Apriori versions based on MapReduce for Mining Frequent Patterns on Big Data

J.M. Luna, *Member, IEEE,* F. Padillo, M. Pechenizkiy, *Member, IEEE,* and S. Ventura, *Senior Member, IEEE,*

Abstract—Pattern mining is one of the most important tasks to extract meaningful and useful information from raw data. This task aims to extract item-sets that represent any type of homogeneity and regularity in data. Although many efficient algorithms have been developed in this regard, the growing interest in data has caused the performance of existing pattern mining techniques to be dropped. The goal of this paper is to propose new efficient pattern mining algorithms to work in Big Data. To this aim, a series of algorithms based on the MapReduce framework and the Hadoop open-source implementation have been proposed. The proposed algorithms can be divided into three main groups. First, two algorithms (AprioriMR and IAprioriMR) with no pruning strategy are proposed, which extract any existing item-set in data. Second, two algorithms (SPAprioriMR and TopAprioriMR) that prune the search space by means of the well-known anti-monotone property are proposed. Finally, a last algorithm (MaxAprioriMR) is also proposed for mining condensed representations of frequent patterns. To test the performance of the proposed algorithms, a varied collection of Big Data datasets have been considered, comprising up to $3 \cdot 10^{18}$ transactions and more than 5 million of distinct single-items. The experimental stage includes comparisons against highly efficient and well-known pattern mining algorithms. Results reveal the interest of applying MapReduce versions when complex problems are considered, and also the unsuitability of this paradigm when dealing with small data.

Index Terms—Pattern Mining, Big Data, MapReduce, Hadoop

1 Introduction

Data analysis has a growing interest in many fields like business intelligence, which includes a set of techniques to transform raw data into meaningful and useful information for business analysis purposes. With the increasing importance of data in every application, the amount of data to deal with has become unmanageable, and the performance of such techniques could be dropped. The term Big Data [1] is more and more used to refer to the challenges and advances derived from the processing of such high dimensional datasets in an efficient way.

Pattern mining [2] is considered as an essential part of data analysis and data mining. Its aim is to extract subsequences, substructures or item-sets that represent any type of homogeneity and regularity in data, denoting intrinsic and important properties [3], [4]. This problem was originally proposed in the context of market basket analysis in order to find frequent groups [5] of products that are bought together [6]. Since its formal definition, at early nineties, a high number of algorithms has been described in the literature [7], [8], [9]. Most of these algorithms are based on Apriori-like methods [10], producing a list of candidate item-sets or patterns formed by any combination of single items. However, when the number of these single items to be combined increases, the pattern mining problem turns into an arduous task and more efficient strategies are required [11], [12]. To understand the complexity, let us consider a dataset comprising n single items or singletons. From them, the number of item-sets that can be generated is equal to $2^n - 1$, so it becomes extremely complex with the increasing number of singletons. All of this led to the logical outcome that the whole space of solutions can not always be analysed.

In many application fields [13], however, it is not required to produce any existing item-set but only those considered as of interest, e.g. those which are included into a high number of transactions. In order to do so new methods have been proposed, some of them based on the anti-monotone property as a pruning strategy. It determines that any sub-pattern of a frequent pattern is also frequent, and any super-pattern of an infrequent pattern will be never frequent. This pruning strategy enables the search space to be reduced since once a pattern is marked as infrequent, then no new pattern needs to be generated from it. Despite this, the huge volumes of data in many application fields have caused a decrease in the performance of existing methods. Traditional pattern mining algorithms are not suitable for truly Big Data [14], presenting two main challenges to be solved: computational complexity and main memory requirements [15]. In this scenario sequential pattern mining algorithms on a single machine may not handle the whole procedure and an adaptation of them to emerging technologies [16] might be fundamental to complete process.

MapReduce [17] is an emerging paradigm that has become very popular for intensive computing. The programming model offers a simple and robust method for writing parallel algorithms. Some authors [18] have recently described the significance of the MapReduce framework for processing large datasets, leading other

- *Dr. Luna is with the Department of Computer Science, University of Jaen, 23071, Jaen, Spain.*
  E-mail: jmluna@uco.es
- *F. Padillo is with the Department of Computer Science and Numerical Analysis, University of Cordoba, Rabanales Campus, 14071 Cordoba, Spain.*
  E-mail: fpadillo@uco.es
- *Dr. Pechenizkiy is with the Department of Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands.*
  E-mail: m.pechenizkiy@tue.nl
- *Dr. Ventura is with the Department of Computer Science and Numerical Analysis, University of Cordoba, Rabanales Campus, 14071 Cordoba, Spain. Dr. Ventura also belongs to Department of Information Systems, King Abdulaziz University, Saudi Arabia Kingdom.*
  E-mail: sventura@uco.es

Manuscript received –, –; revised –, –

parallelization schemes such as Message Passing Interface (MPI). This emerging technology has been applied to many problems where computation is often highly demanding, and pattern mining is one of them. *Moens et al.* [14] proposed first methods based on MapReduce to mine item-sets on large datasets. These methods were properly implemented by means of Hadoop [19], which is considered as one of the most popular open-source software frameworks for distributed computing on very large data sets.

Considering previous studies and proposals [13], the aim of this work is to provide research community with new and more powerful pattern mining algorithms for Big Data. These new proposals rely on the MapReduce framework and the Hadoop open-source implementation, and they can be classified as:

- No pruning strategy. Two algorithms (AprioriMR and IAprioriMR) are properly designed to extract patterns in large datasets. These algorithms extract any existing item-set in data regardless their frequency.
- Pruning the search space by means of the anti-monotone property. Two additional algorithms (SPAprioriMR and TopAprioriMR) are proposed with the aim of discovering any frequent pattern available in data.
- Maximal frequent patterns. A last algorithm (MaxAprioriMR) is also proposed for mining condensed representations of frequent patterns, i.e. frequent patterns with no frequent supersets.

To test the performance of the proposed models, a series of experiments over a varied collection of Big Data sets has been carried out, comprising up to $3 \cdot 10^{18}$ transactions and more than 5 million of singletons (a search space close to $2^{5,267,646} - 1$). Additionally, the experimental stage includes comparisons against both well-known sequential pattern mining algorithms and MapReduce proposals. The ultimate goal of this analysis is to serve as a framework for future researches in the field. Results have demonstrated the interest of using the MapReduce framework when Big Data is considered. They have also proved that this framework is unsuitable for small data, so sequential algorithms are preferable.

The rest of the paper is organized as follows. Section 2 presents the most relevant definitions and related work; Section 3 describes the proposed algorithms; Section 4 presents the datasets used in the experiments and the results obtained; finally, some concluding remarks are outlined in Section 5.

2 PRELIMINARIES

In this section, some relevant definitions and related works are described.

2.1 Pattern Mining

The term *pattern* is defined as a set of items that represents any type of homogeneity and regularity in data, denoting intrinsic and important properties of data [13]. Formally, let $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be a set of items, a pattern P is defined as $\{P = \{i_j, ..., i_k\} \subseteq \mathcal{I}, j \geq 1, k \leq n\}$. Given a pattern P, its length or size is denoted as $|P|$, i.e. the number of singletons that it includes. Thus, for $P = \{i_1, i_2, ..., i_k\} \subseteq \mathcal{I}$, its size is defined as $|P| = k$. Additionally, given a set of all transactions $\mathcal{T} = \{t_1, t_2, ..., t_m\}$ in a dataset, the support of a pattern P is defined as the number of transactions that P satisfies, i.e. $support(P) = |\{\forall t_l \in \mathcal{T} : P \subseteq t_l\}|$. A pattern P is considered as frequent iff $support(P) \geq threshold$. It is noteworthy that the support of a pattern is monotonic, i.e. none of the super-patterns of an infrequent pattern can be frequent.

Many real-world application domains [20] contain patterns whose length is typically too high. The extraction and storage of such lengthy and frequent patterns requires a high computational time due to the significant amount of time used on computing sub-patterns. In this regard, the discovery of maximal frequent patterns as condensed representations of frequent patterns is a solution to overcome the computational and storage problems. A pattern P is considered as a maximal frequent pattern if it satisfies that none of its immediate super-patterns P^S are frequent, i.e. $\{\nexists P^S : P \subset P^S, support(P^S) \geq threshold\}$.

The use of brute-force approaches for mining frequent patterns requires $M = 2^n - 1$ item-sets to be generated and $O(M \times N \times n)$ comparisons when data comprise N transactions and n singletons. Hence, this computational complexity becomes prohibitively expensive for really large data sets. As a matter of example, let us consider a dataset including 20 singletons and 1 million of instances. Here, any brute-force approach has to mine $M = 2^{20} - 1 = 1,048,575$ item-sets and a total of $2.09 \cdot 10^{13}$ comparisons to compute the support of each pattern.

To overcome the prohibitively high computational time, some authors [21] proposed Eclat where the patterns are extracted by using a vertical data representation. Here, each singleton is stored in the dataset together with a list of transaction-ids where the item can be found. Even though this algorithm speeds up the process, many different exhaustive search approaches have been proposed in literature. For instance, *Han et al.* [7] proposed an algorithm based on prefix trees [22], where each path represents a subset of $\mathcal{I}$. The FP-Growth algorithm is one of the most well-known in this regard, which is able to reduce the database scans by considering a compressed representation of the database thanks to a data structure known as FP-Tree [23]. Following the monotonic property, if a path in the prefix tree is infrequent, any of its sub-trees are also infrequent. In such a situation, an infrequent item-set implies that its complete sub-tree is immediately pruned. Based on the FP-Tree structure, many different authors have proposed extensions of the FP-Growth method. For instance, *Ziani et al.* [20] proposed a modified algorithm for mining maximal frequent patterns.

2.2 MapReduce

MapReduce [17] is a recent paradigm of parallel computing. It allows to write parallel algorithms in a simple way, where the applications are composed of two main phases defined by the programmer: map and reduce. In the map phase, each mapper processes a sub-set of input data and produces key-value ($\langle k, v \rangle$) pairs. Then, an intermediate procedure is carried out, known as shuffle and sort, which sorts and merges all the values associated with the same key k. Finally, the reducer takes this new list of $\langle k, v \rangle$ pairs as input to produce the final $\langle k, v \rangle$ pairs. It is worth mentioning that any map/reduce operation is run in a parallel and distributed way. The flowchart of a generic MapReduce framework is depicted in Figure 1.

