IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Frequent Itemset Mining in Big Data with Effective Single Scan Algorithms

YOUCEF DJENOURI[1,2], DJAMEL DJENOURI[3], JERRY CHUN-WEI LIN[4,5,*] AND ASMA BELHADI[6]

[1]School of Mathematics and Computer Science, Southern Denmark, Odense, Denmark
[2]Dept. of Computer Science, NTNU, Trondheim, Norway (e-mail:youcef.djenouri.ntnu.no)
[3]CERIST Research Center, Algiers, Algeria (e-mail: ddjenouri@acm.org)
[4]School of Computer Science and Technology Harbin Institute of Technology (Shenzhen), Shenzhen, China (*corresponding author, email: jerrylin@ieee.org)
[5] Department of Computing, Mathematics, and Physics Western Norway University of Applied Sciences
[6]RIMA, University of Science and Technology Houari Boumediene (USTHB), Algiers, Algeria (e-mail: abelhadi@usthb.dz)

Corresponding author: Jerry Chun-Wei Lin (e-mail: jerrylin@ieee.org).

ABSTRACT This paper considers frequent itemsets mining in transactional databases. It introduces a new accurate Single Scan approach for Frequent Itemset Mining (SSFIM), its heuristic alternative approach (EA-SSFIM), as well as its parallel implementation on Hadoop clusters (MR-SSFIM). EA-SSFIM and MR-SSFIM target sparse and big databases, respectively. The proposed approach (in all its variants) requires only one scan to extract the candidate itemsets and has the advantage to generate a fixed number of candidate itemsets independently from the value of the minimum support, which accelerates the scan process compared to existing approaches while dealing with sparse and big databases. Numerical results show that the SSFIM outperforms the state-of-the-art FIM approaches while dealing with medium and large databases. Moreover, EA-SSFIM provides similar performance as SSFIM while considerably reducing the runtime for large databases. The results also reveal the superiority of MR-SSFIM compared to the existing HPC-based solutions for FIM using sparse and big databases.

INDEX TERMS Apriori, Frequent Itemset Mining, Heuristic, Parallel Computing, Support Computing.

I. INTRODUCTION

FREQUENT Itemset Mining (FIM) aims at discovering frequent itemsets that are highly correlated in a transactional database. Let T be a set of m transactions $\{T_1, T_2, \ldots, T_m\}$ in a transactional database, and I be a set of different items or attributes, such that $\{I_1, I_2, \ldots, I_n\}$. An itemset X, is a set of items or attributes, i.e., $I \subseteq X$. The support of an itemset is defined as the ratio of the number of transactions that contains X to m. An itemset, X, is frequent if its support is no less than a user's predefined minimum support threshold, Υ_{sup}, [1]. The problem of FIM is to find an efficient approach to extract frequent itemsets in a database. FIM is used in many applications and domains such as basket analysis, social network analysis [2], biological data analysis [3] and decision making [4], [5]. FIM is also called with "Big data" applications such as in frequent genes extractions from DNA in Bio-informatics [6], frequent itemsets extraction from twitter streams in social network analysis [7], analysis of sensorial IoT devices data in smart city applications, etc. This work mainly focuses on mining the information from big transactional databases.

Accurate solutions to the FIM problem are divided into three categories. Approaches in the first category [8]–[10] use multiple scanning and are based on the Apriori heuristic [1] and adopt a generate-and-test approach. They first generate the k-sized candidate itemsets from the $(k-1)$-sized frequent itemsets and then test their frequency. Approaches in the second category [11]–[13] use two scanning steps and are based on the FPGrowth algorithm [14] and adopted a divide-and-conquer approach. They compress the transactional database in the volatile memory using an efficient tree structure then apply recursive mining process to find the frequent itemsets. The third category [5], [15] uses a single scan to derive the required information but they do not focus on the FIM problem. Since the approaches in the three categories discover all the required information against to *minimum threshold*, they are also referred to as accurate approaches. These approaches normally have high computation complexity. The *generate-and-test* approach requires multiple scans of the database, which causes increasing of computation time along with the number of transactions in the databases. The *divide-and-conquer* approach requires a limited number of scans of the

database, but it has a high memory footprint required to hold the entire database. The last category of approaches minimizes database scanning, however, they solely deal with incremental processing and handle the utility-driven problem.

This paper propose an efficient Single Scan Frequent Itemset Mining (SSFIM) for solving FIM problem. The candidate itemsets in SSFIM are first generated from each transaction and stored in a hash table that maintains the support information. Only the non-existing itemsets are inserted when they are generated in the hash table (with initial support value set to 1), while the generation of existing itemsets from a new transaction increases its support value. In the end, the frequency of itemsets in the hash table are compared to Υ_{sup} to determine the frequent itemsets and return them as the results. SSFIM is an efficient method for handling the non-sparse transactional databases, i.e., databases with transactions having low or moderate number of transactions. However, it bluntly blocked for sparse transactions (databases including some transactions with high number of items). Note that the sparseness here is not related to the size of the databases (in terms of transaction) but the size of transactions (in terms of number of items). Metaheuristics, e.g, ant colony optimization [16], swarm intelligence [17] or genetic algorithms [18], represent an alternative way to the accurate solutions for solving the mining problem. These approaches are relatively efficient in terms of runtime while dealing with big databases, but they do not guarantee to find all the required information, i.e., frequent itemsets, with support higher than Υ_{sup}. To overcome this problem, several FIM parallel algorithms [19] have been recently designed for handling big data instances through HPC (High Performance Computing) platforms, e.g., Dmine [20], and cApriori [21]. However, these algorithms still require multiple scanning of the transactional database, and their performance is usually highly sensitive to the value of Υ_{sup}. Limitations of existing approaches are addressed in this paper, which is a comprehensive extension of our previous work [22]. The major contributions of this paper are threefold as follows.

1) We first propose an SSFIM approach, which is a single scan approach to address the FIM. For sparse databases, we then propose a heuristic alternative called EA-SSFIM, which combines the evolutionary procedures and the SSFIM, i.e., the main process is performed by the SSFIM, where the generation process is established by the evolutionary procedures. Moreover, a general approach to deploy SSFIM on parallel computing architectures is presented, which is then instantiated for Hadoop clusters implementation using MapReduce programming model to yields MR-SSFIM (MapReduce for SSFIM)
2) We have first provided theoretically analysis of the SSFIM algorithm in details, as well as the EA-SSFIM and MR-SSFIM. We have also presented the statistical analysis of a large sample of real world transactional databases, and it shows that the expectation of the average runtime tend to vary linearly with the number of transactions for most databases.
3) We establish an intensive experiments to demonstrate the usefulness of SSFIM and its variants. Experiments of SSFIM showed that it outperforms other accurate FIM algorithms and achieves better scalability compared to the other algorithms. Moreover, the developed EA-SSFIM considerably reduces the runtime compared to the other approaches, while maintaining a high accuracy that is very close to SSFIM. The designed MR-SSFIM also outperforms existing HPC-based FIM approaches for big and sparse databases.

