Fast Parallel Mining of Maximally Informative k-Itemsets in Big Data

Saber Salah, Reza Akbarinia, Florent Masseglia

Fast Parallel Mining of Maximally Informative k-Itemsets in Big Data

Saber Salah, Reza Akbarinia and Florent Masseglia
Inria and LIRMM, Zenith team, University of Montpellier, France
Email: first.last@inria.fr

***Abstract*—The discovery of informative itemsets is a fundamental building block in data analytics and information retrieval. While the problem has been widely studied, only few solutions scale. This is particularly the case when i) the data set is massive, calling for large-scale distribution, and/or ii) the length k of the informative itemset to be discovered is high. In this paper, we address the problem of parallel mining of maximally informative k-itemsets (*miki*) based on joint entropy. We propose PHIKS (Parallel Highly Informative K-ItemSet) a highly scalable, parallel *miki* mining algorithm. PHIKS renders the mining process of large scale databases (up to terabytes of data) succinct and effective. Its mining process is made up of only two efficient parallel jobs. With PHIKS, we provide a set of significant optimizations for calculating the joint entropies of *miki* having different sizes, which drastically reduces the execution time of the mining process. PHIKS has been extensively evaluated using massive real-world data sets. Our experimental results confirm the effectiveness of our proposal by the significant scale-up obtained with high itemsets length and over very large databases.**

I. Introduction

Featureset, or itemset, mining [1] is one of the fundamental building bricks for exploring informative patterns in databases. Features might be, for instance, the words occurring in a document, the score given by a user to a movie on a social network, or the characteristics of plants (growth, genotype, humidity, biomass, etc.) in a scientific study in agronomics. A large number of contributions in the literature has been proposed for itemset mining, exploring various measures according to the chosen relevance criteria. The most studied measure is probably the number of co-occurrences of a set of features, also known as frequent itemsets [2]. However, frequency does not give relevant results for a various range of applications, including information retrieval [3], since it does not give a complete overview of the hidden correlations between the itemsets in the database. This is particularly the case when the database is sparse [4]. Using other criteria to assess the informativeness of an itemset could result in discovering interesting new patterns that were not previously known. To this end, information theory [5] gives us strong supports for measuring the informativeness of itemsets. One of the most popular measures is the joint entropy of an itemset. An itemset X that has higher joint entropy brings up more information about the objects in the database.

Saber Salah - This work was partially supported by the Inria Project Lab Hemera.

Features	Documents									
	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
A	1	1	1	1	1	0	0	0	0	0
B	0	1	0	0	1	1	0	1	0	1
C	1	0	0	1	0	1	1	0	1	0
D	1	0	1	1	1	1	1	1	1	1
E	1	1	1	1	1	1	1	1	1	1

TABLE I: Features in the documents

We study the problem of Maximally Informative k-Itemsets (*miki* for short) discovery in massive data sets, where informativeness is expressed by means of joint entropy and k is the size of the itemset [6], [7], [8]. *Miki* are itemsets of interest that better explain the correlations and relationships in the data. Example 1 gives an illustration of *miki* and its potential for real world applications such as information retrieval.

Example 1: In this application, we would like to retrieve documents from Table I, in which the columns d_1, d_{10} are documents, and the attributes A, B, C, D, E are some features (items, keywords) in the documents. The value "1" means that the feature occurs in the document, and "0" not. It is easy to observe that the itemset (D, E) is frequent, because features D and E occur together in almost every document. However, it provides little help for document retrieval. In other words, given a document d_x in our data set, one might look for the occurrence of the itemset (D, E) and, depending on whether it occurs or not, she will not be able to decide which document it is. By contrast, the itemset (A, B, C) is infrequent, as its member features rarely or never appear together in the data. And it is troublesome to summarize the value patterns of the itemset (A, B, C). Providing it with the values $< 1, 0, 0 >$ we could find the corresponding document O_3; similarly, given the values $< 0, 1, 1 >$ we will have the corresponding document O_6. Although (A, B, C) is infrequent, it contains lots of useful information which is hard to summarize. By looking at the values of each feature in the itemset (A, B, C), it is much easier to decide exactly which document they belong to. (A, B, C) is a maximally informative itemset of size $k = 3$.

Miki mining is a key problem in data analytics with high potential impact on various tasks such as supervised learning [9], unsupervised learning [10] or information retrieval [3], to cite a few. A typical application is the discovery of discriminative sets of features, based on joint entropy, which allows distinguishing between different categories of objects.

1550-4786/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDM.2015.86

Unfortunately, it is very difficult to maintain good results, in terms of both response time and quality, when the number of objects becomes very large. Indeed, with massive amounts of data, computing the joint entropies of all itemsets in parallel is a very challenging task for many reasons. First, the data is no longer located in one computer, instead, it is distributed over several machines. Second, the number of iterations of parallel jobs would be linear to k (i.e., the number of features in the itemset to be extracted [7]), which needs multiple database scans and in turn violates the parallel execution of the mining process. We believe that an efficient *miki* mining solution should scale up with the increase in the size of the itemsets, calling for cutting edge parallel algorithms and high performance evaluation of an itemset's joint entropy in massively distributed environments.

We propose a deep combination of both information theory and massive distribution by taking advantage of parallel programming frameworks such as MapReduce [11] or Spark [12]. To the best of our knowledge, there has been no prior work on parallel informative itemsets discovery based on joint entropy. We designed and developed an efficient parallel algorithm, namely Parallel Highly Informative K-itemSet (PHIKS in short), that renders the discovery of *miki* from a very large database (up to Terabytes of data) simple and effective. It performs the mining of *miki* in two parallel jobs. PHIKS cleverly exploits available data at each mapper to efficiently calculate the joint entropies of *miki* candidates. For more efficiency, we provide PHIKS with optimizations that allow for very significant improvements of the whole process of *miki* mining. The first technique estimates the upper bound of a given set of candidates and allows for a dramatic reduction of data communications, by filtering unpromising itemsets without having to perform any additional scan over the data. The second technique reduces significantly the number of scans over the input database of each mapper, i.e., only one scan per step, by incrementally computing the joint entropy of candidate features. This reduces drastically the work that should be done by the mappers, and thereby the total execution time.

PHIKS has been extensively evaluated using massive real-world data sets. Our experimental results show that PHIKS significantly outperforms alternative approaches, and confirm the effectiveness of our proposal over large databases containing for example one Terabyte of data.

The rest of the paper is structured as follows. Section II gives formal definitions of informative itemsets, basic used notations, and the necessary background is given in Section III. In Section IV, we propose our PHIKS algorithm, and depict its whole core mining process. Section V reports on our experimental validation over real-world data sets. Section VI discusses related work, and Section VII concludes.