There are many MapReduce implementations [24], but Hadoop [19] is one of the most widespread due to its open-source implementation, installation facilities and the fundamental assumption that hardware failures are common and should be automatically handled by the platform. Furthermore, Hadoop proposes the use of a distributed filesystem, known as Hadoop

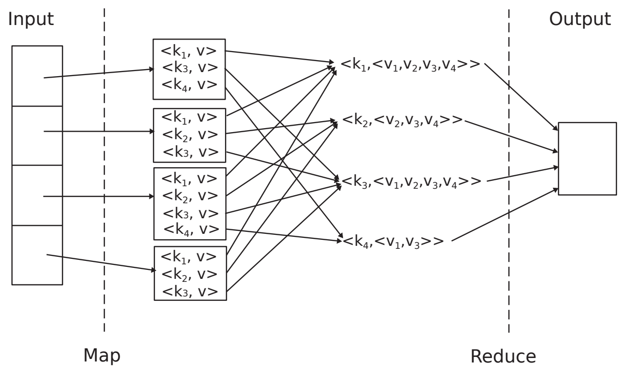


Fig. 1: Diagram of a generic MapReduce framework

Distributed File System (HDFS). HDFS replicates file data into multiple storage nodes that can concurrently access to the data.

Due to the increasing interest in MapReduce to solve data-intensive computing, and the necessity to overcome the computational and storage problems caused by mining patterns on Big Data, more and more authors are focusing their researches on this kind of paradigms. *Moens et al.* [14] have proposed a model, known as BigFIM, based on a breadth-first strategy. In BigFIM, each mapper receives a subset of the whole dataset and any of the patterns required to compute the support are returned. The reducer combines all local frequencies and report only the global frequent patterns. These frequent patterns are distributed to all mappers to act as candidates for the next step of the breadth-first search strategy. It should be noted that BigFIM is an iterative algorithm that needs to be repeated s times to obtain the patterns P of size $|P| = s$. *Moens et al.* [14] have also proposed the Dist-Eclat algorithm, an approach that works into three different steps and each of these steps are distributed among multiple mappers. In the first step, the dataset is divided into equally-sized blocks and distributed to the mappers. Here, each mapper extracts the frequent singletons, i.e. items of size one, from its chuck. In a second step, the algorithm distributes across the mappers all the frequent singletons extracted, and the aim is to find frequent patterns of size $|P| = s$ formed by the singletons. All this information is expressed as a prefix tree so frequent patterns can be extracted in a final step.

3 New proposals based on Apriori

In this section, we propose new efficient pattern mining algorithms to work in Big Data. All of them rely on the MapReduce framework and the Hadoop open-source implementation. Two of these algorithms (AprioriMR and IAprioriMR) enable any existing pattern to be discovered. Two additional algorithms (SPAprioriMR and TopAprioriMR) use a pruning strategy for mining frequent patterns. Finally, an algorithm for mining maximal frequent patterns on MapReduce (MaxAprioriMR) is also proposed.

3.1 Applying multiple reducers

It is worth mentioning that, when dealing with pattern mining on MapReduce, the number of key-value ($\langle k, v\rangle$) pairs obtained might be extremely high, and the use of a single reducer — as traditional MapReduce algorithms do — can be a bottleneck. Under these circumstances, the use of MapReduce in pattern mining is meaningless since the problem is still the memory requirements and the computational cost. To overcome this problem, we propose the use of multiple reducers, which represents a major feature of the algorithms proposed in this work. Each reducer works with a small set of keys in such a way that the performance of the proposed algorithms is improved.

The wrong use of multiple reducers in MapReduce implies that the same key k might be sent to different reducers, causing a lost of information. Thus, it is particularly important to previously fix the set of keys that will be analysed in each reducer. However, this redistribution process cannot be manually carried out due to the keys are not known beforehand and, besides, it is specially sensitive since those reducers that receive a huge number of keys will slow down the whole process.

In order to reduce the difference in number of $\langle k, v\rangle$ pairs analysed by each reducer, and considering that the singletons that form the item-sets are always in the same order, i.e. $\mathcal{I} = \{i_1, i_2, ..., i_n\} \Leftrightarrow 1 < 2 < ... < n$, a special procedure is proposed as follows. In a first reducer, any item-set that comprises the item i_1 is considered. From the remains, those that comprise the item i_2 are placed into a second reducer. The process is repeated according to the number of desirable reducers, in such a way that the remaining set of item-sets is combined in a final reducer. As a matter of example, let us consider a dataset comprising 20 singletons and three different reducers. Here, the first reducer will combine a maximum of $2^{19} = 524,288$ pairs of $\langle k, v\rangle$; the second reducer will combine a maximum of $2^{18} = 262,144$ pairs; and, finally, the third reducer will combine a maximum of $2^{17} + 2^{16} + ... + 2^0 = 262,143$ pairs.

3.2 Apriori versions without space pruning

First Apriori version proposed by *Agrawal et al.* [10] is based on the extraction of any item-set available in data (see Algorithm 1). In this well-known algorithm, *Agrawal et al.* proposed to generate all the feasible item-sets in each transaction and to assign a support of one to them (see lines 4 and 5, Algorithm 1). Then, the algorithm checks whether each of the new item-sets were already generated by previous transactions and, if so, their support is increased in a unity (see lines 6 to 8, Algorithm 1). The same process is repeated for each transaction (see lines 2 to 11, Algorithm 1), giving rise to a list L of patterns including their support or frequency of occurrence. As shown, the higher the number of both transactions and singletons, the higher the

Algorithm 1 Original Apriori algorithm.

```
Input: $\mathcal{T}$   // set of transactions
Output: $L$   // list of patterns found in data
 1: $L = \emptyset$
 2: for each $t \in \mathcal{T}$ do
 3:     for ($s = 1; s \leq |t|; s$++) do
 4:         $C = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \subseteq t \wedge |P| = s\}$
            // candidate item-sets in $t$
 5:         $\forall P \in C$, then $support(P) = 1$
 6:         if $C \cap L \neq \emptyset$ then
 7:             $\forall P \in L : P \in C$, then $support(P)$++
 8:         end if
 9:         $L = L \cup \{C \setminus L\}$   // include new patterns in $L$
10:     end for
11: end for
12: return $L$
```

computational complexity and the memory requirements. It is noteworthy that a significant number of transactions implies a huge number of iterations (see line 2, Algorithm 1) and, therefore, a drop in the runtime. At the same time, a high number of singletons entails a huge number of candidate item-sets in C (see line 4, Algorithm 1), increasing both the computational complexity and the memory requirements.

For the sake of reducing the time required by this intensive computing problem, we propose a new Apriori version based on MapReduce (see Algorithm 2). This first model based on MapReduce, hereinafter called Apriori MapReduce (AprioriMR), mines the whole set of feasible patterns in the database. AprioriMR works by running a mapper for each specific sub-database and each of these mappers is responsible for mining the complete set of item-sets for each sub-database (see AprioriMapper procedure, Algorithm 2). Here, each pattern P from a transaction $t_l \in \mathcal{T}$ is represented in the form $\langle k, v\rangle$, where the key k is the pattern P, whereas the value v is the support of P from the l-th transaction. After that, the shuffle and sort procedure is run, which groups different $\langle P, supp(P)_l\rangle$ pairs by the same key P, and pairs in the form $\langle P, \langle supp(P)_l, ..., supp(P)_m\rangle\rangle$ are produced. Finally, as described in previous sections, a number of reducers is run instead of using a single one as most of the MapReduce implementations do. Each reducer is responsible for combining the set of values representing the supports calculated by the mappers. Hence, for each pair $\langle P_l, \langle supp(P)_1, supp(P)_2, ..., supp(P)_m\rangle\rangle$, the result is a $\langle P, supp(P)\rangle$ pair in such a way that $supp(P) = \sum_{l=1}^{m} supp(P)_l$ (see lines 2 to 4, AprioriReducer, Algorithm 2).

Figure 2 illustrates how the AprioriMR algorithm works. In this example, the input database is divided into four sub-databases, and the AprioriMR algorithm includes four mappers and three reducers. As shown, each mapper mines item-sets (patterns) for its sub-dataset iterating transaction by transaction, producing a set of $\langle P, supp(P)_l\rangle$ pairs for each transaction $t_l \in \mathcal{T}$. Then, in an internal MapReduce grouping procedure, $\langle P, supp(P)_l\rangle$ pairs are grouped by the key P producing

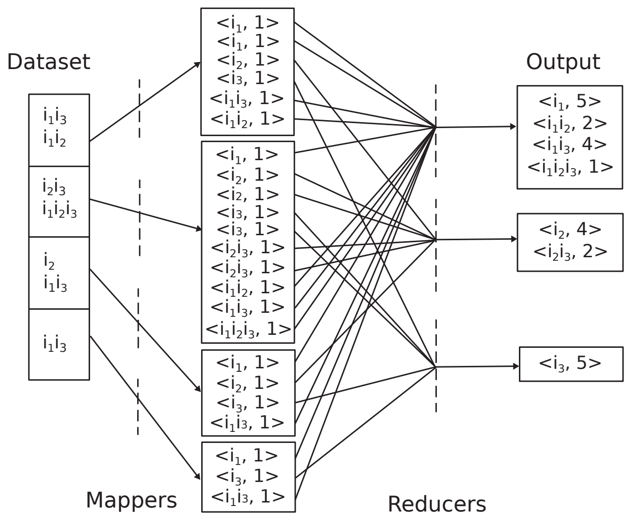

Dataset
i_1i_3 i_1i_2
i_2i_3 $i_1i_2i_3$
i_2 i_1i_3
i_1i_3
<i_1, 1> <i_1, 1> <i_2, 1> <i_3, 1> <i_1i_3, 1> <i_1i_2, 1>
<i_1, 1> <i_2, 1> <i_2, 1> <i_3, 1> <i_3, 1> <i_2i_3, 1> <i_2i_3, 1> <i_1i_2, 1> <i_1i_3, 1> <$i_1i_2i_3$, 1>
<i_1, 1> <i_2, 1> <i_3, 1> <i_1i_3, 1>
<i_1, 1> <i_3, 1> <i_1i_3, 1>
Mappers
Reducers
Output
<i_1, 5> <i_1i_2, 2> <i_1i_3, 4> <$i_1i_2i_3$, 1>
<i_2, 4> <i_2i_3, 2>
<i_3, 5>

Fig. 2: Diagram of the AprioriMR algorithm

$\langle P, \langle supp(P)_1, supp(P)_2, ..., supp(P)_m\rangle\rangle$ pairs. Hence, taking the item-set $\{i_1i_3\}$ as a key, the following pair is obtained $\langle\{i_1i_3\}, \langle 1, 1, 1, 1\rangle\rangle$. As described in Algorithm 2, the reducer phase is responsible for combining the values (supports for each item-set) and produce a final global support. Hence, taking the aforementioned item-set $\{i_1i_3\}$, the resulting value will be $\langle\{i_1i_3\}, 4\rangle$. Finally, it is noteworthy to mention that, as described in Section 3.1, each reducer computes different keys k. Hence, the first reducer computes those item-sets comprising i_1; the second reducer computes those item-sets including i_2; and, finally, i_3 is computed by the third reducer.