The remainder of the paper is organized as follows. Section II reviews existing FIM algorithms. Section III presents the sequential version of SSFIM, and EA-SSFIM is presented in SectionIV. The parallel version (MR-SSFIM) is described in Section V. The performance evaluation is presented in Section VI, whereas Section VII draws the conclusions.

II. RELATED WORK

In this section, accurate approaches used to solve the FIM are first presented and discussed, followed by approaches for big data.

A. ACCURATE APPROACHES TO FIM

Accurate approaches allow to extract the complete information, for instance, frequent itemsets [1], sequential patterns [23], or high-utility itemsets [24], precisely. Several accurate approaches have been developed and studied [1], [14]. According to the required for number of database scan, existing algorithms may be divided into three categories, multiple scans, two scans, and single scan, which will be respectively discussed as follows.

Multiple Scans

Agrawal et al. [1] proposed the Apriori algorithm, where candidate itemsets are generated incrementally and recursively. To generate k-sized itemsets as candidates, the algorithm calculates and combines the frequent (k-1)-sized itemsets. This process is repeated until no candidate itemsets are obtained in an iteration. Many FIM algorithms are based on the Apriori strategy such as DIC [9], and Eclat [8]. The Apriori has the limitations. The Apriori-based algorithm requires multiple scans of a transactional database, which is used to compute the support of candidate itemsets iteratively. Thus, the number of database scans required is proportional to the number of generated candidate itemsets.

Two Scans

Han et al. [14] proposed the FPGrowth algorithm, which uses a compressed FP-tree structure for mining a complete set of frequent itemsets without candidate generation. The algorithm consists of two phases: (i) construct an FP-tree that encodes the dataset by reading the database and mapping

each transaction onto a path in the FP-tree, while simultaneously counting the support of each item and, (ii) extract frequent itemsets directly from the FP-tree using a bottom-up approach to find all possible frequent itemsets ending with a particular item. Li et al. [25] proposed the NFPGrowth algorithm, which improves FPGrowth by constructing an independent head table and allows creating a frequent pattern tree only once to increase the processing speed. In [26], the authors proposed a new FPGrowth algorithm for mining uncertain data. They developed an efficient tree structure to store uncertain data, in which the occurrence count of a node is at least the sum of occurrence counts of all its children nodes. This allows to count rapidly the support of each candidate itemset. Since FPGrowth-based approaches give good results compared to the Apriori-based approaches, they require high memory consumption for construction of FP-tree structure.

Single Scan

Several works based on single scan have been proposed for solving pattern mining problems. Huang et al. [27] presented a P-tree structure to mine the set of frequent itemsets. Yun et al. [5] proposed a single scan approach for the incremental high utility pattern mining problem. The authors suggested a list-based data structure to efficiently maintain and update incremental data. Lee et al. [15] developed a new algorithm for solving erasable pattern mining problem in a streaming way. A new data structure and gain measure are incorporated to manage and update erasable patterns on one scan of the original database. However, these approaches are designed for handling incremental databases and focusing on the high-utility itemset mining. Although P-tree is related to single scan method, but it maintains the huge tree structure in the main memory, and it still requires one scan to first construct the P-tree structure then mine the frequent itemsets with an additional scan from the P-tree.

B. APPROACHES FOR BIG DATA

Several HPC-based approaches have been developed for dealing with big databases and implemented using emerging technologies, such as Hadoop, Mapreduce, MPI, and on different GPU and Cluster architectures [28]. Some of these approach are discussed in the following.

GPU-based Approaches

In [29], CU-Apriori is proposed, which develops two strategies for parallelizing both candidate itemsets generation and support counting on GPU. In the candidate generation, each thread is assigned with two frequent (k-1)-sized itemsets, it compares them to make sure that they share the common (k-2) prefix and then generates a k-sized candidate itemset. In the evaluation, each thread is assigned with one candidate itemset and counts its support by scanning the transactions simultaneously. In [30], a multilevel layer data structure is proposed to enhance the support counting of the frequent itemsets. It divides vertical data into several layers, where each layer is an index table of the next layer. This strategy can completely represent the original vertical structure. In a vertical structure, each item corresponds to a fixed-length binary vector. However, in this strategy, the length of each vector varies, which depends on the number of transactions included in the corresponding item. In [31], the Bit-Q-Apriori algorithm simplifies the process of candidate generation and support counting. Unlike the Apriori-based approach, the Bit-Q-Apriori algorithm generates k-sized candidates by joining 1-sized frequent itemsets and (k-1)-sized frequent itemsets. The bitset structure is used to store identifications of transactions that corresponds to each candidate. Therefore, support counting can be implemented using Boolean operators that reduces multiple scanning of database. In [21], the authors propose the cApriori algorithm, which compresses the transactional database to store the whole database on the shared memory of the given GPU-blocks. The results reveal that cApriori mined the Wikilinks datasets (the largest dataset on the web) in reasonable time.

Cluster-based Approaches

In [32], the BigFIM algorithm is presented, which combines principles from both Apriori and Eclat. BigFIM is implemented using the MapReduce paradigm. The mappers are determined using Eclat algorithm, whereas, the reducers are computed using the Apriori algorithm. In [33], a new HPC-based algorithm that extracts frequent patterns from big graphs is developed. The input graphs are first partitioned among the nodes. A set of optimizations and collective communication operations is then used to minimize information exchange between the different nodes. In [20], Dmine is developed for mining big graph instances. The similarity measure is proposed to partition the graphs among distributed nodes. This strategy reduces the communication between the different computational nodes. This approach has been applied to big graph containing several million nodes and several billion edges. In [34], a hadoop implementation based on MapReduce programming (FiDoop) is proposed for frequent itemsets mining problem. It incorporates the concept of FIU-tree rather than traditional FP-tree of FPgrowth algorithm, for the purpose of improving the storage of the candidate itemsets. An improved version called FiDoop-DP is proposed in [35]. It develops an efficient strategy to partition data sets among the mappers. This allows better exploration of cluster hardware architecture by avoiding jobs redundancy.