II. Definitions

The following definitions introduce the basic requirements for mining maximally informative k-itemsets [7].

Definition 1: Let $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ be a set of literals called $features$. An $itemset$ X is a set of $features$ from $\mathcal{F}$, i.e., $X \subseteq \mathcal{F}$. The $size$ or $length$ of the $itemset$ X is the number of $features$ in it. A $transaction$ T is a set of elements such that $T \subseteq \mathcal{F}$ and $T \neq \emptyset$. A $database$ $\mathcal{D}$ is a set of $transactions$.

Definition 2: The entropy of a feature i in a database $\mathcal{D}$ measures the expected amount of information needed to specify the state of uncertainty or disorder for the feature i in $\mathcal{D}$. Let i be a feature in $\mathcal{D}$, and $P(i = n)$ be the probability that i has value n in $\mathcal{D}$ (we consider categorical data, where the value will be '1' if the object has the feature and '0' otherwise). The entropy of the feature i is given by $H(i) = -(P(i = 0)log(P(i = 0)) + P(i = 1)log(P(i = 1)))$, where the logarithm base is 2.

Definition 3: The *binary projection*, or *projection* of an itemset X in a transaction T ($proj(X, T)$) is the set of size $|X|$ where each item (i.e., feature) of X is replaced by '1' if it occurs in T and by '0' otherwise. The *projection counting* of X in a database $\mathcal{D}$ is the set of projections of X in each transaction of $\mathcal{D}$, where each projection is associated with its number of occurrences in $\mathcal{D}$.

Example 2: Let us consider Table I. The projection of (B, C, D) in d_1 is $(0, 1, 1)$. The projections of (D, E) on the database of Table I are $(1, 1)$ with nine occurrences and $(0, 1)$ with one occurrence.

Definition 4: Given an itemset $X = \{x_1, x_2, \ldots, x_k\}$ and a tuple of binary values $\mathcal{B} = \{b_1, b_2, \ldots, b_k\} \in \{0\,1\}^k$. The joint entropy of X is defined as: $H(X) = -\sum_{\mathcal{B} \in \{0,1\}^{|k|}} J \times log(J)$. Where J= $P(x_1 = b_1, \ldots, x_k = b_k)$ is the joint probability of $X = \{x_1, x_2, \ldots, x_k\}$.

Given a database $\mathcal{D}$, the joint entropy $H(X)$ of an itemset X in $\mathcal{D}$ is proportional to its size k i.e., the increase in the size of X implies an increase in its joint entropy $H(X)$. The higher the value of $H(X)$, the more information the itemset X provides in $\mathcal{D}$. For simplicity, we use the term entropy of an itemset X to denote its joint entropy.

Example 3: Let us consider the database of Table I. The joint entropy of (D, E) is given by $H(D, E) = -\frac{9}{10}log(\frac{9}{10}) - \frac{1}{10}log(\frac{1}{10}) = 0.468$. Where the quantities $\frac{9}{10}$ and $\frac{1}{10}$ respectively represent the joint probabilities of the projection values $(1, 1)$ and $(0, 1)$ in the database.

Definition 5: Given a set $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ of features, an itemset $X \subseteq \mathcal{F}$ of length k is a maximally informative k-itemset, if for all itemsets $Y \subseteq \mathcal{F}$ of size k, $H(Y) \leq H(X)$. Hence, a maximally informative k-itemset is the itemset of size k with the highest joint entropy value.

The problem of mining maximally informative k-itemsets presents a variant of itemset mining, it relies on the joint entropy measure for assessing the informativeness brought by an itemset.

Definition 6: Given a database $\mathcal{D}$ which consists of a set of n attributes (features) $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$. Given a number k, the problem of *miki* mining is to return a subset $F' \subseteq F$ with size k, i.e., $|F'| = k$, having the highest joint entropy in $\mathcal{D}$, i.e., $\forall F'' \subseteq F \wedge |F''| = k \Rightarrow H(F'') \leq H(F')$.

III. Background

A. Miki Discovery in a Centralized Environment

In [7], an effective approach is proposed for *miki* discovery in a centralized environment. Their *ForwardSelection* heuristic uses a "generating-pruning" approach, which is similar to the principle of *Apriori* [2]. i_1, the feature having the highest entropy is selected as a seed. Then, i_1 is combined with all the remaining features, in order to build candidates. In other words, there will be $|\mathcal{F}-1|$ candidates (*i.e.*, $(i_1,i_2),(i_1,i_3),\ldots,(i_1,i_{|\mathcal{F}-1|})$). The entropy of each candidate is given by a scan over the database, and the candidate having the highest entropy, say (i_1,i_2), is kept. A set of $|\mathcal{F}-2|$ candidates of size 3 is generated (*i.e.*, $(i_1,i_2,i_3),(i_1,i_2,i_4),\ldots,(i_1,i_2,i_{|\mathcal{F}-2|})$) and their entropy is given by a new scan over the database. This process is repeated until the size of the extracted itemset is k.

B. MapReduce and Job Execution

MapReduce has gained increasing popularity, as shown by the tremendous success of Hadoop [13], an open-source implementation. Initially proposed in [11], it was popularized by Hadoop [14], an open-source implementation. The idea behind MapReduce is simple and elegant. Given an input file, and two functions map and reduce, each MapReduce job is executed in two main phases: map and reduce. One map task is created per input split. Each map task reads its corresponding input split, applies the map function on each input pair and generates intermediate key-value pairs. Once the map task is completed, the master is notified about the location of the generated intermediate key-values. In the reduce phase, each intermediate key is assigned to one of the reduce workers. Each reduce worker retrieves the values corresponding to its assigned keys from all the map workers, and merges them using an external merge-sort. Then, it groups pairs with the same key and calls the reduce function on the corresponding values. This function will generate the final output results.

IV. PHIKS Algorithm

In a massively distributed environment, a possible naive approach for *miki* mining would be a straightforward implementation of *ForwardSelection* [7] (see Section III-A). However, given the "generating-pruning" principle of this heuristic, it is not suited for environments like Spark [12] or MapReduce [11] and would lead to very bad performances. The main reason is that each scan over the data set is done through a distributed job (i.e., there will be k jobs, one for each generation of candidates that must be tested over the database). Our experiments, in Section V, give an illustration of the catastrophic response times of *ForwardSelection* in a straightforward implementation on MapReduce (the worst, for all of our settings). This is not surprising since most algorithms designed for a centralized itemset mining do not perform well in massively distributed environments in a direct implementation [15], [16], [17], and *miki* don't escape that rule.

Such an inadequacy calls for new distributed algorithmic principles. To the best of our knowledge, there is no previous work on distributed mining of *miki*. However, we may build on top of cutting edge studies in frequent itemset mining, while considering the very demanding characteristics of *miki*.