A important handicap of AprioriMR is the extremely high number of $\langle k, v\rangle$ pairs that it may generate, which is related to the number of singletons available in data. To deal with this issue a new algorithm is proposed (see Algorithm 3) —hereinafter called Iterative Apriori MapReduce (IAprioriMR). It does not create the whole set of candidate item-sets C for each transaction $t_l \in \mathcal{T}$ but the subset $c \subseteq C$ comprising patterns of size $|P| = s$.

Algorithm 2 AprioriMR algorithm.

```
begin procedure AprioriMapper($t_l$)
1: for ($s = 1; s \leq |t_l|; s$++) do
2:     $C = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \subseteq t_l \wedge |P| = s\}$
       // candidate item-sets in $t_l$
3:     $\forall P \in C$, then $supp(P) = 1$ // support is initialized
4:     for each $P \in C$ do
5:         emit $\langle P, supp(P)_l\rangle$    // emit the $\langle k, v\rangle$ pair
6:     end for
7: end for
end procedure

// In a grouping procedure values $supp_l$ are grouped for each pattern $P$, producing pairs $\langle P, \langle supp_1, supp_2, ..., supp_m\rangle\rangle$

begin procedure AprioriReducer($\langle P, \langle supp(P)_1, ..., supp(P)_m\rangle\rangle$)
1: $support = 0$
2: for each $supp \in \langle supp(P)_1, supp(P)_2, ..., supp(P)_m\rangle$ do
3:     $support$ += $supp$
4: end for
5: emit $\langle P, support\rangle$
end procedure
```

Algorithm 3 IAprioriMR algorithm.

```
begin procedure IAprioriMapper($t_l$, $s$)
1: $C = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \subseteq_l \wedge |P| = s\}$
   // candidate item-sets of size $s$ in $t_l$
2: $\forall P \in C$, then $supp(P)_l = 1$
3: for each $P \in C$ do
4:     emit $\langle P, supp(P)_l\rangle$    // emit the $\langle k, v\rangle$ pair
5: end for
end procedure

// In a grouping procedure values $supp_l$ are grouped for each pattern $P$, producing pairs $\langle P, \langle supp_1, supp_2, ..., supp_m\rangle\rangle$

begin procedure IAprioriReducer($\langle P, \langle supp(P)_1, ..., supp(P)_m\rangle\rangle$)
1: $support = 0$
2: for each $supp \in \langle supp(P)_1, supp(P)_2, ..., supp(P)_m\rangle$ do
3:     $support$ += $supp$
4: end for
5: emit $\langle P, support\rangle$
end procedure
```

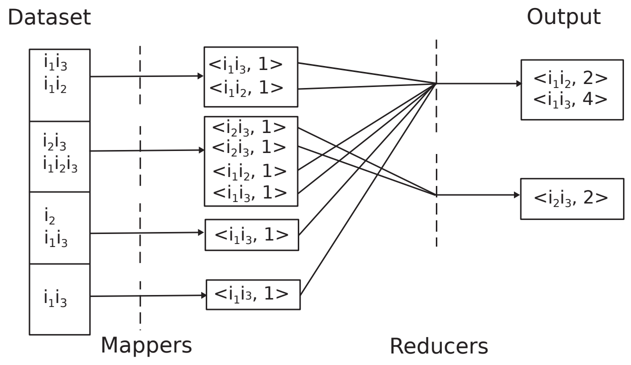


Fig. 3: Diagram of the proposed IAprioriMR algorithm when the second iteration is run, i.e., $\{\forall P, |P| = 2\}$.

Hence, a set of iterations is required, one per different item-set-size. In such a way, IAprioriMR includes the same features as AprioriMR, i.e. the dataset is divided into different chunk of data, one per mapper, and each mapper is responsible for analysing each single transaction $t_l \in \mathcal{T}$ to generate $\langle P, support(P)_l \rangle$ pairs. Finally, it also includes multiple reducers to scale down the computational cost. The main difference between AprioriMR and IAprioriMR lies in the mapper, since IAprioriMR obtains any pattern P of size $|P| = s$ for each sub-database (see line 1, IAprioriMapper procedure, Algorithm 3). In this regard, the number of $\langle P, support(P)_l \rangle$ pairs produced by each mapper is lower than those produced by the AprioriMR algorithm. However, IAprioriMR requires to iterate a number of times (one per different s) to produce the whole set of patterns available in data. For a better understanding, Figure 3 illustrates the diagram of the proposed IAprioriMR algorithm when patterns of size 2 are mined. In this example, the input database is divided into 4 sub-databases, one per mapper. As shown, each mapper mines any existing pattern P in each transaction $t_l \in \mathcal{T}$, producing $\langle P, support(P)_l \rangle$ pairs in the same way as AprioriMR does. Finally, similarly to AprioriMR, the reducer phase is responsible for combining the set of values representing the supports calculated by the mappers.

3.3 Apriori versions with space pruning

The task of finding all patterns in a database is quite challenging since the search space exponentially increases with the number of single items occurring in the database. As described in previous sections, given a dataset comprising n singletons, a maximum number of $2^n - 1$ different patterns can be found in that dataset. Additionally, this dataset might contain plenty of transactions and the process of calculating the frequency for each pattern might be considered as a tough problem [13]. All of this highlights the importance of pruning the search space, given rise to the paradigm of constraint-based mining [3]. A really important pruning strategy in the pattern mining field is anti-monotone property, which determines that any sub-pattern of a frequent pattern is also frequent, and any super-pattern of an infrequent pattern will be never frequent.

Apriori may also include a space pruning strategy by using a level-wise paradigm in which all the frequent patterns of size $|P| = s$ are generated by using all the frequent patterns of size $s-1$. Thus, the main characteristic of Apriori is that every subset of a frequent pattern is also frequent, so it follows the anti-monotone property. In order to obtain new patterns of size $|P| = s$, the algorithm uses joins of singletons and frequent patterns of length $s - 1$ (see Algorithm 4).

Algorithm 4 Apriori algorithm with anti-monotone property.

```
Input: $\mathcal{T}, thr$   //set of transactions and support threshold
Output: $L$   // list of patterns found in data
1: $I = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \in \mathcal{T} \wedge |P| = 1 \wedge support(P) \geq thr\}$ // frequent singletons
2: $L_1 = I$   // $I$ as the set of single items in data
3: for ($s = 2; s \leq |I|; s$++) do
4:     $C$ = generate item-sets of size $s$ from $L_{(s-1)}$ and $I$
5:     for each $t \in \mathcal{T}$ do
6:         $\forall P \in C : P \subseteq t$, then $support(P)$++
7:     end for
8:     $L_j = \{\forall P \in C : support(P) \geq thr\}$
9: end for
10: $L = \cup_{k=1}^{|I|} L_k$
11: return $L$
```

Algorithm 5 SPAprioriMR algorithm.

```
begin procedure SPAprioriMapper($t_l$, $s$)
1: $L_{inf}$ = load the list of infrequent item-sets of size $s - 1$
2: $C = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \subseteq t_l \wedge |P| = s, \nexists j \in L_{inf} : j \subset P\}$ // candidate patterns of size $s$ in $t_l$ such as they do not include any infrequent pattern from $L_{inf}$
3: $\forall P \in C$, then $supp(P)_l = 1$
4: for each $P \in C$ do
5:     emit $\langle P, supp(P)_l \rangle$   // emit the $\langle k, v \rangle$ pair
6: end for
end procedure

// In a grouping procedure values $supp_m$ are grouped for each pattern $P$, producing pairs $\langle P, \langle supp_1, supp_2, ..., supp_m \rangle \rangle$

begin procedure SPAprioriReducer($\langle P, \langle supp(P)_1, ..., supp(P)_m \rangle \rangle$)
1: $support = 0$
2: for each $supp \in \langle supp(P)_1, supp(P)_2, ..., supp(P)_m \rangle$ do
3:     $support$ += $supp$
4: end for
5: if $support \geq threshold$ then
6:     emit $\langle P, support \rangle$
7: else
8:     keep $\langle P, support \rangle$ into the list of infrequent patterns
9: end if
end procedure
```

In this section, we propose an iterative approach based on MapReduce (see Algorithm 5) to mine frequent patterns by considering minimum support threshold as a search space pruning. This proposal, named SPAprioriMR (Space Pruning Apriori MapReduce), starts by mining the frequent singletons, i.e. $\{\forall i \in I : |i| = 1 \wedge support(i) \geq threshold\}$, in a dataset by following the MapReduce methodology. In the next iteration, SPAprioriMR mines any pattern of size 2 that can be obtained when infrequent singletons are not considered to produce new item-sets. The process is repeated in such a way that, in the s-th iteration, the algorithm produces patterns of size $|P| = s$ by not considering those infrequent patterns of size $s - 1$.

For a better understanding, Figure 4 shows the iterations required for the SPAprioriMR algorithm for a sample dataset.

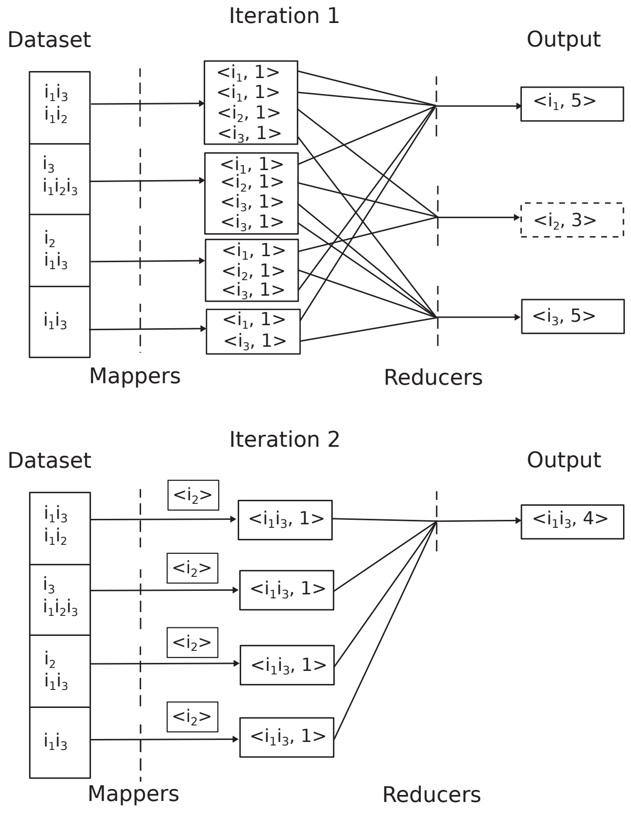


Fig. 4: Iterations run by the SPAprioriMR algorithm on a sample dataset. The minimum support threshold was fixed to a value of 4.