III. SINGLE SCAN FREQUENT ITEMSET MINING (SSFIM)

A. ALGORITHM DESCRIPTION

The principle of SSFIM is that an individual transaction contains all the information required to generate a certain number of candidate frequent itemsets. Transactions in a database can then be processed one by one to obtain candidate frequent itemsets. While processing transactions, the support of generated candidate itemsets can be updated, so that at the end of a single scan of the database, it becomes possible to assess which itemsets satisfy the Υ_{sup} constraint.

SSFIM generates all possible itemsets for each transaction. If an itemset that resulted from processing a transaction has already been generated when processing a previous transaction, then its support is simply incremented by one. Otherwise, it is initiated with a support set to one. The process is repeated until all transactions in the database are processed. The designed SSFIM allows to find all frequent itemsets with only one scan of the transactional database. SSFIM is also complete, i.e., the frequent itemsets are extracted directly from the transactional database and a given itemset is frequent iff it is found Υ_{sup} times in the database. Consequently, no information is lost in the itemset generation process. The SSFIM is given by Algorithm 1.

Algorithm 1 SSFIM Algorithm

1: **Input**: T: Transactional database. Υ_{sup}: Minimum support user's threshold.
2: **Output**: F: The set of frequent itemsets.
3: **for** each Transaction T_i **do**
4: S ← GenerateAllItemsets(T_i).
5: **for** each itemset t ∈ S **do**
6: **if** t ∈ h **then**
7: h(t) ← h(t)+1
8: **else**
9: init: h(t) ← 1
10: **end if**
11: **end for**
12: **end for**
13: F ← ∅
14: **for** each itemset t ∈ h **do**
15: **if** h(t) ≥ Υ_{sup} **then**
16: F ← F ∪ t
17: **end if**
18: **end for**
19: **return** F

The inputs of the algorithm are the transactional database, T, used for generating and computing the candidates itemsets, and the minimum support value Υ_{sup}. A hash table, h, is used to store all generated itemsets with their partial number of occurrences. The algorithm returns the set of all frequent itemsets, F. From each transaction in T, the set of all possible itemsets, S, is first computed (line 4). For instance, if the transaction T_i contains the items a, b, and c, then S contains the itemsets a, b, c, ab, ac, bc, and abc. Afterwards, the hash table h is updated with information (support) regarding each generated itemset, t, in S (lines 6 throughout 10). If t already exists as a key in h then the entry with key t in h ($h(t)$) is incremented. Otherwise, a new entry with is created for, t, with an support initialized to one. Finally, after completing itemset support calculation for all transactions, the final hash table is used to determine F (line 12 throughout 18). Only itemsets with support no less than Υ_{sup} are added to F.

For instance, consider the transactions $T = \{T_1 : \{a, b\}, T_2 : \{a, b, c\}, T_3 : \{a, c, d\}, T_4 : \{a, d\}\}$. SSFIM starts by scanning the first transaction $\{a, b\}$ and extracting all possible candidates itemsets, i.e., $\{a, b, ab\}$. An entry in h is then created for each candidate itemset. For the second transaction $\{a, b, c\}$, SSFIM determines all possible candidate itemsets, i.e., $\{a, b, c, ab, ac, bc, abc\}$. The itemsets a, b, and ab already exist in h, so their supports are increased, while entries are created for the remaining itemsets that are not in h yet. The same process is repeated for every transaction. In the end, the itemsets in h with support no less than Υ_{sup} are returned as frequent. In this example, if $\Upsilon_{sup} = 1/2$ then the returned set of frequent itemsets is, $\{a, b, c, ab, bc\}$. This is the set that would be returned by any accurate FIM algorithm, e.g, Apriori.

B. THEORETICAL ANALYSIS

In this section, the worst case runtime complexity of SSFIM is first analyzed and compared to that of Apriori. This will just represent an upper bound of the computation cost, but far from the real computation cost of most cases in practice. We then present a statistical analysis to derive a less pessimistic, more likely average complexity.

1) Worst Case Complexity

The runtime complexity of SSFIM is proportional to, i) the cost required for the generation of the candidate itemsets, added to, ii) the cost to determine frequent itemsets from the generated itemsets. The memory complexity is propositional to the total number of the candidate itemsets generated. For (i), the number of candidates generated from a transaction, T_i, is $2^{|T_i|} - 1$, where $|T_i|$ represents the number of items of T_i. Thus, the total number of generated candidate itemsets is,

$$\sum_{i=1}^{m}(2^{|T_i|} - 1), \tag{1}$$

where m is the number of transactions in the database T. This represents the number of steps for the generation of the candidate itemsets, and an upper bound of the memory cost as well (when all the generated itemsets at each step are not found in the hash table and thus inserted). Let, T_{max}, be the maximum number of items per transaction, then the number of candidate itemsets is bounded by $m(2^{T_{max}} - 1)$, and the generation operation in the worst case is $O(m2^{T_{max}})$ [1]

Determining frequent itemsets from candidate ones requires to scan the hash table, h, and, for each candidate itemset, to evaluate its frequency against Υ_{sup}. The number of steps for this is also given by Eq. 1. In the worst case, this operation needs $m(2^{T_{max}} - 1)$ steps. Consequently, the worst case complexity of SSFIM is,

$$O(m2^{T_{max}}) \tag{2}$$

According to Hegland [36], the theoretical complexity of the Apriori algorithm is,

$$O(m \times n^2), \tag{3}$$

[1] this also represents the worst case memory complexity (a very pessimistic upper bound)

where, n, is the number of items in the database.

The worst case complexity of SSFIM depends exponentially with T_{max} (the number of items generated per transaction), but linearly with m. The complexity of the Apriori algorithm depends linearly with m, but polynomially with n. Generally, $m \ll n$, and T_{max} is dominated by n and m. In this case, Eq. 2 is considered asymptotically better than Eq. 3. This is fulfilled in most existing transactional databases (as it will be justified later). For instance, in the well-known case of basket analysis, the number of products in stock at a supermarket can reach several thousands, whereas the average number of products bought by an individual customer hardly exceeds a few dozens. In the remainder of this section, we will statistically analysis a large sample of databases to infer expected value of T_{max} and then stochastically analyze the theoretical complexity given above.

2) Statistical Analysis

The theoretical analysis provided thus far shows that the complexity of SSFIM depends on the transaction's size, i.e., the number of items per transaction, rather than the problem's size (n). Theoretically, the transaction's size, may vary from 1 to the size of the database (n). But in practice, bounding the terms T_i with n is very pessimistic and unrealistic. In the following, we stochastically derive more reasonable average complexity through an estimation of the distribution of the expected number of items per transaction using real databases. 500 different database instances have been retrieved from renowned UCI machine learning and frequent itemsets repositories[2].