Interestingly, in the case of frequent itemsets in MapReduce, a mere algorithm consisting of two jobs outperforms most existing solutions [18] by using the principle of SON [19], a divide and conquer algorithm. Unfortunately, despite its similarities with frequent itemset mining, the discovery of *miki* is much more challenging. Indeed, the number of occurrences of an itemset X in a database $\mathcal{D}$ is additive and can be easily distributed (the global number of occurrences of X is simply the sum of its local numbers of occurrences on subsets of $\mathcal{D}$). Entropy is much more combinatorial since it is based on the the projection counting of X in $\mathcal{D}$ and calls for efficient algorithmic advances, deeply combined with the principles of distributed environments.

A. Distributed Projection Counting

Before presenting the details of our contribution, we need to provide tools for computing the projection of an itemset X on a database $\mathcal{D}$, when $\mathcal{D}$ is divided into subsets on different splits, in a distributed environment, and entropy has to be encoded in the key-value format. We have to count, for each projection p of X, its number of occurrences on $\mathcal{D}$. This can be solved with an association of the itemset as a key and the projection as a value. On a split, for each projection of an itemset X, X is sent to the reducer as the key coupled with its projection. The reducer then counts the number of occurrences, on all the splits, of each (key:value) couple and is therefore able to calculate the entropy of each itemset. Communications may be optimized by avoiding to emit a $key : val$ couple when the projection does not appear in the transaction and is only made of '0' (on the reducer, the number of times that a projection p of X does not appear in $\mathcal{D}$ is determined by subtracting the number projections of X in D from $|\mathcal{D}|$).

Example 4: Let us consider $\mathcal{D}$, the database of Table I, and the itemset $X=(D,E)$. Let us consider that $\mathcal{D}$ is divided into two splits $S_1=\{d_1..d5\}$ and $S_2=\{d_6..d_{10}\}$. With one simple MapReduce job, it is possible to calculate the entropy of X. The algorithm of a mapper would be the following: for each document d, emit a couple $(key : val)$ where $key = X$ and $val = proj(X,d)$. The first mapper (corresponding to S_1) will emit the following couples: $((D,E) : (1,1))$ 4 times and $((D,E) : (0,1))$ once. The second mapper will emit $((D,E) : (1,1))$ 5 times. The reducers will do the sum and the final result will be $((D,E) : (1,1))$ occurs 9 times and $(((D,E) : (0,1))$ once.

B. Discovering miki in Two Rounds

Our heuristic will use at most two MapReduce jobs in order to discover the k-itemset having the highest entropy. The goal of the first job is to extract locally, on the distributed subsets of $\mathcal{D}$, a set of candidate itemsets that are likely to have a high global entropy. To that end, we apply the principle of

Split	A	B	C	D	E
S_1	0	0	1	0	0
	0	1	0	0	0
	1	0	1	0	0
	1	1	0	0	0
	0	1	1	0	1
	1	0	0	0	0
S_2	0	0	0	0	1
	0	1	0	1	1
	1	0	0	0	1
	1	1	0	1	1
	0	1	0	0	0
	1	0	0	1	1

TABLE II: Local Vs. global entropy

ForwardSelection locally, on each mapper, and grow an itemset by adding a new feature at each step. After the last scan, for each candidate itemset X of size k we have the projection counting of X on the local data set. A straightforward approach would be to emit the candidate itemset having the highest local entropy. Then the reducers would collect the local *miki* and we would check their global entropy on $\mathcal{D}$ by means of a second MapReduce job. Unfortunately, this approach would not be correct, since an itemset might have the highest global entropy, while actually not having the highest entropy in each subset. Example 5 gives a possible case where a global *miki* does not appear as a local *miki* on any subset of the database.

Example 5: Let us consider $\mathcal{D}$, the database given by Table II, which is divided into two splits of six transactions. The global *miki* of size 2 in this database is (A, B, E). More precisely, the entropy of (A, B, E) on $\mathcal{D}$ is given by $-\frac{1}{12} \times log(\frac{1}{12}) \times 4 - \frac{2}{12} \times log(\frac{2}{12}) \times 4 = 2.92$. However, if we consider each split individually, (A, B, E) always has a lower entropy compared to at least one different itemset. For instance, on the split S_1, the projections of (A, B, E) are $(0, 0, 0)$, $(0, 1, 0)$, $(1, 1, 0)$ and $(0, 1, 1)$ with one occurrence each, and $(1, 0, 0)$ with two occurrences. Therefore the entropy of (A, B, E) on S_1 is 2.25 (*i.e.*, $-\frac{1}{6} \times log(\frac{1}{6}) \times 4 - \frac{2}{6} \times log(\frac{2}{6}) = 2.25$). On the other hand, the projections of (A, B, C) on S_1 are $(0, 0, 1)$, $(0, 1, 0)$, $(1, 0, 1)$, $(1, 1, 0)$, $(0, 1, 1)$ and $(1, 0, 0)$ with one occurrence each, and the entropy of (A, B, C) on S_1 is 2.58 (*i.e.*, $-\frac{1}{6} \times log(\frac{1}{6}) \times 6 = 2.58$). This is similar on S_2 where the entropy of (A, B, E) is 2.25 and the entropy of (A, B, D) is 2.58. However, (A, B, C) and (A, B, D) both have a global entropy of 2.62 on $\mathcal{D}$, which is lower than 2.92, the global entropy of (A, B, E) on $\mathcal{D}$.

Since it is possible that a global *miki* is never found as a local *miki*, we need to consider a larger number of candidate itemsets. This can be done by exploiting the set of candidates that are built in the very last step of *ForwardSelection*. This step aims to calculate the projection counting of $\mathcal{F} - k$ candidates and then compute their local entropy. Instead of only emitting the itemset having the larger entropy, we will emit, for each candidate X, the projection counting of X on the split, as explained in Section IV-A. The reducers will then be provided with, for each local candidate X_i ($1 \leq i \leq m$, where m is the number of mappers, or splits), the projection counting of X on a subset of $\mathcal{D}$. The main idea is that the itemset having the highest entropy is highly likely to be in that set of candidates. For instance, in the database given by Table II and $k = 3$, the global *miki* is (A, B, E), while the local *miki* are (A, B, C) on S_1 and (A, B, D) on S_2. However, with the technique described above, the itemset (A, B, E) will be a local candidate, and will be sent to the reducers with the whole set of projections encountered so far in the splits. The reducer will then calculate its global entropy, compare it to the entropy of the other itemsets, and (A, B, E) will eventually be selected as the *miki* on this database.

