinSupp$

Preprocessing

4: Distributive computing in q chunks of transactions: $\{S_1, \ldots, S_q\}$
5: $\{< t_1, it_1 >, \ldots, < t_n, it_m >\} \leftarrow S_i.\textbf{FlatMap}()$
6: $ArrayData = \{< it_1, [ListTr_1] >, \ldots, < it_m, [ListTr_m] >\} \leftarrow S_i.\textbf{GroupByKey}(it_k)$
7: End distributive computation

Phase 1: FreqItems()

8: Distributive computing in q chunks of $ArrayData$: $\{AD_1, \ldots, AD_q\}$
9: $\{< it_1, [ListTr_1] >, \ldots, < it_m, [ListTr_m] >\} \leftarrow AD_i.\textbf{FlatMap}()$
10: $\{< it_1, card_1 >, \ldots, < it_m, card_m >\} \leftarrow \textbf{Reduce}(\sum ListTr_k)$
$\#\sum ListTr_k$ computes the number of 1s in $ListTr_k$ (see Figure 6)
11: $ItemSupport \leftarrow \textbf{Support}(< it_k >)$
12: $DicFreqItemset \leftarrow \textbf{Filter}(ItemSupport \geq MinSupp)$
13: End distributive computation

Phase 2: Candidate generation

14: $Candidate \leftarrow DicFreqItemset$ #Candidates of Length =1
15: $Length = 2$
16: $Global\_FreqItemset \leftarrow Candidate$
17: **broadcast**($Global\_FreqItemset$) #Creation of a broadcast variable available across all the cores in the cluster
18: **while** $|DicFreqItemset| > 1$ **do**
19: Distributive computing in q chunks of $ArrayData$: $\{AD_1, \ldots, AD_q\}$
20: $\{< its_1, [ListTr_1] >, \ldots, < its_m, [ListTr_m] >\} \leftarrow AD_i.\textbf{FlatMap}()$
21: $\{< its_1, card_1 >, \ldots, < its_m, card_m >\} \leftarrow \textbf{Reduce}(\sum ListTr_k)$
22: $ItemsetSupport \leftarrow \textbf{Support}(< its_k >)$
23: $DicFreqItemset = \textbf{Filter}(ItemsetSupport \geq MinSupp)$
24: End distributive computation
25: $Length++$
26: $Candidate \leftarrow CandidateGen(DicFreqItemset, Length)$
27: $Global\_FreqItemset.\textbf{Append}(Candidate)$
28: **end while**
29: **return** $Global\_FreqItemset$

4 A new Spark association rule mining algorithm

Once frequent itemsets are extracted, the final step is to uncover the association rules that assess the predetermined thresholds for support and confidence. This process has been also designed to follow the MapReduce framework.

The result of previous FIM algorithms results in a list of frequent itemsets that will be in RDD format used by Spark. After that, for each partition the Map function generates the possible rules which are determined by the

This will return, for each partition, the list of rules using the following format $<key, value>$ where

- key: represents a rule where the antecedent is separated from the consequent by *string*_, where the underscore is used to separate both parts of the rule.
- value: contains the confidence of the rule

To finish, the *Reduce* function is applied to generate the final association rules (see line 6 of Algorithm 4) Fig. 9.

Algorithm 4 Spark procedure for association rule mining

1: **Input:** *Candidate:* Candidate list of frequent itemsets
2: **Input:** *MinConf:* minimum confidence threshold
3: **Output:** *Rules* : Association Rules exceeding $MinConf$
4: Distributive computing in r chunks of Candidate: $\{C_1, \ldots, C_r\}$
5: $\{< Rule_1, Conf_1 >, \ldots, < Rule_s, Conf_s >\} \leftarrow C_j$.**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: $Rules \leftarrow$ **Reduce**$(Conf_i \geq MinConf)$
7: End distributive computation
8: **return** Rules

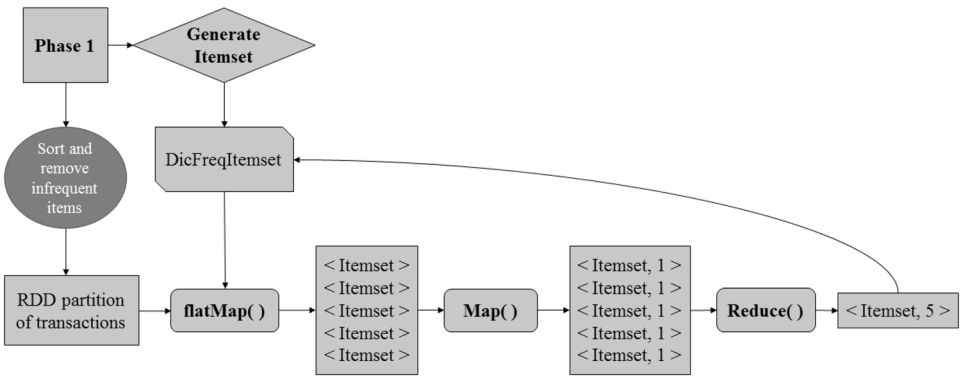


Fig. 4 Phase 2 of DATID algorithm using Spark

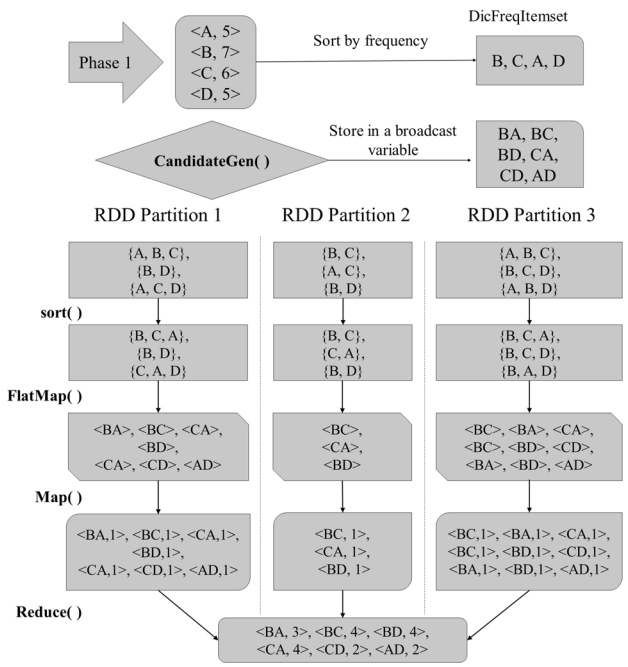


Fig. 5 Process for extracting frequent itemsets with DATID Algorithm

5 Experiments and results

We now move on to the different experiments we carried out with the PFP (available in the Spark library), our implementation of the YAFIM algorithm [25], DApriori, DATID and DECLAT algorithms in order to compare their performance. Our aim was to study the behaviour of the different available algorithms implemented using Big Data philosophy, in particular using the Spark technology. Traditional association rule extraction algorithms do not enable the use of this type of large data sets because they have no memory capacity or their execution time is extremely high.