1) **Runtime Analysis**: Let $D = \{D_j\}$ be a set of real world database instances, where each D_j is a different database, $O_{i,j}$ the number of transactions in D_j containing i items, and ρ_i the percentage of transactions containing i items in D, i.e.,

$$\rho_i = \frac{\sum_{j=1}^{|D|} O_{i,j}}{\sum_{j=1}^{|D|} |D_j|} \tag{4}$$

From a statistical standpoint, D is the set of population *population* to be studied, while each data instance represents an *individual* in the population. Our objective is to estimate from the identified population the random variable, x, which represents the number of items per transaction.

The 500 selected databases are used to determine, ρ_i, for each number of items, i.

Fig. 1 shows the distribution of the number of items per transactions in the population (red color). Using curve fitting on the obtained results, we estimated the number of items per transaction in the population with a normal (Gaussian) distribution, $x \sim \mathcal{N}(\mu, \sigma^2)$, with $\mu = 20$ and $\sigma^2 = 5.2$., (which is plotted in Fig. 1 as well). We performed χ^2 test with XLSTAT using a significance level $\alpha = 2\%$, and the results show that the test hypothesis "the data instances follow a Normal distribution with mean 20 and variance 5.2", cannot be rejected, i.e., this confirms the hypothesis with a 98% confidence interval.

[2]available at https://archive.ics.uci.edu/ml/datasets.html and http://fimi.ua.ac.be/data/, respectively

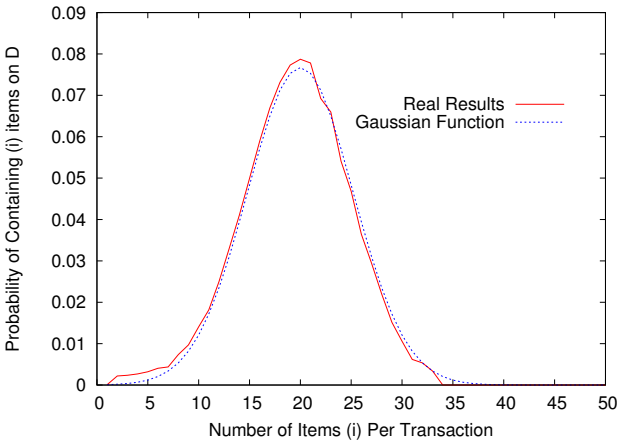


FIGURE 1. Normal Distribution of the number of items $\mathcal{X} \rightarrow \mathcal{N}(20, 5.2)$ using 500 sample database instances

By replacing $|T_i|$ with x in Eq. 1 for both steps (candidate itemsets generation and determining frequent itemsets from candidates), the term representing the complexity (now average complexity) becomes $2\sum_{i=1}^{m}(2^x - 1)$.

Let us denote this function by $g(x)$. The aim now is to determine the expected value of $g(x)$, i.e., $E[g(x)]$, given that the variable x follows a normal distribution. We have, $E[g(x)] = E[\sum_{i=1}^{m}(2^x - 1)] = \sum_{i=1}^{m}(E[2^x] - 1)$, which leads to:

$$E[g(x)] = m(E[2^x] - 1) \tag{5}$$

Since, $E[h(x)] = \int_{-\infty}^{+\infty} h(x)f(x)dx$, where $f(x)$ is the probability density of the normal distribution. This yields,

$$E[g(x)] = \int_{-\infty}^{+\infty} m(E[2^x] - 1)\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx \tag{6}$$

After integration and simplification, we obtain,

$$E[g(x)] = m(\frac{e^{\frac{\mu^2\sigma^4 + \ln(2)^2}{4\sigma^{16}}}\sqrt{\sigma^2}}{8\sqrt{2}} - 1) \tag{7}$$

Now, by replacing μ and σ^2 with the results obtained in our statistical study, we obtain,

$$E[g(x)] = 7.10 \times m \tag{8}$$

Based on this statistical test of the number of items per transaction in real world database instances, we realize that SSFIM complexity is more likely to depend linearly (on average) with m, while the Apriori heuristic complexity is quadratic (Eq.3).

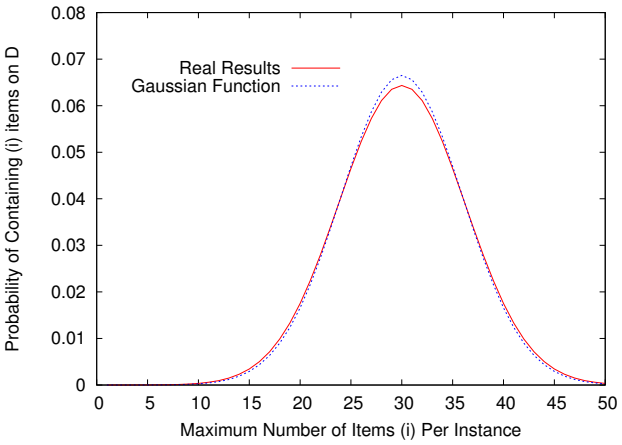


FIGURE 2. Normal Distribution of the maximum number of items per instance $\mathcal{X} \rightarrow \mathcal{N}(30, 6.0)$ using 500 sample database instances

2) **Memory Cost and Applicability Analysis**: The previous analysis on the average number of items gives a clear vision on the average runtime complexity. However, the applicability of the algorithm is directly affected by the memory cost, which in turn is affected by the maximum number items per transaction (and not the average one). Let us define $T_{max}^{D_j}$ as the maximum number of items in D_j, and, O_i, as the number of instances having maximum number of items equal to i. O_i can be defined as follows:

$$O(i) = |\{D_j / T_{max}^{D_j} = i\}| \tag{9}$$

ρ_i is redefined as the percentage of instances in D that contain i items as maximum number of items, i.e.,

$$\rho_i = \frac{O(i)}{|D|} \tag{10}$$

Our objective now is to estimate from the identified population the random variable, x, which represents the maximum number of items per instance.
The previous selected 500 databases are used to determine ρ_i for each number of items i.
Fig. 2 shows the distribution of the maximum number of items per instance in the population (solid line). Using curve fitting on the obtained results, we estimated the maximum number of items per instance in the population with a normal distribution, $x \sim \mathcal{N}(\mu, \sigma^2)$, with $\mu = 30$ and $\sigma^2 = 6.0$.. Similarly to the previous test, the χ^2 test is performed with a significance level $\alpha = 2\%$, and the results show that the test hypothesis "the data instances follow a Normal distribution with mean 30 and variance 6.0", cannot be rejected, i.e., this confirms the hypothesis with a 98% confidence interval.
Our experiments show that the parallel implementation (to be presented later) allows to safely run all databases with up to $T_{max}^{D_j} = 37$. Beyond this limit, the algorithm starts crashing. From this statistical sample that has confidence interval of 98% (normal distribution in Fig. 2), we can check that the ratio of databases having $T_{max}^{D_j} \leq 37$ is 0.91. Therefore, we conclude the coverage probability of our parallel (accurate) implementation is no less than 0.91. For the uncovered databases, we propose an alternative approach (EA-SSFIM), which is presented in the Section IV. Note that the uncovered databases are some sporadic databases with $T_{max}^{D_j} > 37$.
To derive memory complexity, we follow the same process as in previous step (except that $g(x) = \sum_{i=1}^{m}(2^x - 1)$ instead of $2\sum_{i=1}^{m}(2^x - 1)$. We obtain,