Unfortunately, it is possible that X has not been generated as a candidate itemset on the entire set of splits (consider a biased data distribution, where a split contains some features with high entropies, and these features have low entropies on the other splits). Therefore, we have two possible cases at this step:

1) X is a candidate itemset on all the splits and we are able to calculate its exact projection counting on $\mathcal{D}$, by means of the technique given in Section IV-A.
2) There is (at least) one split where X has not been generated as a candidate and we are not able to calculate its exact projection counting on $\mathcal{D}$.

The first case does not need more discussion, since we have collected all the necessary information for calculating the entropy of X on $\mathcal{D}$. The second case is more difficult since X might be the *miki* but we cannot be sure, due to lack of information about its local entropy on (at least) one split. Therefore, we need to check the entropy of X on $\mathcal{D}$ with a second MapReduce job intended to calculate its exact projection counting. The goal of this second round is to check that no local candidate has been ignored at the global scale. At the end of this round, we have the entropy of all the promising candidate itemsets and we are able to pick the one with the highest entropy. This is the architecture of our approach, the raw version of which (without optimization) is called *Simple-PHIKS*. So far, we have designed a distributed architecture and a *miki* extraction algorithm that, in our experiments reported in Section V outperforms *ForwardSelection* by several orders of magnitude. However, by exploiting and improving some concepts of information theory, we may significantly optimize this algorithm and further accelerate its execution at different parts of the architecture, as explained in the following sections.

C. Candidate Reduction Using Entropy Upper Bound

One of the shortcomings of the basic version of our two rounds approach is that the number of candidate itemsets, which should be processed in the second job, may be high for large databases as it will be illustrated by our experiments in Section V. This is particularly the case when the features are not uniformly distributed in the splits of mappers. These candidate itemsets are sent partially by the mappers (i.e., not by all of them), thus we cannot compute their total entropy in

the corresponding reducer. This is why, in the basic version of our approach, we compute their entropy in the second job by reading again the database.

Here, we propose an efficient technique for significantly reducing the number of candidates. The main idea is to compute an upper bound for the entropy of the partially sent itemsets, and discard them if they have no chance to be a global *miki*. For this, we exploit the available information about the *miki* candidates sent by the mappers to the corresponding reducer.

Let us describe formally our approach. Let X be a partially sent itemset, and m be a mapper that has not sent X and its projection frequencies to the reducer R that is responsible for computing the entropy of X. In the reducer R, the frequency of X projections for a part of the database is missing, i.e., in the split of m. We call these frequencies as *missing* frequencies. We compute an upper bound for the entropy of X by estimating its missing frequencies. This is done in two steps. Firstly, finding the biggest subset of X, say Y, for which all frequencies are available and secondly, distributing the frequencies of Y among the projections of X in such a way that the entropy of X be the maximum.

1) Step 1: the idea behind the first step is that the frequencies of the projections of an itemset X can be derived from the projections of its subsets. For example, suppose two itemsets $X = \{A, B, C, D\}$ and $Y = \{A, B\}$, then the frequency of the projection $p = (1, 1)$ of Y is equal to the sum of the following projections in X: $p_1 = (1, 1, 0, 0)$, $p_2 = (1, 1, 0, 1)$, $p_3 = (1, 1, 1, 0)$ and $p_4 = (1, 1, 1, 1)$. The reason is that in all these four projections, the features A and B exist, thus the number of times that p occurs in the database is equal to the total number of times that the four projections p_1 to p_4 occur. This is stated by the following lemma.

Lemma 4.1: Let the itemset Y be a subset of the itemset X, i.e., $Y \subseteq X$. Then, the frequency of any projection p of Y is equal to the sum of the frequencies of all projections of X which involve p.

Proof. The proof can be easily done as in the above discussion.

In Step 1, among the *available subsets* of itemset X, i.e., those for which we have all projection frequencies, we choose the one that has the highest size. The reason is that its intersection with X is the highest, thus our estimated upper bound about the entropy of X will be closer to the real one.

2) Step 2: let Y be the biggest available subset of X in reducer R. After choosing Y, we distribute the frequency of each projection p of Y among the projections of X that are derived from p. There may be many ways to distribute the frequencies. For instance, in the example of Step 1, if the frequency of p is 6, then the number of combinations for distributing 6 among the four projections p_1 to p_4 is equal to the solutions which can be found for the following equation: $x_1 + x_2 + x_3 + x_4 = 6$ when $x_i \geq 0$. In general, the number of ways for distributing a frequency f among n projections is equal to the number of solutions for the following equation:

$$x_1 + x_2 + ... + x_n = f \ for \ x_i \geq 0 \qquad (1)$$

Obviously, when f is higher than n, there is a lot of solutions for this equation. Among all these solutions, we choose a solution that maximizes the entropy of X. The following lemma shows how to choose such a solution.

Lemma 4.2: Let $\mathcal{D}$ be a database, and X be an itemset. Then, the entropy of X over $\mathcal{D}$ is the maximum if the possible projections of X over $\mathcal{D}$ have the same frequency.

Proof. The proof is done by implying the fact that in the entropy definition (see Definition 2), the maximum entropy is for the case where all possible combinations have the same probability. Since, the probability is proportional to the frequency, then the maximum entropy is obtained in the case where the frequencies are the same. □

The above lemma proposes that for finding an upper bound for the entropy of X (i.e., finding its maximal possible entropy), we should distribute equally (or almost equally) the frequency of each projection in Y among the derived projections in X. Let f be the frequency of a projection in Y and n be the number of its derived projections, if (f modulo n) = 0 then we distribute equally the frequency, otherwise we first distribute the quotient among the projections, and then the rest randomly.

After computing the upper bound for entropy of X, we compare it with the maximum entropy of the itemsets for which we have received all projections (so we know their real entropy), and discard X if its upper bound is less than the maximum found entropy until now.