To this end, all the performed experiments have been executed on a cluster with 32 cores on 2 Intel Xeon E5 and with 200 GB of RAM; and four different datasets have been employed. These datasets and the parameters used can be found in Table 3. The first three datasets are available at the UCI Machine Learning repository and the last one (`Otto` database) can be found in the open repository KAGGLE:

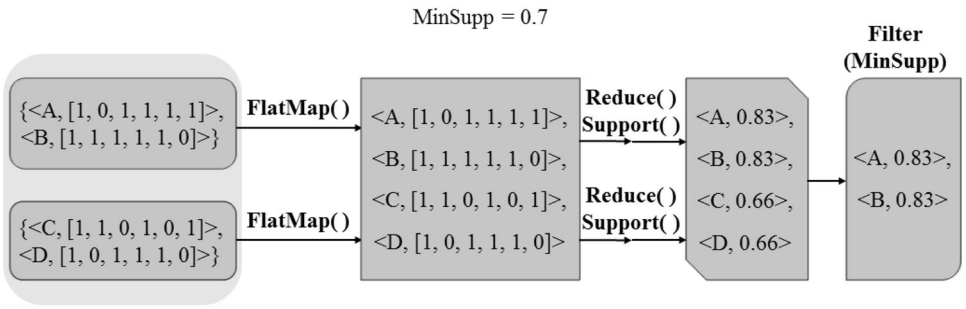


Fig. 6 Example of first Phase for DECLAT algorithm

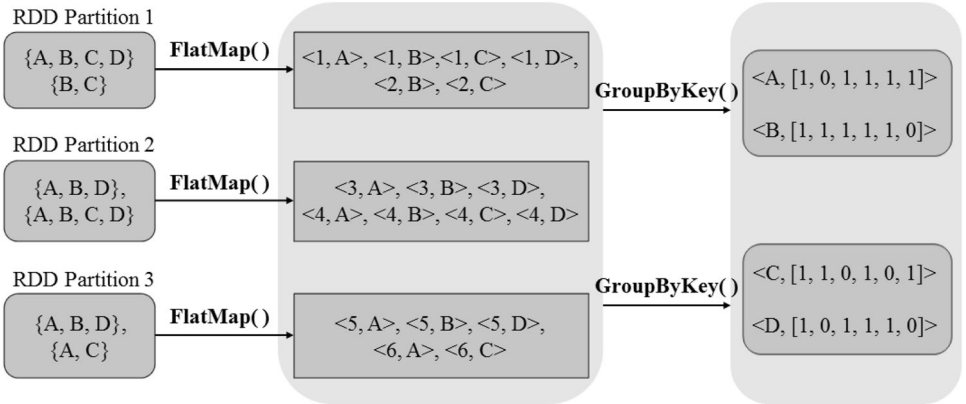


Fig. 7 Example of transformation of Data for DECLAT algorithm

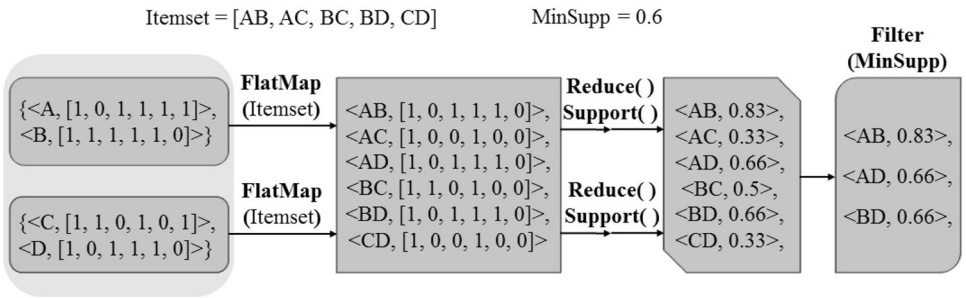


Fig. 8 Example of second Phase for DECLAT algorithm

- `Poker` dataset has 1 025 000 instances and 11 attributes. In this dataset each transaction is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank) making a total of 78 items. [55]
- `Susy` dataset [56], which consists of 5 000 000 instances and 18 attributes, contains simulated collisions events at the Large Hadron Collider.
- `Higgs` dataset [57], which has 11 000 000 instances and 28 attributes, consists of another trial of simulated collisions at the Large Hadron Collider.
- `Otto` dataset, which consists in 90 000 instances and 93 attributes, contains features about different products of the Otto group online shop.

The described datasets will be used to study the behaviour of the three algorithms with regard to different perspectives:

- The first group of experiments has been designed to study the consumption time with regard to the number of transactions and different configurations for the number of items in the datasets. For the latter, we have restricted the performed the experiments for the `Otto` dataset with different scales of magnitude for the number of items, because the number of items is lower in the rest of datasets.
- The second group of experiments aimed to analyse the scalability with regard to the memory usage for every algorithm.
- The third group of experiments is aimed to study the efficiency and speed up of the different algorithms. For that, several executions have been run for the different datasets by varying the number of processors (or cores) in order to study the percentage of improvement and the speed up achieved by increasing the number or cores.