$$E[g(x)] = 12.75 \times m \tag{11}$$

This statistical test based on the maximum number of items per instance confirms again that the memory cost of SSFIM depends only on the number of transactions, m, in a database and it is unrelated to n. Moreover, SSFIM complexity is more likely to depend linearly with, m, contrary to the Apriori-based approach.

IV. EA-SSFIM

A. PRINCIPLE

This section presents an alternative approach of SSFIM called EA-SSFIM, which employs an evolutionary algorithm in the generation process. The main difference between SSFIM and EA-SSFIM is in the generation process of the itemsets from each transaction. An evolutionary algorithm is applied to generate potential solutions from each transaction T_i. Afterwards, the supports of the generated solutions (itemsets) are maintained in a hash table. This process is repeated for all transactions. The satisfied itemsets with supports greater than Υ_{sup} are then returned. In the following, we present the main operation of the EA-SSFIM, which is the generation process.

B. GENERATION PROCESS

The evolutionary algorithm is first applied for each transaction; it starts by initializing population from each transaction T_i, and then applies crossover, mutation, and selection operators for the generated chromosomes. To efficiently generate potential solutions, the hash table is used to guide the search process. This process is repeated until the maximum number of iterations is reached. We denote by ω_i the solution space of the transaction T_i. The solution space of the EA depends to the number of items per transaction. The size of the solution space ω_i is calculated as: $2^{|T_i|} - 1$.

The algorithm defines intelligent operators to explore potential solutions from each ω_i. Each solution s is defined as a vector of $|T_i|$ elements. The $j - th$ element is set to 1, if the $j - th$ item of the transaction T_i appears in s; 0, otherwise. The population initialization is first performed by randomly generating $popsize_i$ individuals from the solution space ω_i. For instance, if we have a transaction such that T_1:$\{a, b, c, d, e\}$, then ω_1 contains 31 potential solutions. If

Algorithm 2 Generation Process

Input: T_i: The i^{th} transaction. $PopSize_i$: Population size of ω_i of T_i. MAX_i: The maximum number of iterations of ω_i of T_i. Υ_{sup}: Minimum support user's threshold.
Output: S_i: Potential solutions in ω_i.
$S_i \leftarrow \emptyset$
for j=1 to MAX_i **do**
  $Indiv \leftarrow PopulationInitialization(\omega_i, PopSize_i)$
  $S_i \leftarrow S_i \cup Indiv$
  for each $(indiv_1, indiv_2) \in$ Indiv **do**
    $Offs \leftarrow Crossover(Indiv_1, Indiv_2)$
    $S_i \leftarrow S_i \cup Offs$
    for each $offs \in OFFS$ **do**
      $S_i \leftarrow S_i \cup Mutation(offs)$
    end for
  end for
  for each solution $s \in S_i$ **do**
    if $h(s) \geq h(\bar{s})$ **then**
      $S_i \leftarrow S_i \cup Intensify(s)$
    else
      $S_i \leftarrow S_i \cup Intensify(\bar{s})$
    end if
  end for
  Selection(Pop_size_i)
  $j \leftarrow j + 1$
end for
return S_i

we set $popsize_1$ as 2, the individuals $indiv_1$: 00111 represents the itemset $\{cde\}$ and the individuals $indiv_2$: 01001 represents the itemset $\{be\}$. The crossover and the mutation operators are then performed to generate other potential solutions.

Afterwards, for a given solution s, if the current support of s is greater than the current support of $\bar{s}$, the intensification in s is launched. Otherwise, the diversification from $\bar{s}$ is established. Note that $\bar{s}$ is defined by the set of items that does not belong to s. For instance, if s : 00111, then $\bar{s}$: 11000. This process is repeated for maximum number of iterations MAX_i. The algorithm of the generation process is given in Algorithm 2. The main EA operators in ω_i is explained as follows:

a: Crossover

The crossover operator is applied on a parents to generate two offsprings. The two parents are divided in two parts by selecting a crossover point *cp*. The first offspring is generated by merging the first part of the first parent and the second part of the second parent. The second offspring is generated by merging the first part of the second parent and the second part of the first parent. For instance, consider the previous example, and the two parents p_1:00111 and p_2:01001, and the crossover point *cp*=2. The offsprings generated from these parents respectively are off_1:00111, which represents the itemset $\{cde\}$; and off_2:0111, which represents the itemset $\{bcde\}$.

b: Mutation:

The mutation operator is applied on each generated offspring by flipping the value of the gene having less support value in the hash table entry. For instance, consider the offsprings off:1100 of the transaction T_3:{*a*,*b*,*c*,*f*}, and $h(a)$ is set to 1 and $h(b)$ is set to 2. The mutation operator yields the offsprings $\{01100\}$ by replacing the item a with the item c. In the case of finding two lower items with the same support, one of these items is selected randomly to perform the mutation operator.

c: Intensification:

It is launched when the current support of the a solution, say s, is greater than the current support of the remaining itemsets in the transaction of s. This allows to look for another pertinent solutions in s. The intensification is performed by removing one item at the same time from the solution s. For example, considering the transaction $\{acde\}$ and the solution s:0111 that represents the itemset $\{cde\}$. With the intensification, the following itemsets are generated, $\{cd\}$, $\{ce\}$, and $\{de\}$.

d: Diversification:

It is established when the current support of the given solution s is less than the current support of the remaining itemsets in the transaction of s. This allows to explore another regions in the solution space of the given transaction. The diversification is performed by removing one item at the same time from the solution $\bar{s}$. For example, considering the transaction $\{abcdef\}$ and the solution s : 000111. We first compute $\bar{s}$: 111000 that represents the itemsets $\{abc\}$. With the diversification, the itemsets $\{abc\}$, $\{ab\}$, $\{ac\}$, and $\{bc\}$ are generated.