Here, a minimum support threshold of 4 is considered, so any pattern or item-set having a support less than this value is marked as infrequent (dashed line). The algorithm mines all the singletons in the first iteration, discovering that i_2 is infrequent because it is satisfied by only 3 transactions. In the next iteration, the aim of each mapper is two-fold, first it receives two sets: infrequent patterns discovered in the previous iteration and a chunk of data. From these two sets and analysing the transaction $t_l \in \mathcal{T}$, it obtains the set of $\langle P, support(P)_l \rangle$ pairs. As shown in Figure 4, the algorithm stops once the maximum size for the frequent patterns is reached —no additional pattern can be obtained from the set $\{\{i_1\}, \{i_3\}, \{i_1i_3\}\}$.

SPAprioriMR considers the monotonic property since an infrequent pattern P will not be considered any longer. Nevertheless, it still needs high computation requirements, specially when the size of the infrequent patterns is too high. In such a situation, a high number of iterations is required to obtain all the frequent patterns, hampering the mining process. Thus, for the sake of reducing the number of frequent patterns in situations where this value is too high, we propose a new algorithm (see Algorithm 6), known as Top Apriori MapReduce (TopAprioriMR), that does not always obtain the complete set of frequent patterns but it guarantees that the resulting set comprises those patterns with the highest support values. TopAprioriMR works similarly to SPAprioriMR, i.e. it requires a set of iterations and each iteration is responsible for mining the set of frequent patterns of size s. The main feature of TopAprioriMR is its procedure for updating the input database by removing all the infrequent item-sets found (lines 1 to 2, TopAprioriMapper procedure, Algorithm 6). It

Algorithm 6 TopAprioriMR algorithm.

```
begin procedure TopAprioriMapper($t_l$, $s$)
1: $L_{inf}$ = load the list of infrequent patterns of size $s - 1$
2: $t_l$ = update $t_l$ by removing all the infrequent item-sets found
3: $C = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \subseteq t_l \wedge |P| = s\}$
   // candidate patterns of size $s$ in $t_l$
4: $\forall P \in C$, then $supp(P)_l = 1$
5: for each $P \in C$ do
6:     emit $\langle P, supp(P)_l \rangle$    // emit the $\langle k, v \rangle$ pair
7: end for
end procedure

// In a grouping procedure values $supp_m$ are grouped for each pattern $P$, producing pairs $\langle P, \langle supp_1, supp_2, ..., supp_m \rangle \rangle$

begin procedure TopAprioriReducer($\langle P, \langle supp(P)_1, ..., supp(P)_m \rangle \rangle$)
1: $support = 0$
2: for each $supp \in \langle supp(P)_1, supp(P)_2, ..., supp(P)_m \rangle$ do
3:     $support$ += $supp$
4: end for
5: if $support \geq threshold$ then
6:     emit $\langle P, support \rangle$
7: else
8:     keep $\langle P, support \rangle$ into the list of infrequent patterns
9: end if
end procedure
```

implies a smaller dataset throughout the iterations, reducing the computational time and the memory requirements. As shown in Algorithm 6, the TopAprioriReducer procedure is completely the same as SPAprioriReducer.

Figure 5 shows a sample running of the TopAprioriMR algorithm by considering a minimum support threshold value of 4. In the first iteration, it analyses all the singletons that could be found in the database. The map procedure is responsible for generating all the $\langle k, v \rangle$ pairs from each specific transaction, in such a way that the key k is formed by each singleton. Then, similarly to the other proposed algorithms, an internal MapReduce grouping procedure is carried out to produce $\langle k, \langle v_1, v_2, ..., v_m \rangle \rangle$ pairs. In this example, taking the singleton $\{i_3\}$ as a key, the following pair is obtained $\langle \{i_3\}, \langle 1, 1, 1, 1, 1 \rangle \rangle$. Finally, the reducer phase is responsible for merging all the values associated with the same key k and producing the final support values, i.e. $\langle \{i_3\}, 5 \rangle$. It is noteworthy to mention that, as described in Section 3.1, each reducer computes different keys k. Hence, the first reducer computes those patterns comprising i_1; the second reducer computes those patterns including i_2; and, finally, i_3 is computed by the third reducer. Whereas all the frequent patterns are emitted by the reducer (lines 5 to 7, TopAprioriMapper procedure, Algorithm 6), infrequent patterns are stored in an external file (lines 7 to 9, TopAprioriMapper procedure, Algorithm 6) in order to update the database in the next iteration.

Following with the second iteration, and similarly to the previous iteration, the TopAprioriMR algorithm distributes the dataset into mappers, and the previous resulting set of infrequent patterns are also loaded (line 1, TopAprioriMapper procedure, Algorithm 6). Here, each mapper is responsible for analysing the input and making an updating process, removing from data the infrequent patterns discovered in the previous iteration (line 2, TopAprioriMapper procedure, Algorithm 6). Then, each mapper

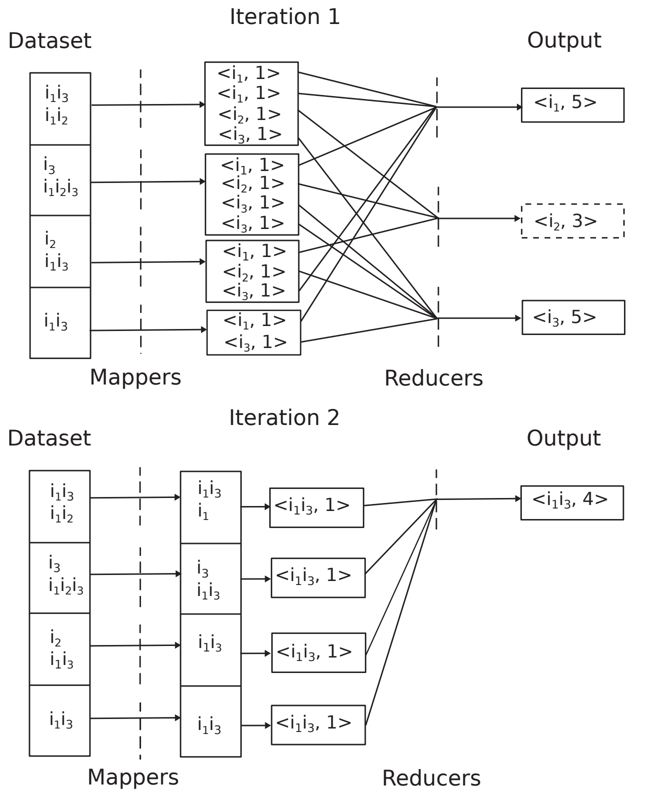


Fig. 5: Iterations run by the TopAprioriMR algorithm on a sample dataset. The minimum support threshold was fixed to a value of 4.

produces any pattern P of size $|P| = 2$ that can be derived from the specific transaction $t_l \in \mathcal{T}$. The final step of this second iteration is the reducer procedure, producing $\langle\{i_1, i_3\}, 4\rangle$ as a frequent pattern.

3.4 Maximal frequent patterns

In many situations, it is interesting to obtain a small set of frequent patterns instead of the whole set of them. In this regard it is possible to use the Apriori algorithm for mining maximal frequent patterns, also known as condensed representations of frequent patterns. As previously stated, a maximal frequent pattern is formally defined as a frequent pattern P that satisfies that none of its immediate super-patterns P^S are frequent, i.e. $\{\nexists P^S \mid P \subset P^S \wedge\ support(P^S) \geq threshold\}$. Algorithm 7 shows the pseudo-code of the traditional Apriori algorithm for mining maximal frequent item-sets. It starts by obtaining a pattern comprising all the singletons in data (line 1, Algorithm 7). From this, the algorithm generates sub-patterns looking for those that are frequent according to a minimum support threshold thr. If a new sub-pattern comprises items that are maximals, then this sub-pattern is discarded (line 9, Algorithm 7). The algorithm continues in an iterative way till no sub-pattern can be obtained from C, i.e. $C = \emptyset$.

In order to reduce the computational time required by the Apriori version, we present a novel proposal (see Algorithm 8) based on MapReduce paradigm. The proposed algorithm, called MaxAprioriMR (Maximal Apriori MapReduce), starts by mining the set of patterns $\mathcal{P}$ comprising patterns P of size $|P| = s$, in such a way that no super-pattern of size $s+1$ could be mined. For

Algorithm 7 Apriori algorithm for maximal frequent patterns.

```
Input: $\mathcal{T}, thr$    // set of transactions and support threshold
Output: $L$    // list of maximal frequent patterns found in data
 1: $C = \cup\{\forall P : P = \{i_j, ..., i_n\} \wedge P \in \mathcal{T} \wedge |P| = 1\}$
    // item-set comprising all the singletons
 2: for $(; |C| \geq 1; )$ do
 3:     for each $t \in \mathcal{T}$ do
 4:         $\forall P \in C : P \subseteq t$, then $support(P)$++
 5:     end for
 6:     $C = \{\forall P \in C : support(P) < thr\}$
 7:     $L = L \cup \{\forall P \in C : support(P) \geq thr\}$
 8:     $C$ = generate sub-patterns of size $|P| - 1$ from $C$
 9:     $C = \{\forall P \in C : P \nsubseteq L\}$
10: end for
11: return $L$
```

Algorithm 8 MaxAprioriMR algorithm.

```
begin procedure MaxAprioriMapper($t_l$, $s$)
 1: $L_{inf}$ = load the list of infrequent patterns of size $s - 1$
 2: $t_l$ = update $t_l$ by removing all the infrequent item-sets found
 3: $C = \{\forall P : P = \{i_j, ..., i_n\} \wedge P \subseteq t_l \wedge |P| = s\}$
    // candidate patterns of size $s$ in $t_l$
 4: $\forall P \in C$, then $supp(P)_l = 1$
 5: for each $P \in C$ do
 6:     emit $\langle P, supp(P)_l\rangle$    // emit the $\langle k, v\rangle$ pair
 7: end for
end procedure

// In a grouping procedure values $supp_m$ are grouped for each
pattern $P$, producing pairs $\langle P, \langle supp_1, supp_2, ..., supp_m\rangle\rangle$

begin procedure
MaxAprioriReducer($\langle P, \langle supp(P)_1, ..., supp(P)_m\rangle\rangle$)
 1: $support = 0$
 2: for each $supp \in \langle supp(P)_1, supp(P)_2, ..., supp(P)_m\rangle$
    do
 3:     $support$ += $supp$
 4: end for
 5: if $support \geq threshold$ then
 6:     emit $\langle P, support\rangle$
 7: else
 8:     keep $\langle P, support\rangle$ into the list of infrequent item-sets
 9: end if
end procedure
```

a better understanding, Figure 6 shows the algorithm iterations for a sample dataset. Here, the minimum support threshold value is fixed to 4, so a pattern P is marked as maximal (solid line) iff it is frequent ($support(P) \geq 4$) and it has no frequent super-pattern ($\nexists P^s : P \subset P^s \wedge support(P^s) \geq 4$). Once a maximal frequent pattern of size $|P| = s$ is obtained, then this pattern is kept into an external file that comprises the set of maximal frequent patterns discovered. This set of maximal frequent patterns is used in a second iteration, i.e., in the mining of those maximal frequent patterns of size $|P| = s - 1$. In this algorithm, the aim of each mapper is two-fold: first it removes any sub-pattern comprised into a maximal frequent pattern (line 3, Algorithm 8); second, it mines any existing frequent pattern of size $s-1$ from the resulting dataset (line 2, Algorithm 8) — this set of discovered frequent pattern is