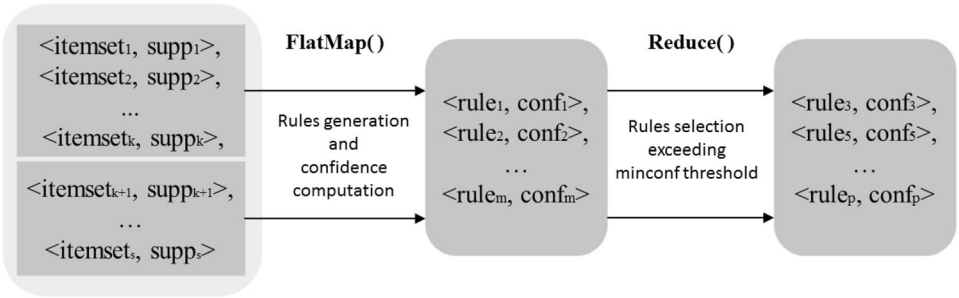


Fig. 9 Main procedure for Spark association rule mining

Table 3 Dataset and parameters description

Name	Transactions	Features	Items	MinSupport	MinConfidence
`Poker`	1 025 010	11	78	0.05	0.7
`Susy`	5 000 000	18	54	0.2	0.7
`Higgs`	11 000 000	28	92	0.2	0.7
`Otto`	90 000	93	4 300	0.2	0.7

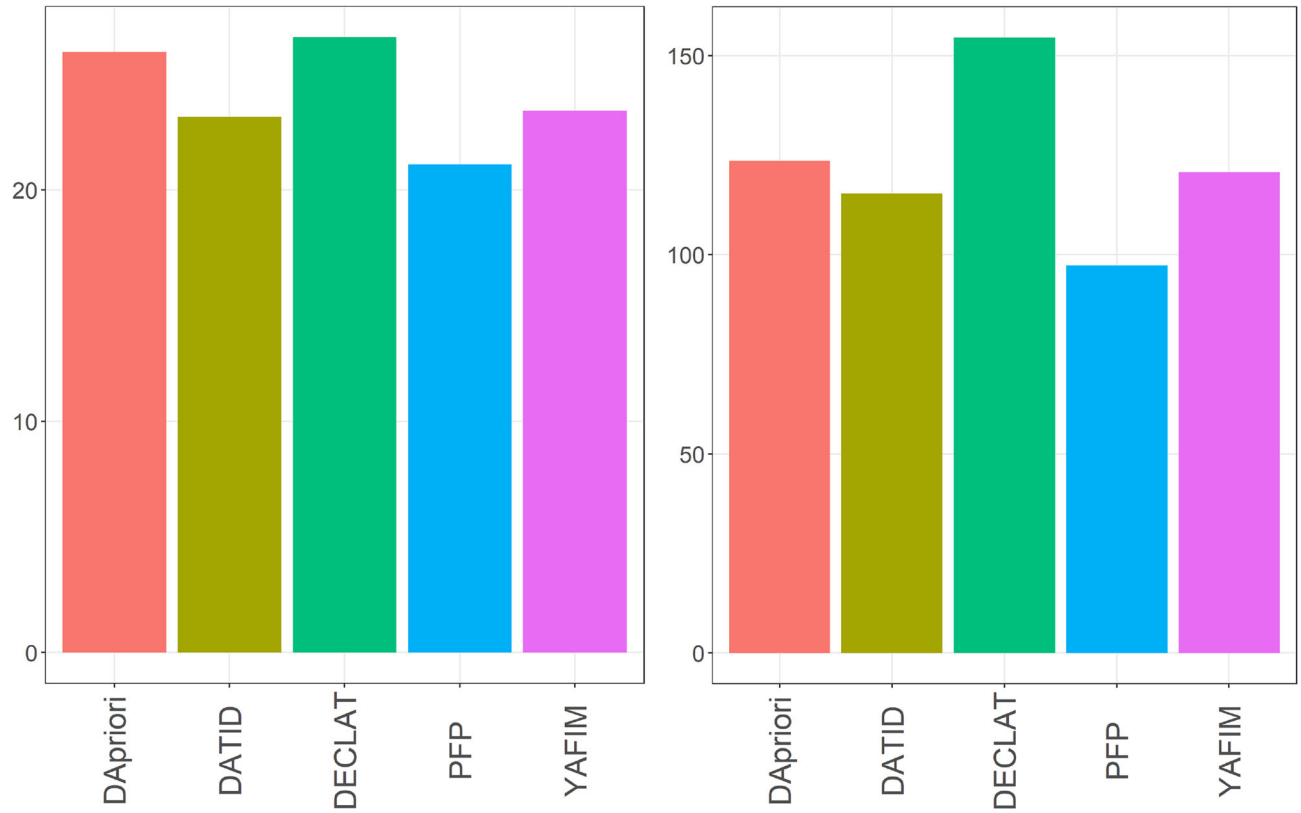


Fig. 10 Runtime in minutes for `Poker` (left) and `Higgs` (right) datasets

Figures 10 and 11 contain the runtime in minutes for each algorithm in the four datasets. The difference between the DApriori, DATID, DECLAT and YAFIM algorithms is not very significant regarding the runtime, but the DATID significantly improves the memory usage in some cases (see Fig. 13). Moreover the memory usage of DECLAT is worse than the others algorithms because the use of lists is more inefficient. The PFP algorithm outperforms the DApriori, DATID and DECLAT with respect to the runtime in every dataset, but DATID improves memory consumption in all cases except for the `Otto` database which remains equal in the rest of the cases.