The memory complexity of itemset generation in the i-th transaction is always polynomial, i.e., it is $O(IMAX_i \times PopSize_i)$. This provides tremendous asymptotic reduction compared to SSFIM that has exponential complexity, i.e., $O(2^{|T_i|-1})$. The worst case total complexity for EA-SSFIM is $O(m \times \max_i\{IMAX_i\} \times \max_i\{PopSize_i\} \times T_{max})$, which is always $O(m \times T_{max})$.

V. PARALLEL IMPLEMENTATION: MR-SSFIM

To run SSFIM on any parallel architecture, the following sequential steps have to be followed:

1) **Partition the database**: the transactional database is divided into many partitions, whereby each partition contains a set of transactions. To ensure load balancing among the different parallel nodes, the partitions should have the same size (the same number of transactions).
2) **Compute and store the local results**: in this step, each parallel node generates all itemsets from the

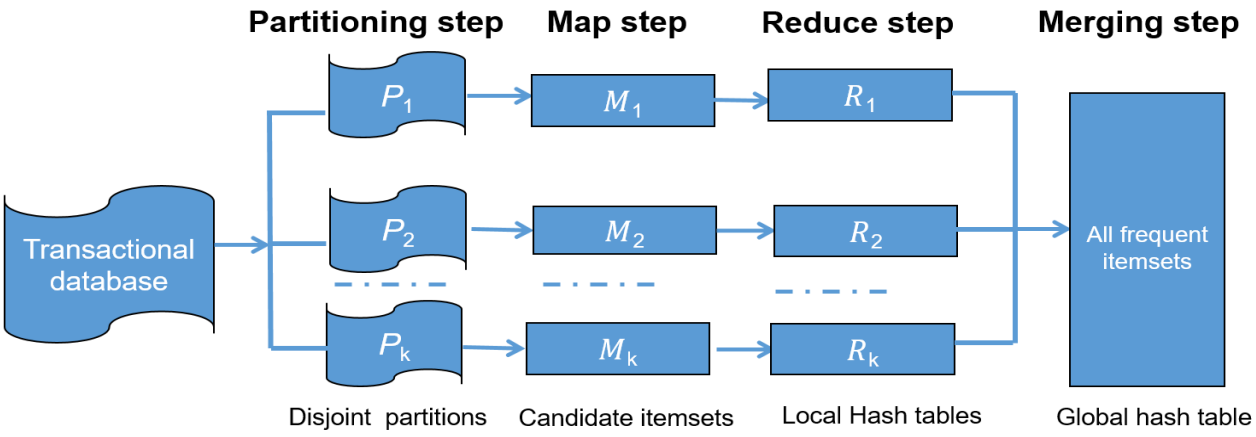


FIGURE 3. MR-SSFIM Framework

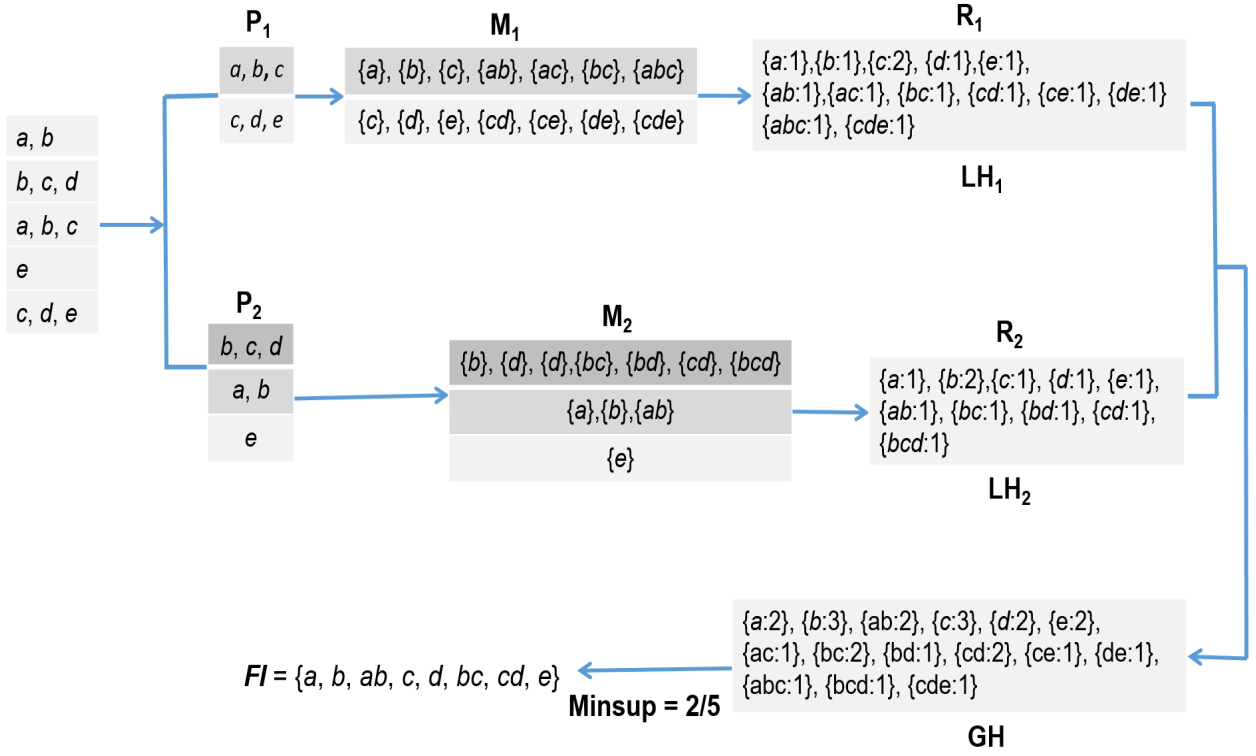


FIGURE 4. MR-SSFIM Illustration

transactions that are assigned to it and stores them in a local hash table. The latter is built following the same logic of building the hash table, h, in the sequential implementation of SSFIM (See Section III). That is, when a new candidate itemset is generated, a new entry is initialized in the hash table or, if the itemset is already indexed by the hash table, then its frequency is incremented by one.

3) **Merge local results**: once the local hash tables are calculated, a designated node should merge the local hash tables into a global one. This can be done using a sum reduction technique [37].
4) **Send back the results to CPU**: finally, the global hash table is scanned by the designated node to retain only the itemsets that exceed the minimum support constraint, which are sent back to the CPU node.