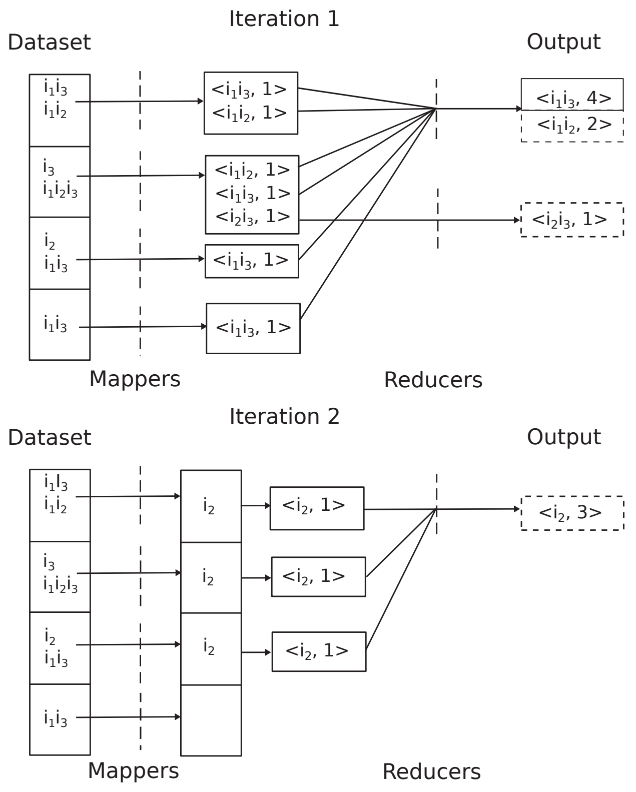


Fig. 6: Iterations run by the MaxAprioriMR algorithm on a sample dataset. The minimum support threshold was fixed to a value of 4.

maximal since none of its previous super-patterns are frequent. Thus, the resulting set of maximal frequent patterns comprises the pattern $\{i_1i_3\}$, from which we could obtain the following frequent patterns $\{\{i_1\},\{i_3\},\{i_1i_3\}\}$.

4 Experiments

In this section, the performance of the proposed algorithms is studied for different data sizes. The goal of this study is fourfold:

- Analysis of the computational time for a varied set of algorithms, comprising both sequential and MapReduce pattern mining algorithms.
- Evaluation of the performance of pattern mining algorithms when a pruning strategy is considered.
- Analysis of the performance of algorithms for mining condensed representations of frequent item-sets.
- Study of the accuracy in the resulting set of solutions.

All the experiments were run on a HPC cluster comprising 12 compute nodes. Each of the nodes comprised two Intel Xeon E5645 CPUs with 6 cores at 2.4 GHz and 24 GB DDR memory. Cluster operating system was Rocks cluster 6.1 x64. As for the specific details of the software used, the experiments have been run on Hadoop 2.6, with a maximum of 144 maps tasks, and a maximum of 3 reducers.

It is noteworthy that the runtime obtained for all the algorithms in this experimental analysis is the average runtime obtained for 10 different runs. The aim of this average is to alleviate the variations in runtime derived from the resource management.

4.1 Datasets used

This experimental section considers a large number of different Big Datasets comprising either synthetic and real-world datasets. The goal of this study is to analyse the performance of different algorithms for mining frequent patterns, so the datasets considered highly vary with the number of both instances and singletons.

In a first analysis, the datasets used comprise a search space that varies from 2^4 to 2^{20}. The number of instances considered in this first analysis varies from $2 \cdot 10^6$ to $2 \cdot 10^8$ instances. As for the file-size, it varies from 9.6 MB to 2.7 GB. As for the mining of maximal frequent patterns, datasets comprising a size in the search space up to 2^{18} is considered. Here, the number of distinct instances varies from $3 \cdot 10^7$ to $3 \cdot 10^{12}$. Analysing the file size, it is noteworthy that the smallest dataset has a size of 86 MB, whereas the biggest one has a file size of 814 GB. Finally, we have conducted experiments on real world datasets considering the *webdoc* dataset[1], which is the one comprising the highest number of singletons — it comprises a search space of $2^{5,267,646}$. This datasets has a file size of 1.5 GB.

4.2 Sequential Mining Vs MapReduce

In this experimental study, the performance of a set of algorithms when the number of both attributes and instances increases have been analysed. In this study, a set of synthetic datasets have been properly created to analyse how the number of both instances and attributes affects the algorithms' performance. Single-items generated for these datasets are normally distributed by following a Gaussian distribution throughout the whole set of instances. The number of single-items varies from 8 to 20, whereas the number of instances varies from $2{\cdot}10^6$ to $2{\cdot}10^8$. In this regard, the computational cost varies from $O(2^8 - 1 \times 2 \cdot 10^6 \times 8)$ to $O(2^{20} - 1 \times 2 \cdot 10^8 \times 20)$.

In a first analysis, the runtime of different proposals when no pruning strategy is compared. In this analysis (see Figure 7), a dataset comprising a search space of size $2^8 - 1$ is considered.

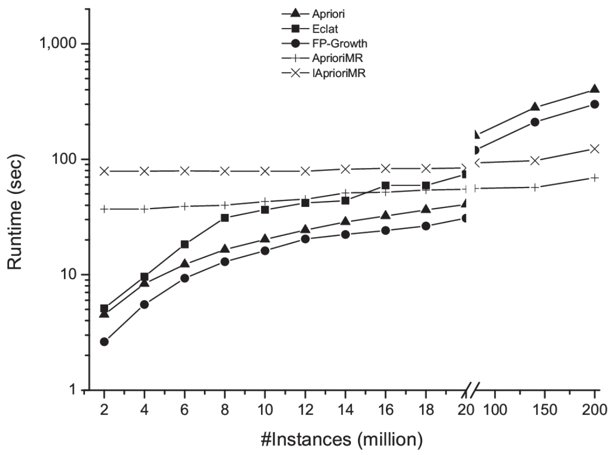


Fig. 7: Runtime of different algorithms when they are run on datasets comprising 8 singletons and a number of instances that varies from $2{\cdot}10^6$ to $2{\cdot}10^8$. The computational complexity is up to $O(2^8 - 1 \times 2 \cdot 10^8 \times 8)$.

[1]This dataset is considered by the pattern mining community as the biggest one, which is electronically available at the website: http://fimi.ua.ac.be/data/

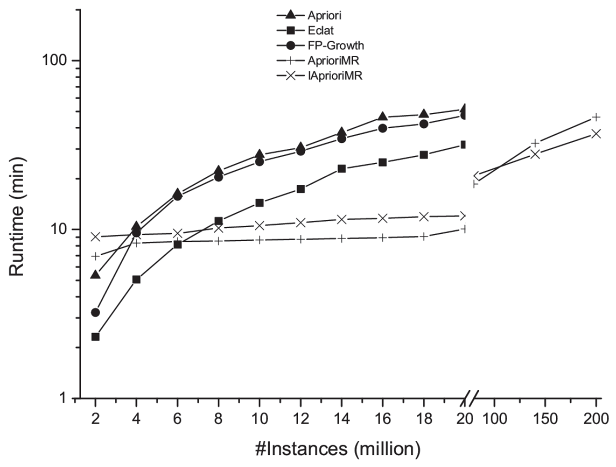


Fig. 8: Runtime of different algorithms when they are run on datasets comprising 20 singletons and a number of instances that varies from $2{\cdot}10^6$ to $2{\cdot}10^8$. The computational complexity is up to $O(2^{20}-1 \times 2\cdot 10^8 \times 20)$.

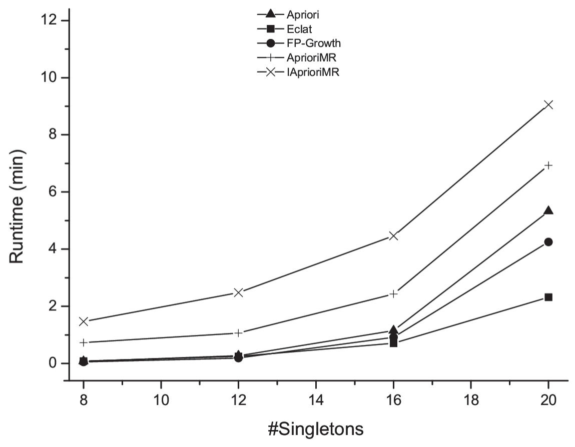


Fig. 9: Runtime of different algorithms when they are run on datasets comprising 2 million of instances and a number of singletons that varies from 8 to 20. The computational complexity is up to $O(2^{20}-1 \times 2\cdot 10^6 \times 20)$.

Results illustrate that sequential algorithms like Apriori, FP-Growth or Eclat behave really well even when datasets includes a large number of instances (up to 20 million). In situations where the number of instances continues growing, the proposed Apriori versions based on MapReduce (AprioriMR and IAprioriMR) behave better than the others. Results reveal that, for a dataset comprising more than 80 million of instances and 8 single-items, the use of MapReduce version is appropriate, outperforming sequential algorithms (FP-Grwoth, Apriori and Eclat). Continuing the analysis, and focusing on the runtime of MapReduce versions, it is obtained that the number of instances barely affect the runtime of the proposed algorithms. This behaviour involves that, for huge datasets, the use of MapReduce is alluring. Finally, it is interesting to note that Eclat does not appear in the analysis for datasets comprising more than 80 million of instances due to its low performance. To understand this issue, it is worth noting that Eclat includes the whole tid-list for each pattern, so extremely large datasets (according to the number of instances) cannot be handled into memory (hardware limitations).