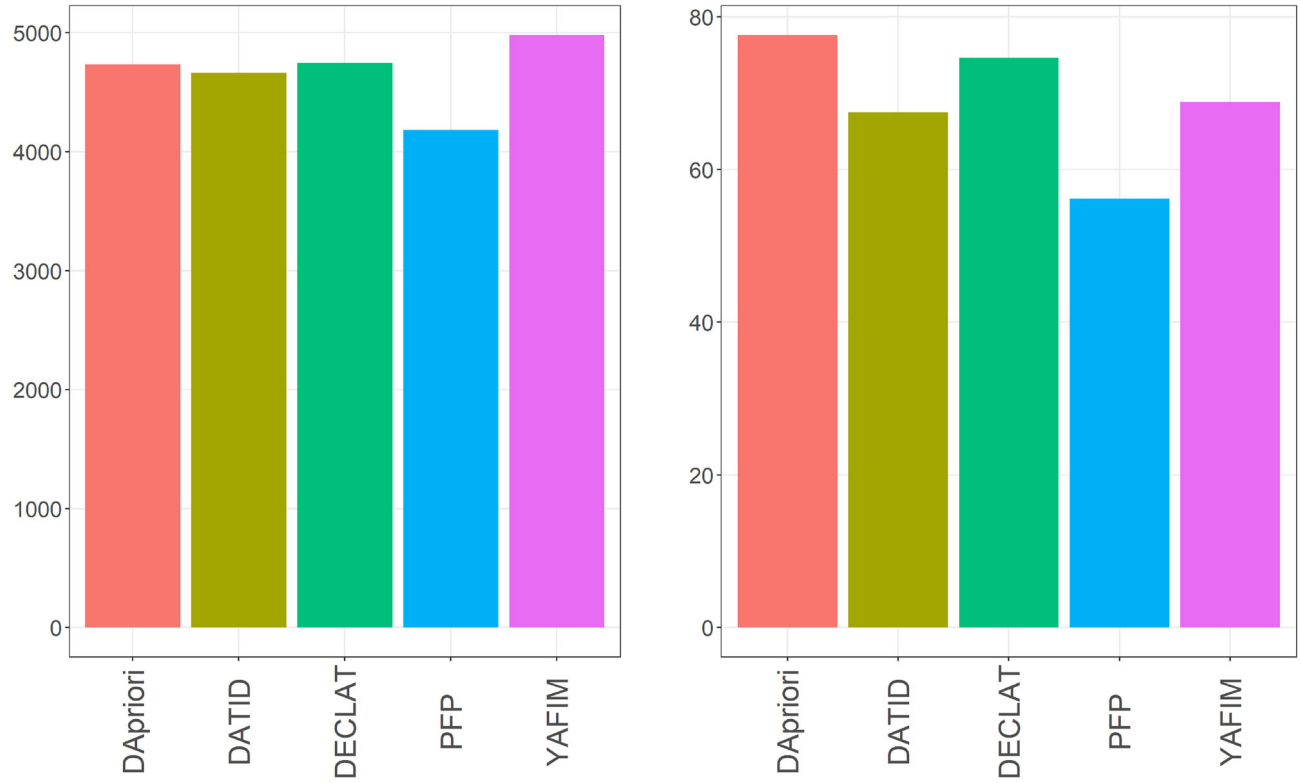


Fig. 11 Runtime in minutes for `Otto` (left) and `Susy` (right) datasets

It is important to note that PFP is not exhaustive, in the sense that not all frequent itemsets are retrieved since it only gives top Q frequent itemsets with the highest length (i.e. the highest itemset containing each frequent item). This is because it was originally designed for query recommendation where only top Q frequent itemsets are needed. This issue is a drawback when using this implementation of PFP algorithm for association rule mining, because frequent itemsets of every length are needed to extract the association rules, unless we are interested in only the association rules containing items with high support. We have adapted this part (i.e. association rule mining) using the Q parameter to group the sets of items according to the frequency of the items of length 1. Then, in the following phases, these groups are used to build the different combinations to generate the sets of k-itemsets.

Therefore, we must take care, because more confident rules could be missed due to their low support. This phenomenon is clearly depicted in Fig. 17 where PFP does not obtain all the association rules for the same support and confidence thresholds.

Figures 10 and 11 show that the execution time for `Susy` (5 M transactions) and `Higgs` (11 M transactions) is lower compared to `Otto` (0.09 M transactions) dataset. This is due to the high dimensionality of `Otto` database in the number of items. This phenomenon can also be observed in Fig. 12, where the runtime exponentially increases with respect to the number of items. It can be seen therefore that by growing the number of items the algorithm worsens their operation more than by increasing the number of transactions. This is because the algorithm needs to perform many searches when combining the different items to create the itemsets in each iteration.

Figure 13 shows the memory usage of each algorithm for every dataset. It can be seen that DATID outperforms the others in almost all cases (it remains the same for the `Otto` database). This is due to the treatment of TidLists to accelerate the queries to the database.

With the purpose of measuring the efficiency of our proposal and compare it to the existent approaches, we have analysed the *speed up* and the *efficiency* [58–60] according to the number of cores. For that, we have computed the well-known measure of speed up defined as [60, 61]

$$S_n = T_1/T_n \tag{6}$$

where T_1 is the time of the sequential algorithm and T_n is the execution time of the distributed algorithm using several cores. The efficiency [58–60] is defined in a similar way as

$$E_n = S_n/n = T_1/(n \cdot T_n). \tag{7}$$

In Figs. 14 and 15 it can be seen the results obtained for `Higgs` database (similar results are obtained for the

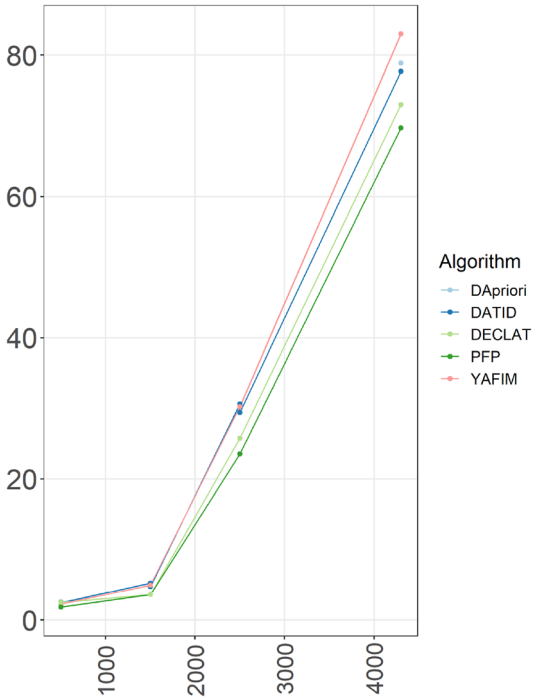


Fig. 12 Runtime in hours (y-axis) for `Otto` dataset with different sets of items (x-axis)