D. Prefix/Suffix

When calculating the local *miki* on a mapper, at each step we consider a set of candidates having size j that share a prefix of size $j - 1$. For instance, with the database of Table II and the subset of split S_1, the corresponding mapper will extract (A, B) as the *miki* of size 2. Then, it will build 3 candidates: (A, B, C), (A, B, D) and (A, B, E). A straightforward approach for calculating the joint entropy of these candidates would be to calculate their projection counting by means of an exhaustive scan over the data of S_1 (*i.e.*, read the first transaction of S_1, compare it to each candidate in order to find their projections, and move to the next transaction). However, these candidates share a prefix of size 2: (A, B). Therefore, we store the candidates in a structure that contains the prefix itemset, of size $j - 1$, and the set of $|\mathcal{F} - j|$ suffix features. Then, for a transaction T, we only need to i) calculate $proj(p, T)$ where p is the prefix and ii) for each suffix feature f, find the projection of f on T, append $proj(f, T)$ to $proj(p, T)$ and emit the result. Let us illustrate this principle with the example above (*i.e.*, first transaction of S_1 in Table II). The structure is as follows: {prefix=(A, B):suffixes=C, D, E}. With this structure, instead of comparing (A, B, C), (A, B, D) and (A, B, E) to the transaction and find their respective projections, we calculate the projection of (A, B), their prefix, *i.e.*, $(0, 0)$, and the projection of each suffix, *i.e.*, (1), (0) and (0) for C, D, and E respectively. Each suffix projection is then added to the prefix projection and emitted. In our case, we build three projections: $(0, 0, 1)$, $(0, 0, 0)$ and $(0, 0, 0)$, and the

mapper will emit $((A, B, C) : (0, 0, 1))$, $((A, B, D) : (0, 0, 0))$ and $((A, B, E) : (0, 0, 0))$.

E. Incremental Entropy Computation in Mappers

In the basic version of our two rounds approach, each mapper performs many scans over its split to compute the entropy of candidates and finally find the local miki. Given k as the size of the requested itemset, in each step j of the k steps in the local miki algorithm, the mapper uses the itemset of size $j-1$ discovered so far, and builds $|F|-j$ candidate itemsets before selecting the one having the highest entropy. For calculating each joint entropy, a scan of the input split is needed in order to compute the frequency (and thus the probability) of projections. Let $|F|$ be the number of features in the database, then the number of scans done by each mapper is $O(k*|F|)$. Although the input split is kept in memory, this high number of scans over the split is responsible for the main part of the time taken by the mappers.

In this section, we propose an efficient approach to significantly reduce the number of scans. Our approach that incrementally computes the joint entropies, needs to do in each step just one scan of the input split. Thus, the number of scans done by this approach is $O(k)$.

To incrementally compute the entropy, our approach takes advantage of the following lemma.

Lemma 4.3: Let X be an itemset, and suppose we make an itemset Y by adding a new feature i to X, i.e., $Y = X + \{i\}$. Then, for each projection p in X two projections $p_1 = p.0$, and $p_2 = p.1$ are generated in Y, and the sum of the frequency of p_1 and p_2 is equal to that of p, i.e., $f(p) = f(p_1) + f(p_2)$.
proof. The projections of Y can be divided into two groups: 1) those that represent transactions containing i; 2) those representing the transactions that do not involve i. For each projection p_1 in the first group, there is a projection p_2 in the second group, such that p_1 and p_2 differ only in one bit, i.e., the bit that represents the feature i. If we remove this bit from p_1 or p_2, then we obtain a projection in X, say p, that represents all transactions that are represented by p_1 or p_2. Thus, for each project p in X, there are two projections p_1 and p_2 in Y generated from p by adding one additional bit, and the frequency of p is equal to the sum of the frequencies of p_1 and p_2. □

Our incremental approach for *miki* computing proceeds as follows. Let X be the miki in step j . Initially, we set $X = \{\}$, with a null projection whose frequency is equal to n, i.e., the size of the database. Then, in each step j $(1 \leq j \leq k)$, we do as follows. For each remaining feature $i \in F - X$, we create a hash map $h_{i,j}$ containing all projections of the itemset $X + \{i\}$, and we initiate the frequency of each projection to zero. Then, we scan the set of transactions in the input split of the mapper. For each transaction t, we obtain a set S that is the intersection of t and $F - X$, i.e., $S = t \cap (F - X)$. For each feature $i \in S$, we obtain the projection of t over $X + \{i\}$, say p_2, and increment by one the frequency of the projection p_2 in the hash map $h_{i,j}$. After scanning all transactions of the split, we obtain the frequency of all projections ending with 1. For computing the projections ending with 0, we use Lemma 4.3 as follows. Let $p.0$ be a projection ending with 0, we find the projection $p.1$ (i.e., the projection that differs only in the last bit), and set the frequency of $p.0$ equal to the frequency of p minus that of $p.1$, i.e., $f(p.0) = f(p) - f(p.1)$. By this way, we compute the frequency of projections ending with 0.

After computing the frequencies, we can compute the entropy of itemset $X + \{i\}$, for each feature $i \in F - X$. At the end of each step, we add to X the feature i whose joint entropy with X is the highest. We keep the hash map of the selected itemset, and remove all other hash maps including that of the previous step. Then, we go to the next step until finishing step k. Notice that to obtain the frequency of p in step j, we use the hash map of the previous step, i.e., $H_{i,j-1}$, this is why, at each step we keep the hash map of the selected miki.

Let us now prove the correctness of our approach using the following Theorem.

Theorem 4.4: Given a database $\mathcal{D}$, and a value k as the size of requested miki. Then, our incremental approach computes correctly the entropy of the candidate itemsets in all steps.
proof. To prove the correctness of our approach, it is sufficient to show that in each step the projection frequencies of $X + \{i\}$ are computed correctly. We show this by induction on the number of steps, i.e., j for $1 \leq j \leq k$.

Base. In the first step, the itemset $X + \{i\} = \{i\}$ because initially $X = \{\}$. There are two projections for $\{i\} : p_1 = (0)$ and $p_2 = (1)$. The frequency of p_2 is equal to the number of transactions containing i. Thus during the scan of the split, we correctly set the frequency of p_2. Since there is no other projection for i, the frequency of p_1 is equal to $n - f(p_2)$, where n is the size of the database. This frequency is found correctly by our approach. Thus, for step $j = 1$ our approach finds correctly the projection frequencies of $X + \{i\}$.

Induction. we assume that our approach works correctly in step $j-1$, then we prove that it will work correctly in step j. The proof can be done easily by using Lemma 4.3. According this lemma, for each projection p in step $j-1$ there are two projections $p_1 = (p.0)$, and $p_2 = (p.1)$ in step j. The frequency of p_2 is computed correctly during the scan of the split. We assume that the frequency of p has been correctly computed in step $j-1$. Then, Lemma 4.3 implies that the frequency of p_1 has been also well computed since we have $f(p) = f(p_1) + f(p_2)$. □

F. Complete Approach

Our approach depicts the core mining process of Parallel Highly Informative K-itemSet Algorithm (PHIKS). The major steps of PHIKS algorithm for *miki* discovery are summarized in Algorithms 1 and 2. Algorithm 1 depicts the mining process of the first MapReduce job of PHIKS, while Algorithm 2 depicts the mining process of its second MapReduce job.