Following with the experimental analysis, the number of singletons is increased from 8 to 20 so the search space growths from $2^8-1=255$ to $2^{20}-1=1,048,575$. Figure 8 illustrates the runtime for the set of algorithms considered in this study, denoting the good performance of MapReduce versions. Results reveal the importance of using the MapReduce paradigm for mining patterns on huge search spaces rather than using sequential pattern mining algorithms. It demonstrates that sequential mining algorithms are appropriate to low complex problems, requiring algorithms based on MapReduce to handle with tough tasks. In this regard, and considering datasets with more than 80 million of instances, none of the sequential versions considered in this study are able to be run (hardware limitations).

The second part of this experimental analysis is related to the number of single-items, so the aim is to demonstrate how the algorithms behave when this number increases. In this sense, Figure 9 illustrates the performance of the algorithms when datasets comprising 2 million of instances are considered, and the number of single-items varies from 8 to 20. Results demonstrate that small problems should be solved by sequential pattern mining algorithms (Apriori, FP-Growth and Eclat), which require a runtime lower than MapReduce versions. Following with this analysis, and considering now 20 million of instances (see Figure 10), the MapReduce versions tend to perform better when a high number of both instances and singletons are considered. Finally, and considering now 200 million of instances, it is obtained that sequential pattern mining algorithms cannot be run due to hardware limitations (see Figure 11). In this final analysis, it is obtained that MapReduce algorithms behave quite similar and they are really appropriate to handle huge datasets where an extremely large number of comparisons are required.

The final part of this experimental study is related to the orders of magnitude in the difference for the runtime (see Table 1). Let us consider AprioriMR as baseline. Here, it is obtained more than 0.5 order of magnitude in the average difference (mean value when considering different number of both instances and single-items) when AprioriMR is compared to sequential pattern mining algorithms. Taking the most favourable case, AprioriMR obtains

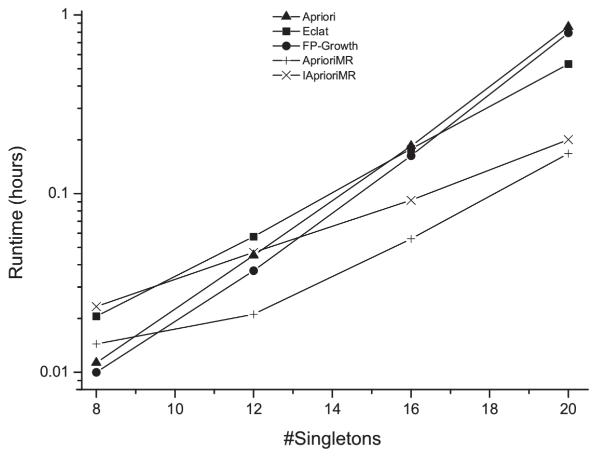


Fig. 10: Runtime of different algorithms when they are run on datasets comprising 20 million of instances and a number of singletons that varies from 8 to 20. The computational complexity is up to $O(2^{20}-1 \times 2\cdot 10^7 \times 20)$.

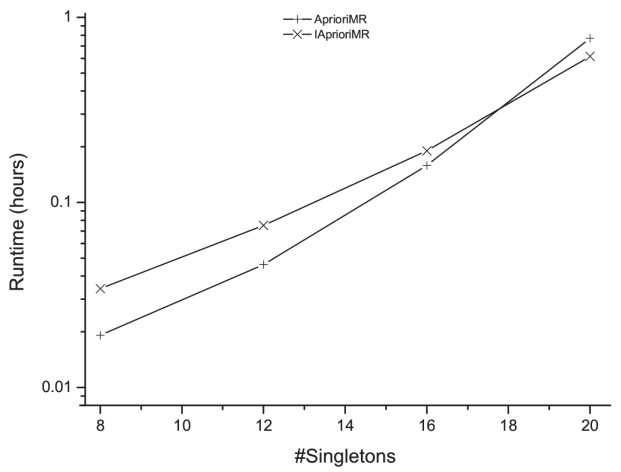


Fig. 11: Runtime of different algorithms when they are run on datasets comprising 200 million of instances and a number of singletons that varies from 8 to 20. The computational complexity is up to $O(2^{20} - 1 \times 2 \cdot 10^8 \times 20)$.

TABLE 1: Results show the orders of magnitude in the difference of the runtime when each pair of algorithms (no pruning strategy) is compared. Either the average, worst (the most unfavourable) and best case (the most favourable) have been considered.

Comparisons	Average	Worst	Best
AprioriMR vs Apriori	0.577	-1	2
AprioriMR vs FP-Growth	0.654	-1	2
AprioriMR vs Eclat	0.538	-1	2
AprioriMR vs IAprioriMR	0.231	0	1
IAprioriMR vs Apriori	0.346	-1	2
IAprioriMR vs FP-Growth	0.308	-1	2
IAprioriMR vs Eclat	0.538	-1	2
IAprioriMR vs AprioriMR	-0.231	-1	0

2 orders of magnitude in the difference with regard to Apriori, FP-Growth and Eclat. Continuing the analysis, let us compare the runtime of the two MapReduce Apriori versions for mining patterns without pruning the search space. In this study, results reveal that both algorithm similarly behave, obtaining an average difference in order of magnitude of 0.231 (mean value when considering different number of both instances and single-items). It is noteworthy to mention that negative orders of magnitude means that, in some datasets (small datasets comprising a small search space), the proposed MapReduce algorithms behave worse than sequential pattern mining algorithms. Finally, considering the IAprioriMR algorithm, it is obtained values in the average difference of the orders of magnitude lower than the ones obtained for AprioriMR. Nevertheless, to be fair, it should be noted that IAprioriMR performs better than AprioriMR for extremely complex problems (see Figure 11 when using more than 18 singletons).

4.3 MapReduce versions with space pruning

In this second experimental study, our aim is to analyse the behaviour of the proposed MapReduce Apriori versions when a pruning strategy is considered to reduce the search space. The aim of these algorithms is not the discovery of any existing pattern but a subset of interesting patterns according to a minimum predefined value of frequency. It is noteworthy to mention that this experimental analysis includes comparisons with regard to the two existing proposals (BigFIM and DistEclat) based on MapReduce for mining patterns of interest.

First, it is interesting to study the performance of the MapReduce algorithms when increasing the size $|P| = s$ of the discovered patterns. The aim is to determine whether the patterns' size is detrimental to the algorithms' performance. Figure 12 illustrates the resulting runtime required for mining patterns of different sizes $s = 1$, $s = 2$, $s = 3$ and $s = 4$. Results reveal that the size of the patterns does not affect the performance of the algorithms. Finally, it should be highlighted that either TopApriori and SPAprioriMR perform much better than existing proposals based on MapReduce for mining patterns (BigFIM and DistEclat) regardless the length of the patterns to be discovered.

Second, the performance of the proposed algorithms (SPAprioriMR and TopAprioriMR) when the minimum support threshold varies is analised. As depicted in Figure 13, for extremely low threshold values (huge search spaces) TopAprioriMR outperforms SPAprioriMR. However, when increasing the support threshold value (smaller search spaces), both TopAprioriMR and SPAprioriMR tend to equally behave.

Third, once it has been demonstrated that the proposed

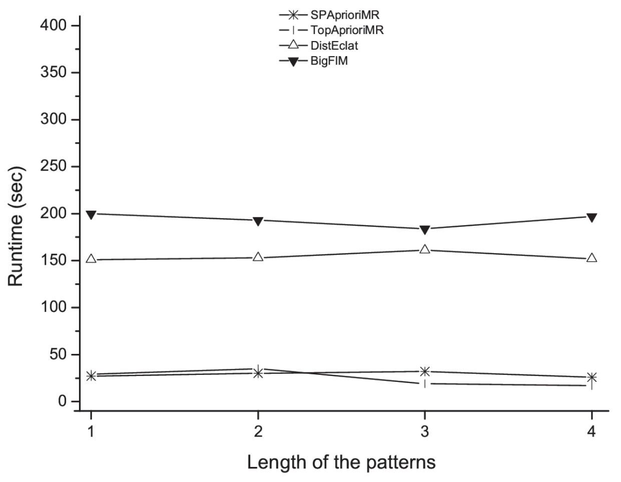


Fig. 12: Runtime required for mining all the patterns of size $s = 1$, $s = 2$, $s = 3$ and $s = 4$.

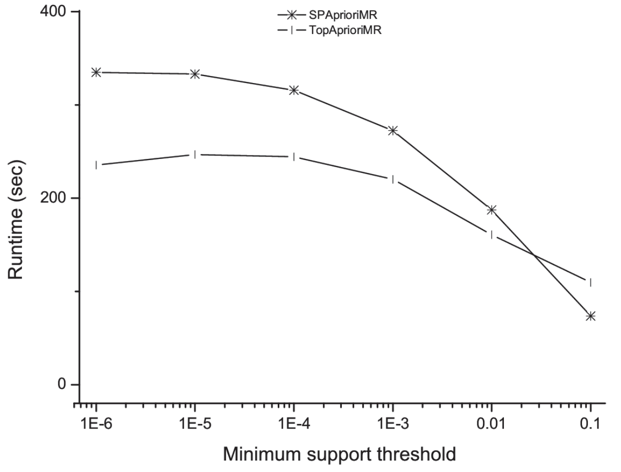


Fig. 13: Runtime when considering a dataset comprising 20 million of instances and 16 single-items, and considering different values for the minimum support threshold.

MapReduce versions outperform sequential pattern mining algorithms, we aim to study their behaviour for extremely large datasets —the number of singletons varies from 4 to 18 and the number of instances varies from $3{\cdot}10^7$ to $3{\cdot}10^8$ (the computational cost varies from $O(2^4 - 1 \times 3 \cdot 10^7 \times 4)$ to $O(2^{18} - 1 \times 3 \cdot 10^8 \times 18)$. In this analysis, the proposed algorithms with the state of the art in pattern mining using MapReduce have been compared, i.e. DistEclat and BigFIM. First results are illustrated in Figure 14, depicting that all the algorithms based on MapReduce similarly behave for 30 million of instances and up to 12 singletons. For bigger search spaces, DistEclat and TopAprioriMR outperform the others. As described in [14], DistEclat is a distributed version of Eclat [25] and it performs faster than BigFIM (as it is also demonstrated in our experimental results).

Continuing the analysis, the number of instances is increased in one order of magnitude, considering datasets comprising 300 million of instances. In this second analysis, the BigFIM algorithm has been discarded since it was demonstrated (see Figure 14) that DistEclat performs in a better way. Results for this second analysis are illustrated in Figure 15, showing that TopAprioriMR outperforms the other algorithms, specially for a number of single-items greater than 10. It is also interesting to note that the performance of SPAprioriMR seems to be logarithmic with regard to the increasing number of singletons.