The instantiation of above four steps must be carefully designed to fit the hardware in use. In the remainder of this section, MapReduce instantiation of this generic approach is presented. MapReduce is a popular data processing paradigm for efficient and fault tolerant workload distribution in large clusters. MapReduce is a well-known data processing model first suggested by Deanand Ghemawat [38]. A MapReduce computation mainly operates on two phases: i) the Map

Algorithm 3 MR-SSFIM

Input: T: Transactional database. Υ_{sup}: Minimum support user's threshold.
Output: F: The set of frequent Itemsets.
{Partition}
P $\leftarrow$ SLS(T)
for each partition P_i **do**
  {Map}
  $C_i \leftarrow \emptyset$
  for $T_i^j \in P_i$ **do**
    $C_i \leftarrow C_i \cup GenerateAllItemsets(T_i^j)$
  end for
  $< C_i, |C_i| > \leftarrow < key, value >$
  {Reduce}
  $LH_i \leftarrow \emptyset$
  for $c \in C_i$ **do**
    $LH_i(c) \leftarrow 1$
    for $c' \in C_i$ **do**
      if $c' = c$ **then**
        $LH_i(c) \leftarrow LH_i(c) + 1$
      end if
    end for
  end for
end for
{Merging}
$GH \leftarrow \emptyset$
for $i = 1 \in k$ **do**
  for $c \in LH_i$ **do**
    if $c \in GH$ **then**
      $GH(c) \leftarrow GH(c) + LH_i(c)$
    else
      $GH(c) \leftarrow LH_i(c)$
    end if
  end for
end for
$F \leftarrow \emptyset$
for $t \in GH$ **do**
  if $GH(t) \geq \Upsilon_{sup}$ **then**
    F $\leftarrow$ F $\cup$ t
  end if
end for
return F

phase: Splits the input database into high number of partitions, and assigned to different cluster nodes. ii) the Reducer step first receive the output of the mappers and then produce the final results. MR-SSFIM (the instantiation of SSFIM on Hadoop clusters, see Fig. 3) uses the MapReduce model. Step by step explanation of MR-SSFIM is given as follows. The transactional database, T, is first divided into the set of k disjoint partitions $P = \{P_1, P_2 ... P_k\}$, where $P_i \cap P_j = \emptyset$, $\forall (i,j) \in [1..k]^2$ and $\bigcup_{i=1}^{k} P_i = T$. To ensure load balancing between mappers, the partitions are constructed by minimizing the following function as:

$$(\sum_{i=1}^{k}(\sum_{j=1}^{k}(\sum_{l=1}^{|P_i|}(2^{T_i^l} - 1) - \sum_{l=1}^{|P_j|}(2^{T_j^l} - 1)))) \quad (12)$$

Solving this equation by accurate solver requires high computational time. The SLS (Stochastic Local Search) algorithm [39] is used to find the set of partitions P. Each partition P_i is then sent to the mapper $\mathcal{M}_i$. The latter processes the transactions of the partition, P_i, and generates all candidate itemsets from each transaction T_i^j.

Gradually, it creates a set of candidate itemsets $\mathcal{C}_i$. When $\mathcal{M}_i$ scans all transactions of the partition, P_i, it sends $\mathcal{C}_i$ to the reducer $\mathcal{R}_i$. The reducer $\mathcal{R}_i$ scans the candidate itemset $\mathcal{C}_i$, and computes the local support of each itemset that belongs to $\mathcal{C}_i$. This allows to create the local hash table, $\mathcal{LH}_i$. The merging step is then performed to determine the global support of all itemsets and extract all frequent itemsets from the global hash table $\mathcal{GH}$. The pseudo-code of MR-SSFIM is given in Algorithm 3. From a theoretical standpoint, MR-SSFIM improves the baseline SSFIM by exploiting Hadoop clusters while generating and evaluating candidate itemsets. The load balancing is respected, since the transactions are well assigned to the mappers. Mappers can process transactions independently and send the results to the reducers. The reducers then incrementally updates the local hash table to build the global hash table, and extract the set of all frequent itemsets.

Fig. 4 illustrates MR-SSFIM on two cluster nodes. The set of transactions {{*a*,*b*}, {*b*,*c*,*d*}, {*a*,*b*,*c*}, {*e*}, {*c*,*d*,*e*}} is partitioned into two partitions $P_1 : \{\{a, b, c\}, \{c, d, e\}\}$, and $P_2 : \{\{b, c, d\}, \{a, b\}, \{e\}\}$. The first mapper $\mathcal{M}_1$ generates 14 candidate itemsets from P_1, whereas the second mapper $\mathcal{M}_2$ generates 11 itemsets from P_2. The load balancing is respected thanks to the partitioning strategy employed in MR-SSFIM. Afterwards, the reducers $\mathcal{R}_1$ and $\mathcal{R}_2$ create the local hash tables $\mathcal{LH}_1$, and $\mathcal{LH}_2$, respectively from the data output of the mappers $\mathcal{M}_1$, and $\mathcal{M}_2$. The local hash tables are merged to determine the global hash table $\mathcal{GH}$. For instance, the support of the itemset $\{c\}$ is 3 in $\mathcal{GH}$, which is obtained by merging the entry $\{c{:}2\}$ of $\mathcal{LH}_1$, and the entry $\{c{:}1\}$ of $\mathcal{LH}_2$. At the end, the set of all frequent itemsets $F = \{\{a\}, \{b\}, \{ab\}, \{c\}, \{d\}, \{bc\}, \{cd\}, \{e\}\}$ is derived using $2/5$ as minimum support threshold.

VI. NUMERICAL ANALYSIS

The proposed approaches are experimentally evaluated using different transactional database instances. Database instances are classified according to their size, i.e., the number of transactions, into small (with less than $3,000$ transactions), average (between $3,000$ and $80,000$ transactions), large ($80,000$ to $520,000$ transactions), and big data instances with more than 5 million transactions. Small, average and large instances have been retrieved from the Frequent Mining Dataset Repository[3], which are scientific repositories com-