V. Experiments

To evaluate the performance of our PHIKS algorithm, we have carried out extensive experimental tests. In section V-A,

Algorithm 1: PHIKS: Job1

Input: n data splits $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of a database $\mathcal{D}$, K the size of *miki*
Output: A *miki* of Size K

//Mapper Class 1
map(*key: Line Offset*: $\mathcal{K}_1$, $value$ = Whole S_i: $\mathcal{V}_1$ **)**
- $\mathcal{F}_i \leftarrow$ the set of features in S_i
- $\forall f \in \mathcal{F}_i$ compute $H(f)$ on S_i, using prefix/suffix
- $n \leftarrow 1$ // current size of itemsets
- *HInFS* $\leftarrow max(H(f)), \forall f \in \mathcal{F}_i$
// *HInFS* is the itemset of size n
// having the highest entropy
while $i \neq k$ **do**
  - $n++$
  - $\mathcal{C}_n \leftarrow$ BuildCandidates(*HInFS*, $\mathcal{F}_i \backslash$*HInFS*)
  - $\forall c \in \mathcal{C}_p, H(c_i) \leftarrow$ ComputeJointEntropy(c, S_i)
  - *HInFS* $\leftarrow max(H(c)), \forall c \in \mathcal{C}_n$
// $\mathcal{C}_k$ contains all the candidate itemsets of size k
// and $\forall c \in \mathcal{C}_k$, the joint entropy of c is in $H(c_i)$
for $c \in \mathcal{C}_k$ **do**
  - $\mathcal{P}_c \leftarrow$ projections(c, S_i)
  for $p \in \mathcal{P}_c$ **do**
    - **emit**($key = c : value = p$)

//Reducer Class 1
reduce(*key: itemset* c,
$list(values)$*: projections(c)* **)**
if c *has been emitted by all the mappers* **then**
  // We have all the projections of c on $\mathcal{D}$
  // we store its entropy in a file "complete"
  - $H(c) \leftarrow$ IncrJointEntropy(c,projections(c))
  - **emit**(c,$H(c)$) in a file *Complete*
else
  // Missing nformation. We have to estimate
  // the upper bound of c's joint entropy
  // and store it in a file "Incomplete"
  - $Est \leftarrow$ UpperBound(c,projections(c))
  - **emit**(c, Est) in a file "Incomplete"
close( )
- $C_{max} \leftarrow$ CandidateWithMaxEntropy("Complete")
- **emit**(C_{max}, $H(C_{max})$)
in a file "CompleteMaxFromJob1"
for $c \in$ *"Incomplete"* **do**
  if $Est(c) > H(C_{max})$ **then**
    // c is potentially a *miki*, it has
    // to be checked over $\mathcal{D}$
    - **emit**(c,Null) in a file "ToBeTested"

Algorithm 2: PHIKS: Job2

Input: Database $\mathcal{D}$, K *miki* Size
Output: Tested *miki* of Size K

//Mapper Class 2
map(*key: Line Offset*: $\mathcal{K}_1$, $value$ = Transaction: $\mathcal{V}_1$ **)**
- Read file 'ToBeTested' from Job1 (once) in the mapper
- $\mathcal{F} \leftarrow$ set of itemsets in 'ToBeTested'
for $f \in \mathcal{F}$ **do**
  - $p \leftarrow$ projections(f, $\mathcal{V}_1$)
  emit (key*: f,* $value$*: p*)

//Reducer Class 2
reduce(*key: itemset* f,
$list(values)$*: projections(f)* **)**
// we have all the projections of f on $\mathcal{D}$ that come
// from all mappers
// we compute its joint entropy and we write the result to a file
// "CompleteFromJob2"
- $H(f) \leftarrow$ IncrJointEntropy(f,projections(f))
- write(f, $H(f)$) to a file "CompleteFromJob2" in HDFS
// optional, we emit the result of use later, from the close() method
- **emit** (key*: f,* $value$*:* $H(f)$)
close( )
// emit miki having highest joint entropy
- read file "CompleteMaxFromJob1"
- read file "CompleteFromJob2"
- Max $\leftarrow$ max("CompleteMaxFromJob1", "CompleteFromJob2")
- **emit**(*miki*,Max)

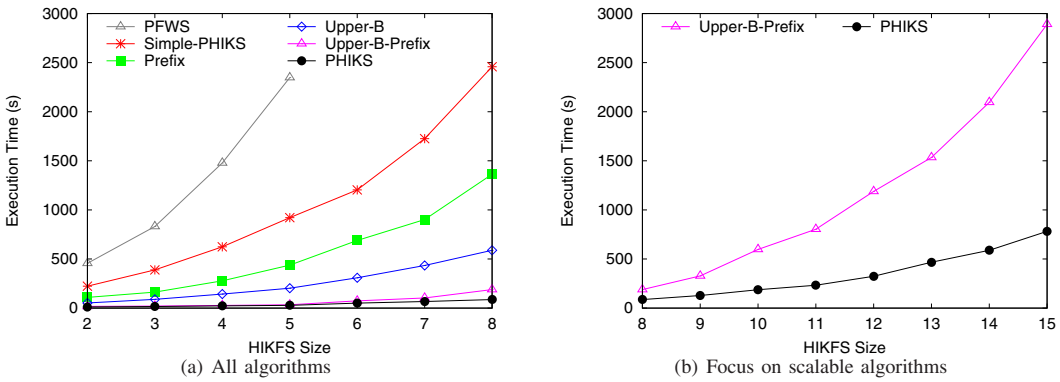
we depict our experimental setup, and in section V-B we investigate and discuss the results of our different experiments.

A. Experimental Setup

We implemented PHIKS algorithm on top of Hadoop-MapReduce using Java programming language version 1.7 and Hadoop [13] version 1.0.3.

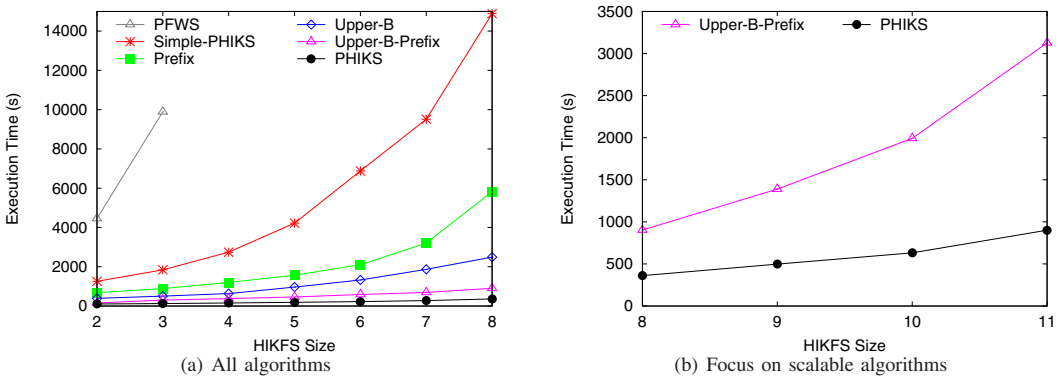
To better evaluate the performance of PHIKS algorithm, we used two real-world data sets. The first one is the 2014 English Wikipedia data set articles [20], having a total size of 49 Gigabytes and composed of 5 million articles. The second data set is a sample of ClueWeb English data set [21] with size of around one Terabyte and having 632 million articles. For each data set, we performed a data cleaning task; we removed all English stop words from all articles, and obtained data sets where each article represents a transaction (features are the corresponding words in the article).