Finally, the experimental analysis is focused on the TopAprioriMR algorithm, which is the one with the best results in performance. Taking this algorithm as baseline, it can be illustrated how its performance is detrimental to the variation of the number of compute nodes (see Figure 16). The ultimate aim of this study is to determine how dependent this algorithm is with regard to the hardware architecture. As shown the algorithm is highly dependent on the number of compute nodes, specially for a small number. However, it is fair to highlight that, when the number of compute nodes continues increasing, then the performance barely varies.

Continuing the analysis of the TopAprioriMR algorithm, and once its good performance on different scenarios have been demonstrated, it has been considered of interest to push it to the limit. In this sense, extremely large datasets are used (according to the number of single-items) comprising search spaces that varies from $2^{3\cdot10^4} - 1$ and $2^{7\cdot10^4} - 1$ patterns. Table 2 shows the runtime (in minutes) when TopAprioriMR is run on a datasets which complexity varies from from $O(2^{30,000} - 1 \times 3 \cdot 10^6 \times 30,000)$ to $O(2^{70,000} - 1 \times 3 \cdot 10^7 \times 70,000)$. Results reveal that TopAprioriMR is able to produce an output when it is pushed to the limit, mining frequent patterns (in almost 300 minutes) in a search space of size $2^{30,000} - 1$ and considering $3 \cdot 10^6$ of instances.

Finally, the TopAprioriMR algorithm is applied to the *web-*

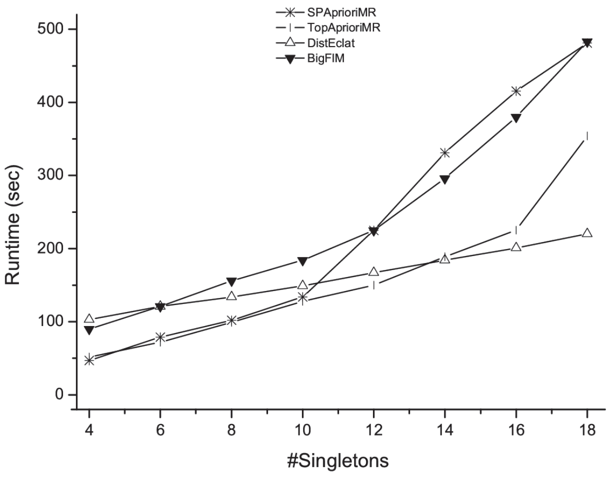


Fig. 14: Runtime of different algorithms when they are run on datasets comprising 30 million of instances and a number of singletons that varies from 4 to 18. The computational complexity is up to $O(2^{18} - 1 \times 3 \cdot 10^7 \times 18)$.

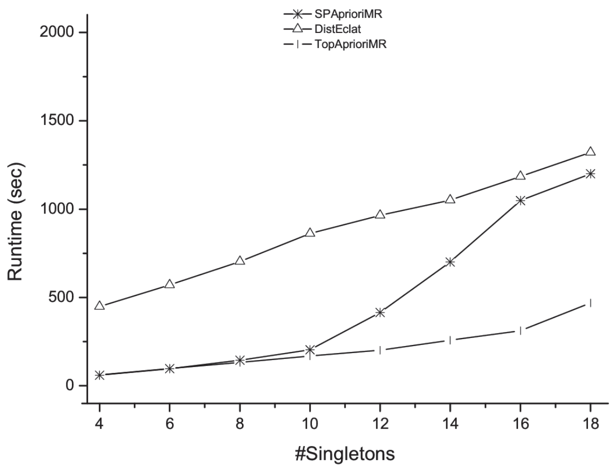


Fig. 15: Runtime of different algorithms when they are run on datasets comprising 300 million of instances and a number of singletons that varies from 4 to 18. The computational complexity is up to $O(2^{18} - 1 \times 3 \cdot 10^8 \times 18)$.

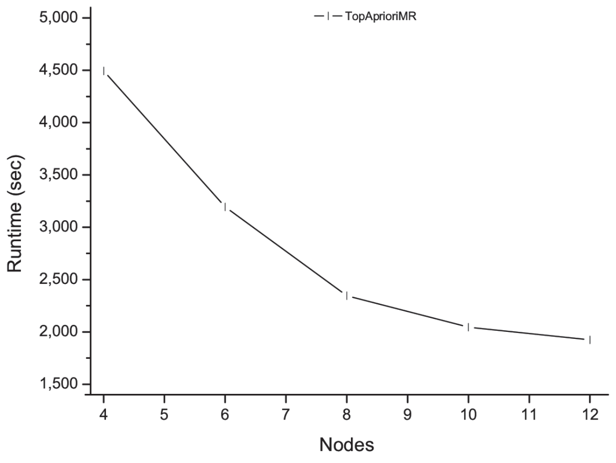


Fig. 16: Runtime required by the TopAprioriMR algorithm when the hardware architecture varies.

TABLE 2: Runtime in minutes of the TopAprioriMR algorithm when different datasets comprising a high number of both instances and single-items is considered.

Search space	Instances	Minimum Support	Runtime (min)	Frequent Patterns
$2^{30,000} - 1$	$3{\cdot}10^6$	0.0252	298.45	760
$2^{30,000} - 1$	$3{\cdot}10^7$	0.0251	594.15	36
$2^{70,000} - 1$	$3{\cdot}10^6$	0.0109	214.85	85
$2^{70,000} - 1$	$3{\cdot}10^7$	0.0108	1407.75	49

TABLE 3: Runtime in minutes of the TopAprioriMR algorithm when it is run on the well-known *webdocs* dataset.

Minimum Support	Runtime (min)	Frequent Patterns
0.01	307.63	400,951
0.20	192.68	319
0.50	177.76	9

docs dataset, which is the biggest dataset available in the FIM community [2]. This dataset comprises 5,267,656 single-items, which are distributed along 1,692,082 of instances.Table 3 shows the results obtained when applying TopAprioriMR, denoting the runtime (in minutes) and the number of patterns discovered for different minimum support threshold values. Results demonstrate the good performance of TopAprioriMR, which is able to discover frequent patterns in close to 300 minutes in a search space of size $2^{5,267,656}$.

4.4 Maximal frequent pattern mining

Finally, an experimental analysis, in which is studied the performance of algorithms for mining maximal frequent patterns, is carried out. In this regard, the study carried out is focused on a comparison between the proposed MaxAprioriMR and FP-Max [20], which is considered as the most efficient algorithm for mining maximal patterns — according to the experimental results described in the repository of the workshops on Frequent Itemset Mining Implementations (http://fimi.ua.ac.be/). Figure 17 shows the runtime in seconds when either FPMax and MaxAprioriMR are run on datasets comprising a different number of instances. Results reveal that, for a small number of instances, FPMax outperformns our Apriori version based on MapReduce. It demonstrates that MapReduce is not suitable for simple problems, but when increasing the computational complexity — according to the number of instances— its use is completely justified.

A major feature of any algorithm for mining maximal frequent patterns is its ability to work on a reduced set of solutions. Each maximal frequent pattern P of size $|P| = s$ implies the presence of $2^s - 2$ subsets of frequent patterns that can be omitted. Hence, each time a maximal frequent pattern is discovered, the search space is reduced. Nevertheless, the procedure for mining these patterns is time-consuming and, in many cases, it is possible to mine the whole set of frequent patterns faster than the set of maximal frequent patterns. To demonstrate this assertion, Figure 18 shows a comparison between the MaxAprioriMR algorithm and TopAprioriMR, which is the MapReduce proposal that achieved better results. Results demonstrate that the number of single-items has a high impact on the performance. Let us considered a dataset comprising 12 single-items (see Figure 18, on the left), in this scenario both TopAprioriMR and MaxAprioriMR equally behave. Analysing the results illustrated in Figure 18, it is obtained that differences between both algorithms are higher with the increment of the single-items. Thus, for a total of 18 single-items (see Figure 18, on the right) it is obtained that TopAprioriMR outperforms MaxAprioriMR, demonstrating that algorithms for mining maximal frequent patterns are time-consuming.

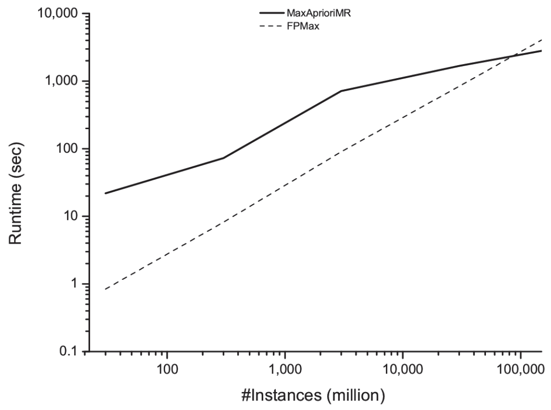


Fig. 17: Runtime of different algorithms when they are run on datasets comprising 16 singletons and up to 30,000 million of instances. The computational complexity is up to $O(2^{18} - 1 \times 3 \cdot 10^{10} \times 18)$.

[2]This dataset is electronically available at the website: http://fimi.ua.ac.be/data/

4.5 Accuracy in the space of solutions

In pattern mining, the most widely used metric to determine the interest of the patterns is the support or frequency of occurrence. Based on this quality measure, different pruning strategies have been proposed [13], the anti-monotone property being the most well known. In any pattern mining algorithm (considering or not pruning strategies), it is important to maintain the accuracy in the set of solutions —avoiding to lose useful patterns. In this regard, it is important to analyse whether the proposed algorithms are able to extract the accurate set of solutions. Table 4 illustrates the number of patterns discovered by each algorithm — the percentage of patterns with regard to the baseline is shown in brackets — for different search spaces and 30,000 million of instances. As shown, MapReduce versions (AprioriMR and IAprioriMR) for mining patterns with no pruning strategy obtain the 100% of the patterns, achieving a 100% in the accuracy of the space of solutions. The same behaviour is obtained with MaxAprioriMR, which obtains exactly the same set of maximal frequent patterns as FPMax, taking a pattern as frequent if it overcomes the minimum support threshold value 0.0001.

TABLE 4: Number of patterns discovered (percentage with regard to the baseline) for different search spaces and 30,000 million of instances. A minimum support threshold of 0.0001 was applied to algorithms that prune the search space.