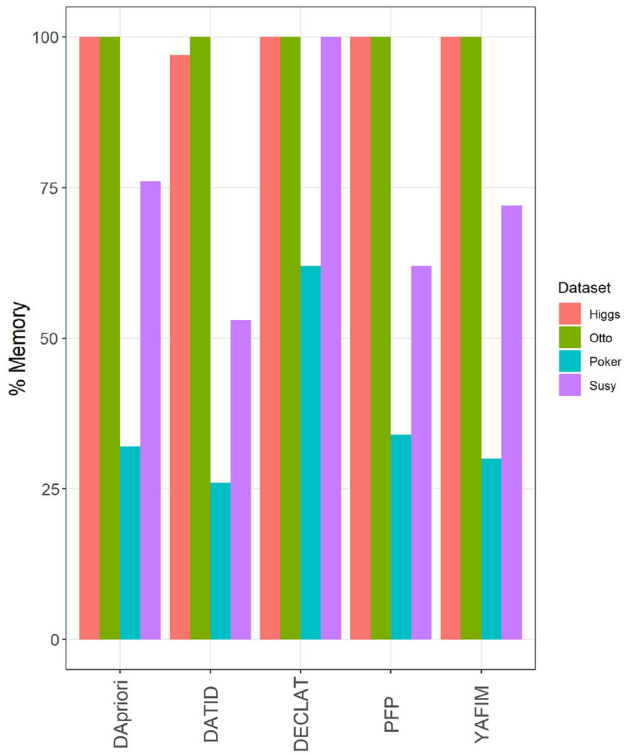


Fig. 13 Memory consumption for each dataset in every algorithm

reminder datasets). In them, we can observe that as the number of cores increases, the efficiency and speed up are improved, although they are not optimal. This is due to the

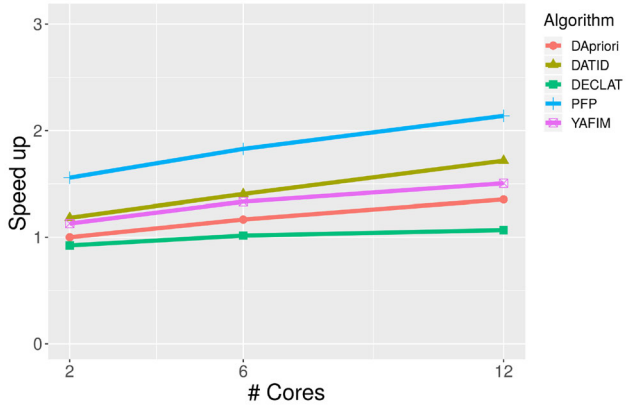


Fig. 14 Speed up of different algorithms for `Higgs` dataset

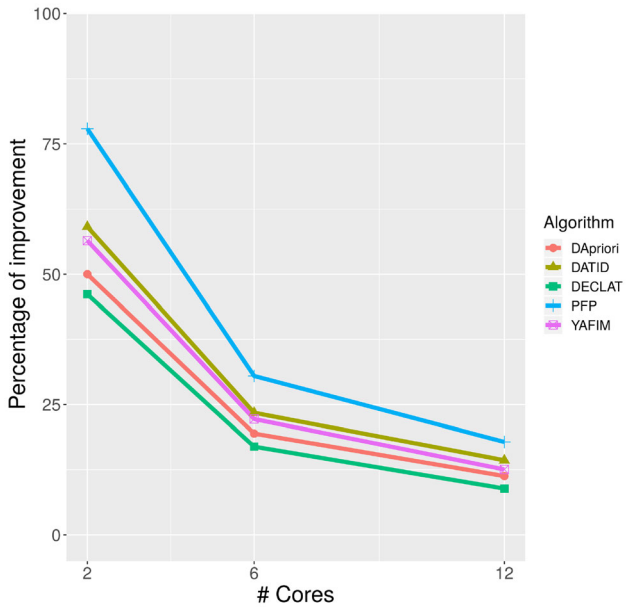


Fig. 15 Efficiency of different algorithms for `Higgs` dataset measured by percentage of improvement

cores workloads and the network congestion employed for the communication among the cores. Moreover, in Fig. 14 it is observed that the speed up increased along the number of processors employed, although it is not proportional as resources expand (see also how the efficiency does not increases in Fig. 15). This same behavior can be observed in other studies of speed up and efficiency in distributed algorithms, where the efficiency is not improved, as desired, with more processing cores [60, 62].

Finally, the generation of rules by each algorithm can be seen in Figs. 16 and 17. The DApriori, DATID, ECLAT and YAFIM algorithms generate all possible rules, while the PFP only discovers some of them. This is because the generation of association rules with the PFP is only made with the sets of longer frequent itemsets and do not consider all frequent itemsets of any length.