For comparison, we implemented a parallel version of Forward Selection [7] algorithm. To specify each presented algorithm, we adopt the notations as follow. We denote by



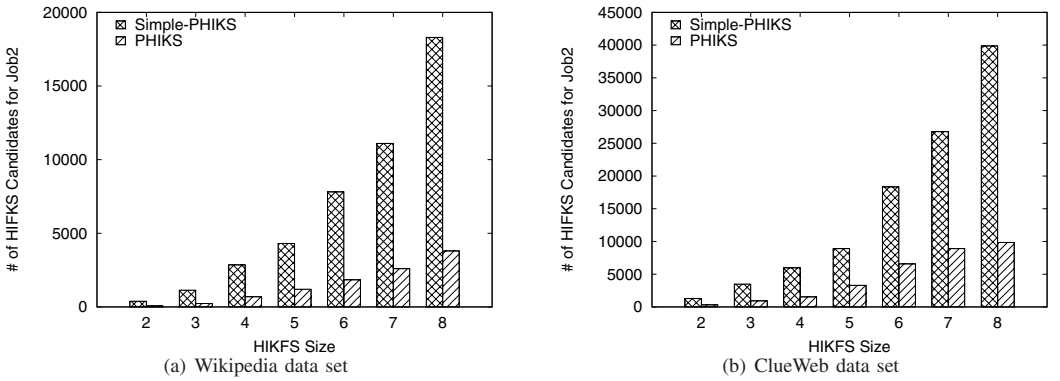
(a) All algorithms (b) Focus on scalable algorithms

Fig. 1: Runtime and scalability on English Wikipedia data set



(a) All algorithms (b) Focus on scalable algorithms

Fig. 2: Runtime and scalability on ClueWeb data set



(a) Wikipedia data set (b) ClueWeb data set

Fig. 3: Candidate pruning

'PFWS' a parallel implementation of Forward Selection algorithm, by 'Simple-PHIKS' an implementation of our basic two rounds algorithm without any optimization, and by 'Prefix' an extended version of Simple-PHIKS algorithm that uses the Prefix/Suffix method for accelerating the computations of the projection values. We denote by 'Upper-B' a version of our algorithm that reduces the number of candidates by estimating the joint entropies of *miki* based on an upper bound joint entropy. We denote by 'Upper-B-Prefix' an extended version of Upper-B algorithm that employs the technique of prefix/suffix. Lastly, we denote by 'PHIKS' an improved version of Upper-B-Prefix algorithm that uses the method of incremental entropy for reducing the number of data split scans at each mapper.

We carried out all our experiments on the Grid5000 [22] platform, which is a platform for large-scale data processing. In our experiments, we have used clusters of 16 and 48 nodes respectively for Wikipedia and ClueWeb data sets. Each machine is equipped with Linux operating system, 64 Gigabytes of main memory, Intel Xeon X3440 4 core CPUs and 320 Gigabytes SATA hard disk. In our experiments, we measured the response time of the compared algorithms, which is the time difference between the beginning and the end of a maximally informative k-itemsets mining process.

B. Runtime and Scalability

Figures 1 and 2 show the results of our experiments on both English Wikipedia and ClueWeb data sets. Figures 1(a) and 1(b) report our experimental results on the whole English Wikipedia data set. Figure 1(a) reports the performance results for an itemset of size k varying from 2 to 8. We see that the response time of Forward Selection algorithm (PFWS) grows exponentially and gets quickly very high compared to other presented algorithms. Above a size $k = 5$ of itemsets, PFWS cannot continue scaling. This is due to the multiple database scans that it performs to determine an itemset of size k (i.e., PFWS needs to perform k MapReduce jobs). In the other hand, the performance of Simple-PHIKS algorithm is better than PFWS; it continues scaling with higher k values. This difference in the performance between the two algorithms illustrates the significant impact of itemset mining in the two rounds architecture.

Moreover, by using further optimizing techniques, we clearly see the improvements in the performance. In particular, with an itemset having size $k = 8$, we observe a good performance behavior of Prefix comparing to Simple-PHIKS. This performance gain in the runtime reflects the efficient usage of Prefix/Suffix technique for speeding up *miki* parallel extraction. Interestingly, by estimating *miki* at the first MapReduce job, we record a very good response time as shown by Upper-B algorithm. In particular, with $k = 8$ we see that Upper-B algorithm roughly outperforms Simple-PHIKS by a factor of 3. By coupling the Prefix/Suffix technique with Upper-B algorithm, we see very good improvements in the response time, which is achieved by Upper-B-Prefix. Finally, by taking advantage of our incremental entropy technique for reducing the number of data split scans, we record an outstanding improvement in the response time, as shown by PHIKS algorithm.

Figure 1(b) highlights the difference between the algorithms that scale in Figure 1(a). Although Upper-B-Prefix continues to scale with $k = 8$, it is outperformed by PHIKS algorithm. With itemsets of size $k = 15$, we clearly observe a big difference in the response time between Upper-B-Prefix and PHIKS. The significant performance of PHIKS algorithm illustrates its robust and efficient core mining process.

In Figures 2(a) and 2(b), similar experiments have been conducted on the ClueWeb data set. We observe that the same order between all algorithms is kept compared to Figures 1(a) and 1(b). In particular, we see that PFWS algorithm suffers from the same limitations as could be observed on the Wikipedia data set in Figure 1(a). With an itemset size of $k = 8$, we clearly observe a significant difference between PHIKS algorithm performance and all other presented alternatives. This difference in the performance is better illustrated in Figure 2(b). By increasing the size k of *miki* from 8 to 11, we observe a very good performance of PHIKS algorithm. Although, Upper-B-Prefix algorithm scales with $k = 11$, it is outperformed by PHIKS.