Search space	AprioriMR	IAprioriMR	Apriori (baseline)
15	15 (100%)	15 (100%)	15
1,023	1,023 (100%)	1,023 (100%)	1,023
65,535	65,535 (100%)	65,535 (100%)	65,535
262,143	262,143 (100%)	262,143 (100%)	262,143

Search space	SPAprioriMR	TopAprioriMR	Apriori (baseline)
15	10 (100%)	10 (100%)	10
1,023	637 (100%)	637 (100%)	637
65,535	14,892 (100%)	14,892 (100%)	14,892
262,143	63,003 (100%)	62,832 (99.998%)	63,003

Search space	MaxApriorMR	FPMax (baseline)
15	6 (100%)	6
1,023	252 (100%)	252
65,535	8,008 (100%)	8,008
262,143	31,824 (100%)	31,824

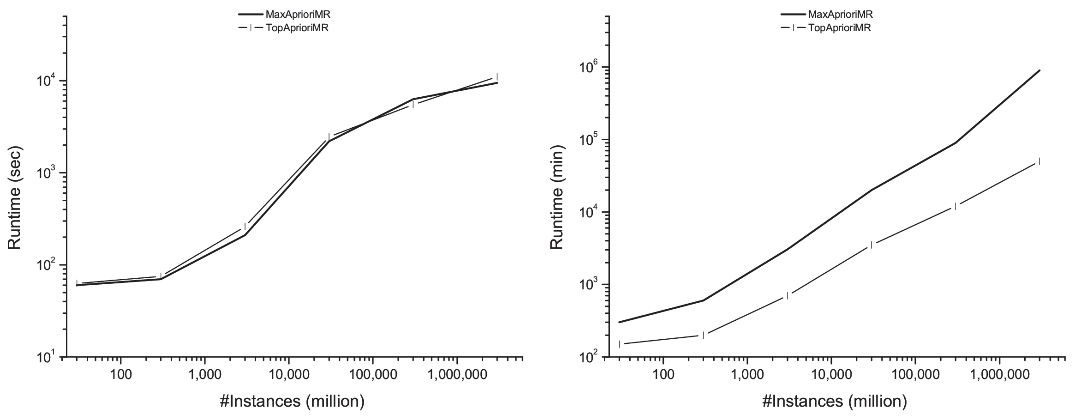


Fig. 18: Runtime of different algorithms when they are run on datasets comprising 12 singletons (left) and 18 singletons (right). The computational complexity is up to $O(2^{18} - 1 \times 3 \cdot 10^{18} \times 18)$

TABLE 5: Features of the three proposed algorithms for mining patterns by using the monotonic property.

Feature	AprioriMR	IAprioriMR	SPAprioriMR	TopAprioriMR	MaxAprioriMR
Support threshold			✓	✓	✓
Iterative ascending order		✓	✓	✓	
Iterative descending order					✓
Mining frequent patterns			✓	✓	
Mining maximal patterns					✓
Original dataset	✓	✓	✓		
Reduced dataset				✓	
Keep: infrequent patterns			✓	✓	
Keep: maximal patterns					✓

As for those algorithms considering any kind of pruning strategy (SPAprioriMR, TopAprioriMR and Apriori with antimonotone property), it is obtained that SPAprioriMR obtains exactly the same set of solutions regardless the search space considered. On the contrary, when considering TopAprioriMR, it may lose a small set of solutions due to its procedure of reducing the input dataset. However, results reveal that the percentage of patterns lost during the mining process is extremely low (close to the 0.002%).

To sum up, results reveal that all the proposed algorithms are able to produce the accurate set of solutions. Only TopAprioriMR may lose a tiny set of solutions, which may be insignificant when comparing the excellent performance achieved.

5 Concluding remarks

In this paper, we have proposed new efficient pattern mining algorithms to work in Big Data. All the proposed models are based on the well-known Apriori algorithm and the MapReduce framework. The proposed algorithms are divided into three main groups: (1) No pruning strategy. Two algorithms (AprioriMR and IAprioriMR) for mining any existing pattern in data have been proposed. (2) Pruning the search space by means of anti-monotone property. Two additional algorithms (SPAprioriMR and TopAprioriMR) have been proposed with the aim of discovering any frequent pattern available in data. (3) Maximal frequent patterns. A last algorithm (MaxAprioriMR) has been also proposed for mining condensed representations of frequent patterns. The features of all these algorithms are summarized in Table 5.

To test the performance of the proposed algorithms, we have conducted a series of experiments on a collection of Big Datasets comprising up to $3{\cdot}10^{18}$ instances and more than 5 million of single-items. All the algorithms have been compared to highly efficient algorithms in the pattern mining field. The experimental stage has revealed that our proposals perform really well for huge search spaces. Results have also revealed the unsuitability of MapReduce frameworks when small datasets are considered.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund, project TIN-2014-55252-P.

References

[1] T. Choi, H. K. Chan, and X. Yue, "Recent development in big data analytics for business operations and risk management," *IEEE Transactions on Cybernetics*, vol. 47, no. 1, pp. 81–92, 2017. [Online]. Available: http://dx.doi.org/10.1109/TCYB.2015.2507599

[2] J. M. Luna, "Pattern mining: current status and emerging topics," *Progress in Artificial Intelligence*, vol. 5, no. 3, pp. 165–170, 2016.

[3] C. C. Aggarwal and J. Han, *Frequent pattern mining*, 1st ed. Springer International Publishing, 2014.

[4] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[5] J. M. Luna, J. R. Romero, C. Romero, and S. Ventura, "On the use of genetic programming for mining comprehensible rules in subgroup discovery," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2329–2341, 2014. [Online]. Available: http://dx.doi.org/10.1109/TCYB.2014.2306819

[6] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[7] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: a frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[8] S. Zhang, Z. Du, and J. Wang, "New techniques for mining frequent patterns in unordered trees," *IEEE Transactions on Cybernetics*, vol. 45, no. 6, pp. 1113–1125, 2015.

[9] J. M. Luna, J. R. Romero, and S. Ventura, "Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules," *Knowledge and Information Systems*, vol. 32, no. 1, pp. 53–76, 2012.

[10] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '93, Washington, D.C., USA, 1993, pp. 207–216.

[11] J. Liu, K. Wang, and B. C. M. Fung, "Mining high utility patterns in one phase without generating candidates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1245–1257, 2016. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2015.2510012

[12] S. Zhang, Z. Du, and J. T. Wang, "New techniques for mining frequent patterns in unordered trees," *IEEE Transactions on Cybernetics*, vol. 45, no. 6, pp. 1113–1125, 2015. [Online]. Available: http://dx.doi.org/10.1109/TCYB.2014.2345579

[13] S. Ventura and J. M. Luna, *Pattern mining with evolutionary algorithms*, 1st ed. Springer International Publishing, 2016.

[14] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for nig data," in *Proceedings of the 2013 IEEE International Conference on Big Data*, ser. IEEEBigData 2013, Santa Clara, CA, USA, 2013, pp. 111–118.

[15] J. M. Luna, A. Cano, M. Pechenizkiy, and S. Ventura, "Speeding-up association rule mining with inverted index compression," *IEEE Transactions on Cybernetics*, vol. 46, no. 12, pp. 3059–3072, 2016. [Online]. Available: http://dx.doi.org/10.1109/TCYB.2015.2496175

[16] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.

[17] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51, no. 1, pp. 107–113, 2008.

[18] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, "Mrpr: A mapreduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, no. Part A, pp. 331–345, 2015.

[19] C. Lam, *Hadoop in action*, 1st ed. Manning Publications Co., 2010.

[20] B. Ziani and Y. Ouinten, "Mining maximal frequent itemsets: a java implementation of fpmax algorithm," in *Proceedings of the 6th International Conference on Innovations in Information Technology*, ser. IIT'09, Piscataway, NJ, USA, 2009, pp. 11–15.

[21] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000.

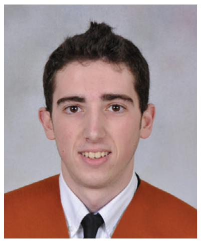
[22] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424–1440, 2004. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2004.77

[23] P. N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 1st ed. Addison Wesley, 2005.

[24] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08, Toronto, Canada, 2008, pp. 260–269.

[25] C. Borgelt, "Efficient implementations of apriori and eclat," in *Proceedings of the 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations*, ser. FIMI 2003, Melbourne, FL, USA, 2003, pp. 90–99.

José María Luna (M'10) received the Ph.D. degree in Computer Science from the University of Granada (Spain) in 2014. His research was firstly granted by the Ministry of Education with FPU AP2010-0041 and, then, he was awarded with a JdC - training PostDoc grant. He is author of the book "Pattern Mining with Evolutionary Algorithms", published by Springer. He has published more than 30 papers in top ranked journals and international scientific conferences, and he is author of two book chapters. He has also been engaged in 4 national and regional research projects. Dr. Luna has contributed to 3 international projects. His research is performed as a member of the Knowledge Discovery and Intelligent Systems Laboratory and is focused on evolutionary computation, pattern mining, association rule mining and its applications. José María Luna is a Member of the IEEE Computer, Computational Intelligence and Systems, Man and Cybernetics societies.



Fancisco Padillo received the M.Sc. degree in Computer Science from the University of Córdoba (Spain) in 2015. His research is performed as a member of the Knowledge Discovery and Intelligent Systems Laboratory and is focused on pattern mining, Big Data and the MapReduce paradigm.

Mykola Pechenizkiy (M'07) received the Ph.D. degree in computer science and information systems from the University of Jyvaskyla (JYU), Finland, in 2005. He is Full Professor in Predictive Analytics at the Department of Computer Science, Eindhoven University of Technology, The Netherlands. He is also Adjunct Professor in Data Mining for Industrial Applications in JYU. He has co-authored over 70 peer-reviewed publications and has been co-organizing several workshops, conferences (IEEE CBMS in 2008 and 2012, EDM in 2011 and 2015) and tutorials (at ECML/PKDD in 2010 and 2012, PAKDD in 2011, and IEEE CBMS in 2010). He has co-edited special issues in SIGKDD Explorations, Elsevier's DKE and AIIM, Springer's Evolving Systems and the Journal of Learning Analytics, and the first Handbook of Educational Data Mining. He serves as the President of the International Educational Data Mining Society.

Sebastián Ventura (M'07-SM'09) is currently a Full Professor at the Department of Computer Science and Numerical Analysis at the University of Cordoba, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He received his B.Sc. and Ph.D. degrees in sciences from the University of Cordoba, Spain, in 1989 and 1996, respectively. He has published more than 150 papers in journals and scientific conferences, and he has edited three books and several special issues in international journals. He has also been engaged in 12 research projects (being the coordinator of four of them) supported by the Spanish and Andalusian governments and the European Union. His main research interests are in the fields of soft-computing, machine learning, data mining, and their applications. Dr. Ventura is a senior member of the IEEE Computer, the IEEE Computational Intelligence and the IEEE Systems, Man and Cybernetics Societies, as well as the Association of Computing Machinery (ACM).