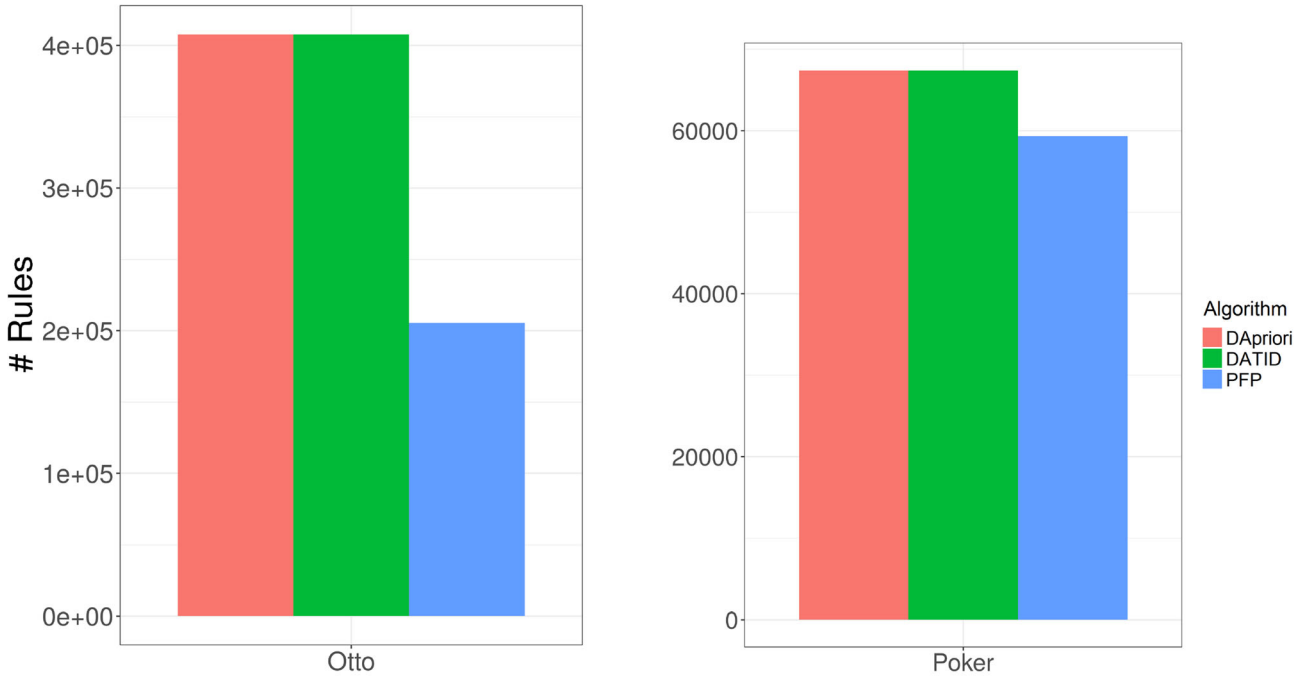


Fig. 16 Number of association rules extracted for `Otto` and `Poker` datasets

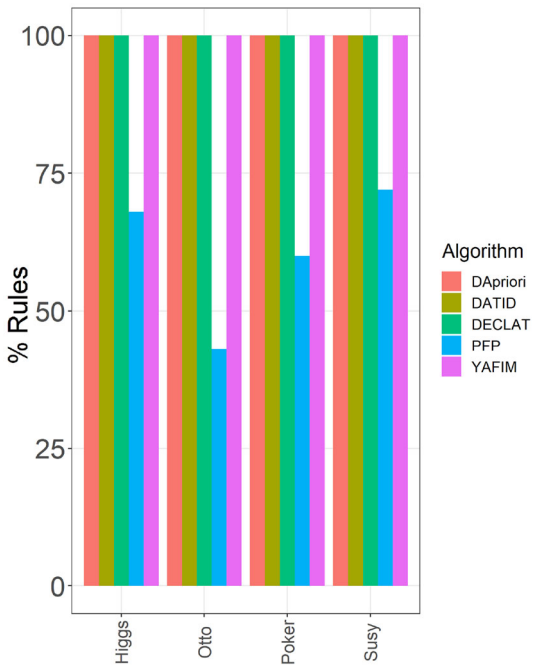


Fig. 17 Percentage of association rules obtained for the different datasets by each algorithm

This also explains the better speed up and efficiency of PFP algorithm, since it is not exhaustive. We remember that, the PFP algorithm was conceived to be used for recommendation systems, where only top Q frequent itemsets are needed.

5.1 Lessons learned

After all the performed experiments it can be observed that the most efficient solution is the Spark PFP. Nevertheless, this algorithm is not exhaustive and does not provide complete data results. So depending on the users' needs, a different algorithm should be chosen. For instance, if it is necessary to search the whole set to explore all possible association rules we need to use DATID (Distributive Apriori-TID) algorithm. On the other hand, if our search is less deep or we only need to obtain a subset of association rules, e.g. in a recommendation system, the PFP will be faster, although some interesting associations could be missing.

6 Conclusions

This paper gives an overview of the main literature about most representative algorithms for association rule mining, focusing in those designed to process very large data. Due to the drawbacks that Hadoop has compared to Spark, we have decided to developed new Big Data algorithms using Spark framework by inspiring on Apriori, Apriori-TID and ECLAT algorithms. These algorithms (DApriori, DATID and DECLAT) have been proved to be capable of processing massive data in large computer clusters. Additionally, these new algorithms have been compared with the YAFIM algorithm and the PFP algorithm available in the Spark library, obtaining that PFP outperforms

DApriori, DATID, DECLAT and YAFIM algorithms as far as the time consumption perspective is concerned, and DATID improves memory consumption. However, the PFP results are not always convenient for their posterior processing to extract association rules, since PFP is not exhaustive and only provides the longest frequent itemsets and their support. Therefore when the user is interested in obtaining the entire set of association rules exceeding the imposed thresholds in massive data, the best option available so far is to use DATID algorithm.

Regarding future research, our intention is to implement more efficient association rule mining algorithms by conveniently changing the PFP to extract all frequent itemsets. We also plan to face the problem of Association Rule Mining for streaming data, i.e. data continuously generated in real time. Additionally, we intend to apply the developed algorithms to extract patterns in sensor meters from buildings to improve their efficiency behaviour [10], a Big Data scenario where traditional association rule mining algorithms fail due to memory overflow problems.

Author contributions CF-B: Conceptualization, Investigation, Software, Writing, Original draft preparation MDR: Supervision, Writing—Reviewing and Editing MJM-B: Supervision, Writing—Reviewing and Editing

Funding Funding for open access publishing: Universidad de Granada/CBUA. The research reported in this paper was partially supported by the BIGDATAMED project, which has received funding from the Andalusian Government (Junta de Andalucía) under grant agreement No P18-RT-1765, by Grants PID2021-123960OB-I00 and Grant TED2021-129402B-C21 funded by Ministerio de Ciencia e Innovación and, by ERDF A way of making Europe and by the European Union NextGenerationEU. In addition, this work has been partially supported by the Ministry of Universities through the EU-funded Margarita Salas programme NextGenerationEU. Funding for open access charge: Universidad de Granada/CBUA.