C. Data Communication

Figure 3 gives a complete overview on the total number of *miki* candidates being tested at the second MapReduce job for both Simple-PHIKS and PHIKS algorithms. Figure 3(a) illustrates the number of *miki* candidates being validated at the first MapReduce job on the Wikipedia data set. By varying the parameter size k of itemsets from 2 to 8, we observe a significant difference in the number of *miki* candidates being sent by each algorithm to its second MapReduce job. With $k = 8$, Simple-PHIKS algorithm sends to its second job roughly 6 times more candidates than PHIKS. This important reduction in the number of candidates to be tested in the second job is achieved due to our efficient technique for estimating the joint entropies of *miki* with very low upper bounds. Likewise, in Figure 3(b), we record a very good performance of PHIKS comparing to Simple-PHIKS. This outstanding performance of Simple-PHIKS algorithm reflects its high capability and its effectiveness for a very fast and successful *miki* extraction.

VI. Related Work

In data mining literature, several endeavors have been made to explore informative itemsets (or featuresets, or set of attributes) in databases [2] [23] [4] [7]. Different measures of itemset informativeness (e.g., frequency of itemset co-occurrence in the database etc.) have been used to identify and distinguish informative itemsets from non-informative ones. Mining itemsets based on the co-occurrence frequency (e.g., frequent itemset mining) measure does not capture all dependencies and hidden relationships in the database, especially when the data is sparse [4]. Therefore, other measures must be taken into account. Low and high entropy measures of itemsets informativeness were proposed [4]. The authors of [4] propose

the use of tree based structure without specifying a length k of the informative itemsets to be discovered. However, as the authors of [4] mentioned, such an approach results in a very large output. [7] suggests to use a heuristic approach to extract informative itemsets of length k based on maximum joint entropy. Such maximally informative itemsets of size k is called *miki*. This approach captures the itemsets that have high joint entropies. An itemset is a *miki* if all of its constructing items shatter the data maximally. The items within a *miki* are not excluding, and do not depend on each other. [7] proposes a bunch of algorithms to extract *miki*. A brute force approach consists of performing an exhaustive search over the database to determine all *miki* of different sizes. However, this approach is not feasible due to the large number of itemsets to be determined, which results in multiple database scans. Another algorithm proposed in [7] consists of fixing a parameter k that denotes the size of the *miki* to be discovered. This algorithm proceeds by determining a top n *miki* of size 1 having highest joint entropies, then, the algorithm determines the combinations of *1-miki* of size 2 and returns the top n most informative itemsets. The process continues until it returns the top n *miki* of size k.

The problem of extracting informative itemsets was not only proposed for mining static databases. There have been also interesting works in extracting informative itemsets in data streams [24] [25]. The authors of [8] proposed an efficient method for discovering maximally informative itemsets (i.e., highly informative itemsets) from data streams based on sliding window.

Parallel mining of informative itemsets from large databases based on frequency informativeness measure has received much attention recently [26] [27] [28]. However, and to the best of our knowledge, there has been no prior work on parallel discovery of maximally informative k-itemsets from massive, distributed, databases.

VII. Conclusion

In this paper, we proposed a reliable and efficient MapReduce based parallel maximally informative k-itemset algorithm namely PHIKS, that has shown significant efficiency in terms of runtime and scalability. PHIKS elegantly determines *miki* in very large databases with at most two rounds. With PHIKS, we propose a bunch of optimizing techniques that renders the *miki* mining process very fast. These techniques concern the architecture at a global scale, but also the computation of entropy on distributed nodes, at a local scale. The result is a fast and efficient discovery of *miki* with high itemset size. Such ability to use high itemset size is mandatory when dealing with Big Data and particularly one Terabyte like what we have done in our experiments. Our results show that PHIKS algorithm outperforms other alternatives by several orders of magnitude, and makes the difference between an inoperative and a successful *miki* extraction.

References

[1] J. Han, *Data mining : concepts and techniques*. Elsevier/Morgan Kaufmann, 2012.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

[3] E. Greengrass, "Information retrieval: A survey," 2000.

[4] H. Heikinheimo, E. Hinkkanen, H. Mannila, T. Mielikäinen, and J. K. Seppänen, "Finding low-entropy sets and trees from binary data," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007, pp. 350–359.

[5] T. M. Cover, *Elements of information theory*. Hoboken, N.J: Wiley-Interscience, 2006.

[6] R. Gray, *Entropy and information theory*. New York: Springer, 2011.

[7] A. J. Knobbe and E. K. Y. Ho, "Maximally informative k-itemsets and their efficient discovery," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2006, pp. 237–244.

[8] C. Zhang and F. Masseglia, "Discovering highly informative feature sets from data streams," in *Database and Expert Systems Applications*, 2010, pp. 91–104.

[9] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," in *Proceedings of International Conference on Emerging Artificial Intelligence Applications in Computer Engineering*, 2007, pp. 3–24.

[10] Z. Ghahramani, "Unsupervised learning," in *Advanced Lectures on Machine Learning*, 2004, pp. 72–112.

[11] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conf. on Hot Topics in Cloud Computing*, 2010, pp. 10–10.

[13] T. White, *Hadoop : the definitive guide*. O'Reilly, 2012.

[14] "Hadoop," http://hadoop.apache.org, 2015.

[15] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in *IEEE International Conference on Big Data*, 2013, pp. 111–118.

[16] K. Berberich and S. Bedathur, "Computing n-gram statistics in mapreduce," in *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*, 2013, pp. 101–112.

[17] I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos, "Mind the gap: Large-scale frequent sequence mining," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2013, pp. 797–808.

[18] R. Anand, *Mining of massive datasets*. New York, N.Y. Cambridge: Cambridge University Press, 2012.

[19] A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 1995, pp. 432–444.

[20] "English wikipedia articles," http://dumps.wikimedia.org/enwiki/latest, 2014.

[21] "The clueweb09 dataset," http://www.lemurproject.org/clueweb09.php/, 2009.

[22] "Grid5000," https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[23] Han, Pei, and Yin, "Mining frequent patterns without candidate generation," *SIGMODREC: ACM SIGMOD Record*, vol. 29, 2000.

[24] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining frequent patterns in data streams at multiple time granularities," 2002.

[25] W.-G. Teng, M.-S. Chen, and P. S. Yu, "A regression-based temporal pattern mining scheme for data streams," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 2003, pp. 93–104.

[26] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "Pfp: parallel fp-growth for query recommendation," in *Proceedings of the ACM Conf. on Recommender Systems (RecSys)*, 2008, pp. 107–114.

[27] M. Riondato, J. A. DeBrabant, R. Fonseca, and E. Upfal, "Parma: a parallel randomized algorithm for approximate association rules mining in mapreduce," in *21st ACM International Conference on Information and Knowledge Management (CIKM)*, 2012, pp. 85–94.

[28] S. Tanbeer, C. Ahmed, and B.-S. Jeong, "Parallel and distributed frequent pattern mining in large databases," in *11th IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2009, pp. 407–414.