Data availibility The used data is available on https://archive.ics.uci.edu/ml/index.php and https://www.kaggle.com

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

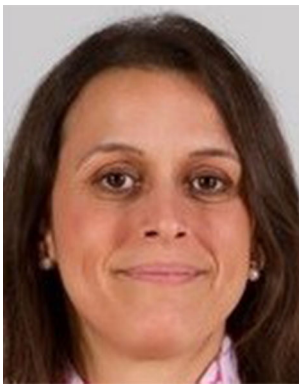
1. Wu, X., Zhu, X., Wu, G.-Q., Ding, W.: Data mining with big data. Knowl. Data Eng. IEEE Trans. **26**(1), 97–107 (2014)
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
4. Agrawal, R., Srikant, R., *et al.*: Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
5. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Know. Data Eng. **12**(3), 372–390 (2000)
6. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM Sigmod Record **29**(2), 1–12 (2000). (**ACM**)
7. Delgado, M., Ruiz, M.D., Sánchez, D.: Studying interest measures for association rules through a logical model. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. **18**(1), 87 (2010). https://doi.org/10.1142/S0218488510006404
8. Delgado, M., Martin-Bautista, M.J., Ruiz, M.D., Sánchez, D.: Detecting anomalous and exceptional behaviour on credit data by means of association rules. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8132 LNAI, pp. 143–154 (2013). https://doi.org/10.1007/978-3-642-40769-7_13
9. Bello-Orgaz, G., Jung, J.J., Camacho, D.: Social big data: Recent achievements and new challenges. Information Fusion **28**, 45–59 (2016). https://doi.org/10.1016/j.inffus.2015.08.005
10. Fernandez-Basso, C., Ruiz, M.D., Martin-Bautista, M.J.: 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
11. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: Mining sequential patterns by pattern-growth: the prefixspan approach. Knowl. Data Eng. IEEE Trans. **16**(11), 1424–1440 (2004)
12. Hüllermeier, E.: Association rules for expressing gradual dependencies. In: Proc. PKDD 2002 Lecture Notes in Computer Science, 2431, pp. 200–211 (2002)
13. Delgado, M., Ruiz, M.D., Sánchez, D.: New approaches for discovering exception and anomalous rules. Int. J. Uncertain. Fuzziness Knowled.-Based Syst. **19**(2), 361–399 (2011)
14. Samadi, Y., Zbakh, M., Tadonki, C.: Comparative study between hadoop and spark based on hibench benchmarks. In: 2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech), pp. 267–275 (2016). IEEE
15. Mavridis, I., Karatza, H.: Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark. J. Syst. Softw. **125**, 133–151 (2017)
16. Lin, X., Wang, P., Wu, B.: Log analysis in cloud computing environment with hadoop and spark. In: 2013 5th IEEE International Conference on Broadband Network & Multimedia Technology, pp. 273–276 (2013). IEEE
17. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. HotCloud **10**(10), 95 (2010)
18. White, T.: Hadoop: The Definitive Guide. Fourth Edition. O'Reilly, (2015)

19. Liu, L.: Performance comparison by running benchmarks on hadoop, spark and hamr. PhD thesis, University of Delaware (2016). http://udspace.udel.edu/bitstream/handle/19716/17628/2015_LiuLu_MS.pdf?sequence=1
20. Li, H., Wang, Y., Zhang, D., Zhang, M., Chang, E.Y.: PFP: parallel fp-growth for query recommendation. In: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 107–114 (2008). ACM
21. Moens, S., Aksehirli, E., Goethals, B.: Frequent itemset mining for big data. In: Big Data, 2013 IEEE International Conference On, pp. 111–118 (2013). IEEE
22. Chaudhary, H., Yadav, D.K., Bhatnagar, R., Chandrasekhar, U.: Mapreduce based frequent itemset mining algorithm on stream data. In: Lobal Conference on Comunication Technologies 2015 (GCCT 2015), pp. 598–603 (2015)
23. Rathee, S., Kaul, M., Kashyap, A.: R-apriori: An efficient apriori based algorithm on spark. In: Proceedings of the PIKM'15, pp. 27–34. ACM, Melbourne, VIC, Australia (2015)
24. Zaki, M.J.: Parallel and distributed association mining: a survey. IEEE Concurr. **4**, 14–25 (1999)
25. Qiu, H., Gu, R., Yuan, C., Huang, Y.: Yafim: A parallel frequent itemset mining algorithm with spark. In: Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, pp. 1664–1671 (2014). IEEE
26. Aggarwal, C.C., Han, J.: Frequent pattern mining. Springer, Berlin (2014)
27. Berzal, F., Blanco, I., Sánchez, D., Vila, M.A.: A new framework to assess association rules. Advances in intelligent data analysis, pp. 95–104. Springer, Berlin (2001)
28. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W., *et al.*: New algorithms for fast discovery of association rules. In: KDD, vol. 97, pp. 283–286 (1997)
29. Zheng, Z., Kohavi, R., Mason, L.: Real world performance of association rule algorithms. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 401–406 (2001). ACM
30. Borgelt, C.: Efficient implementations of apriori and eclat. In: FIMI'03: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, p. 90 (2003)
31. Hunyadi, D.: Performance comparison of Apriori and FP-Growth algorithms in generating association rules. In: Proceedings of the European Computing Conference, pp. 376–381 (2011)
32. Garg, K., Kumar, D.: Comparing the performance of frequent pattern mining algorithms. Int. J. Comput. Appl. **69**(25), 21–28 (2013)
33. Agrawal, R., Shafer, J.C.: Parallel mining of association rules. IEEE Trans. Know. Data Eng. **8**(6), 962–969 (1996). https://doi.org/10.1109/69.553164
34. Shintani, T., Kitsuregawa, M.: Hash based parallel algorithms for mining association rules. In: Parallel and Distributed Information Systems, 1996., Fourth International Conference On, pp. 19–30 (1996). IEEE
35. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: Parallel algorithms for discovery of association rules. Data Mining Know. Discov. **1**(4), 343–373 (1997)
36. Cong, S., Han, J., Hoeflinger, J., Padua, D.: A sampling-based framework for parallel data mining. In: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 255–265 (2005). ACM
37. White, T.: Hadoop: the definitive guide. O'Reilly Media Inc., Sebastopol (2012)
38. Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning spark: lightning-fast big data analysis. O'Reilly Media Inc., Sebastopol (2015)
39. Li, N., Zeng, L., He, Q., Shi, Z.: Parallel implementation of apriori algorithm based on mapreduce. In: Proceedings of the 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. SNPD '12, pp. 236–241. IEEE Computer Society, Washington, DC, USA (2012)
40. Farzanyar, Z., Cercone, N.: Efficient mining of frequent itemsets in social network data based on mapreduce framework. In: Proceedings of the 2013 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), pp. 1183–1188 (2013)
41. Farzanyar, Z., Cercone, N.: Accelerating frequent itemset mining on the cloud: A mapreduce-based approach. In: IEEE 13th International Conference on Data Mining Workshops, pp. 592–598 (2013)
42. Luna, J.M., Padillo, F., Pechenizkiy, M., Ventura, S.: Apriori versions based on mapreduce for mining frequent patterns on big data. IEEE Trans. Cybern. **48**(10), 2851–2865 (2018). https://doi.org/10.1109/TCYB.2017.2751081
43. Wang, L., Feng, L., Zhang, J., Liao, P.: An Efficient Algorithm of Frequent Itemsets Mining Based on MapReduce. Journal of Information Computational Science 11(8), 2809–2816 (2014). https://doi.org/10.12733/jics20103619
44. Chon, K.W., Kim, M.S.: BIGMiner: a fast and scalable distributed frequent pattern miner for big data. Cluster Computing **21**(3), 1507–1520 (2018). https://doi.org/10.1007/s10586-018-1812-0
45. Padillo, F., Luna, J.M., Herrera, F., Ventura, S.: Mining association rules on Big Data through MapReduce genetic programming. Integrated Computer-Aided Engineering **25**(1), 31–48 (2017). https://doi.org/10.3233/ICA-170555
46. Martín, D., Martínez-Ballesteros, M., García-Gil, D., Alcalá-Fdez, J., Herrera, F., Riquelme-Santos, J.C.: MRQAR: A generic MapReduce framework to discover quantitative association rules in big data problems. Knowledge-Based Systems **153**, 176–192 (2018). https://doi.org/10.1016/j.knosys.2018.04.037
47. Singh, S., Garg, R., Mishra, P.K.: Performance analysis of apriori algorithm with different data structures on hadoop cluster. International Journal of Computer Applications **128**(9), 45–51 (2015)
48. Sethi, K.K., Ramesh, D.: Hfim: a spark-based hybrid frequent itemset mining algorithm for big data processing. The Journal of Supercomputing **73**(8), 3652–3668 (2017)
49. Rathee, S., Kashyap, A.: Adaptive-miner: an efficient distributed association rule mining algorithm on spark. Journal of Big Data **5**(1), 6 (2018)
50. 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 **18**(4), 1493–1501 (2015)
51. Fernandez-Basso, C., Francisco-Agra, A.J., Martin-Bautista, M.J., Ruiz, M.D.: Finding tendencies in streaming data using big data frequent itemset mining. Knowledge-Based Systems **163**, 666–674 (2019)
52. Xiao, W., Hu, J.: Sweclat: a frequent itemset mining algorithm over streaming data using spark streaming. The Journal of Supercomputing, 1–16 (2020)
53. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)
54. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 2 (2012). USENIX Association
55. Lichman, M.: UCI Machine Learning Repository (2013). http://archive.ics.uci.edu/ml

56. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature Communications **5**(4308) (2014)
57. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature Communications **5** (2014)
58. Kumar, V.P., Gupta, A.: Analyzing scalability of parallel algorithms and architectures. Journal of parallel and distributed computing **22**(3), 379–391 (1994)
59. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Parallel & Distributed Technology: Systems & Applications **1**(3), 12–21 (1993)
60. Barba-González, C., García-Nieto, J., Benítez-Hidalgo, A., Nebro, A.J., Aldana-Montes, J.F.: Scalable inference of gene regulatory networks with the spark distributed computing platform. In: Del Ser, J., Osaba, E., Bilbao, M.N., Sanchez-Medina, J.J., Vecchio, M., Yang, X.-S. (eds.) Intelligent Distributed Computing XII, pp. 61–70. Springer, Cham (2018)
61. Baldán, F.J., Benítez, J.M.: Distributed fastshapelet transform: a big data time series classification algorithm. Information Sciences **496**, 451–463 (2018)
62. Barba-Gonzaléz, C., García-Nieto, J., Nebro, A.J., Aldana-Montes, J.F.: Multi-objective big data optimization with jmetal and spark. In: Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M., Jin, Y., Grimme, C. (eds.) Evolutionary Multi-Criterion Optimization, pp. 16–30. Springer, Cham (2017)

Carlos Fernandez-Basso Carlos Fernandez-Basso received the degree in computer science, the M.Sc. degree in data science, and the Ph.D. degree in computer science from the University of Granada, Granada, Spain, in 2014, 2015, and 2020, respectively. He is currently a Postdoctoral Fellow with Causal Cognition Lab, University College London, London, U.K. He was a Lead Developer in the EU FP7 Project Energy IN TIME in the topics of building simulation and control, data analytics, and machine learning, and in the COPKIT Project in the topics of cybercrime, Big Data, and machine learning. From 2016 to 2018, he collaborated with the Data Science Institute, Imperial College London, London, U.K., where he has carried out research stays.

M. Dolores Ruiz M. Dolores Ruiz received the degree in mathematics and the European Ph.D. degree in computer science from the Universidad de Granada, in 2005 and 2010, respectively. She is a Professor at the Department of Computer Science and Artificial Intelligence at the University of Granada, Spain, since 2020. She has participated in more than ten projects, including the EU FP7 Projects ePOOLICE and Energy IN TIME, and the COPKIT H2020 project. Her research interests include data mining, information retrieval, energy efficiency, big data, correlation statistical measures, sentence quantification, and fuzzy sets theory. She has organized several special sessions about Data Mining in international conferences and was part of the organization committee of the FQAS'2013, SUM'2017 and FQAS'2023 conferences. Dr. Ruiz belongs to the Approximate Reasoning and Artificial Intelligence Research Group and the Cybersecurity Lab, Universidad de Granada. She is the Principal Investigator of several projects about federated mining and desinformation detection.



Maria J. Martin-Bautista Maria J. Martin-Bautista is a Full Professor at the Department of Computer Science and Artificial Intelligence at the University of Granada, Spain, since 1997. She is a member of the IDBIS (Intelligent Data Bases and Information Systems) research group. Her current research interests include Data Science and Big Data Analytics in Data, Text, Web and Social Networks, Intelligent Information Systems, Knowledge Representation and Uncertainty. She has supervised several Ph. D. Thesis and published more than 100 papers in high impact international journals and conferences. She has participated in more than 20 R+D projects and has supervised several research technology transfers with companies. She has served as a program committee member for several international conferences.