[3] http://fimi.ua.ac.be/

TABLE 1. Database instances used in the evaluation

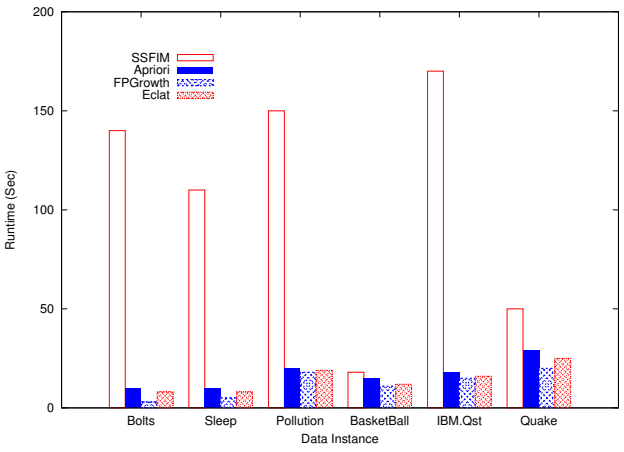
Data set Type	Data set Name	m	n	i
Small	Bolts	40	8	8
	Sleep	56	8	8
	Pollution	60	16	16
	Basket ball	96	5	5
	IBM Quest Standard	1,000	40	20
	Quake	2,178	4	4
Average	Chess	3,196	75	37
	Mushroom	8,124	119	23
	BMS-WebView-1	59,602	497	2.5
	BMS-WebView-2	77,512	3,340	5
Large	Retail	88,162	16,469	10
	Connect	100,000	999	10
	BMP-POS	515,597	1,657	6.5
Big data instance	Wikilinks	5,706,071	3,773,865	23

monly used for benchmarking in the data mining community. The big data instance used in our evaluation is built from real data retrieved from the Internet; the **Wikilinks**[4]. It is a collection of documents representing a subset of Wikipedia pages. It contains 40 millions documents over 3 million entities. The NLP techniques implemented in the NLTK Python package are used to retrieve and transform the Wikilinks documents into the transactional form. The database obtained contains about 5 million transactions.

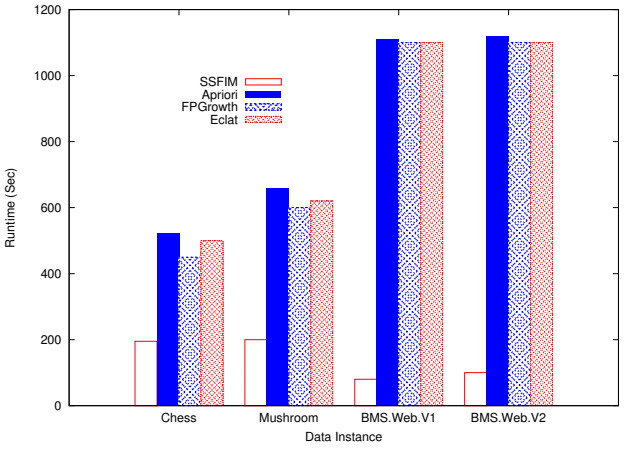
Table 1 describes the different database instances used in the evaluation. Note that the datasets show large variability with regard to the total number of items n. The average number of items per transaction, i, tend to remain limited (below 37) when compared to the database size m. That is, the number of transactions in a database.

A. SSFIM PERFORMANCE

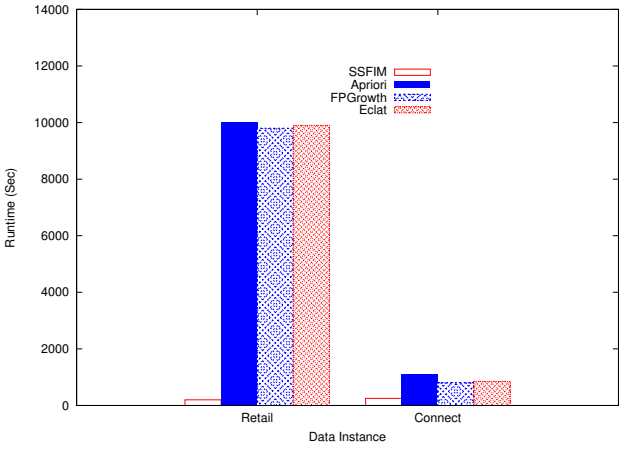
Fig. 5 presents the runtime performance of SSFIM, Apriori, FPGrowth and Eclat, using small, average and large FIM instances. For small instances, SSFIM takes more time compared to other FIM algorithms. However, for average and large instances, SSFIM outperforms all the other algorithms. These results confirm the effectiveness and superiority of the proposed single scan approach when dealing with non-dense and large transactional databases. The next experiment investigates the impact of the minimum support parameter. Remember that the Apriori based algorithms are highly sensible to the minimum support. Fig. 6 shows the runtime performance of Apriori and SSFIM using the BMP-POS instance, i.e., the largest instance in the literature not classified as Big Data. By varying the minimum support (from 100% to 10%), the execution time of the Apriori algorithm highly increases, while the runtime of SSFIM remains stable and further bellow Apriori. The results confirm that the SSFIM is not sensitive to variations of the minimum support. This is justified by the fact that SSFIM is a transaction-based approach in which the number of generated candidates itemsets does not depend on the chosen minimum support.



Small instances



Average instances



Large instances

FIGURE 5. Runtime (Sec) of SSFIM and state-of-the-art FIM approaches

[4]http://www.iesl.cs.umass.edu/data/wiki-links

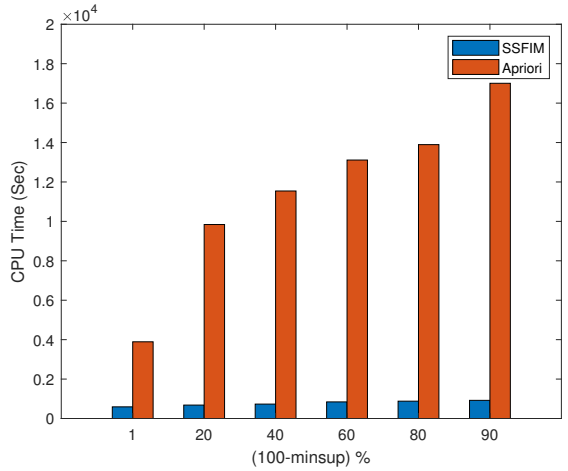


FIGURE 6. Runtime (Sec) of SSFIM and Apriori for different minimum support (%) using BMP-POS instance

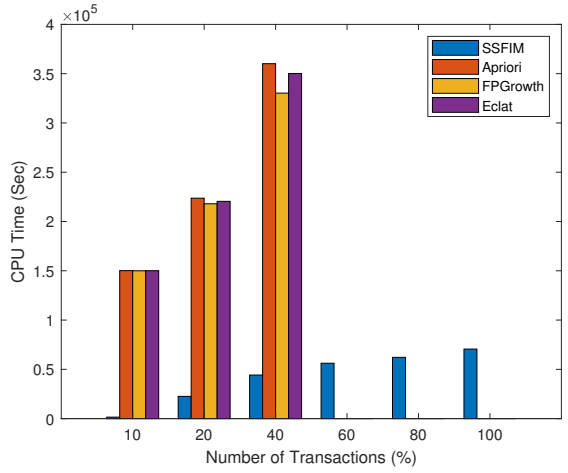


FIGURE 7. Runtime (Sec) of SSFIM and sequential FIM approaches with different number of transactions using Wikilinks instance

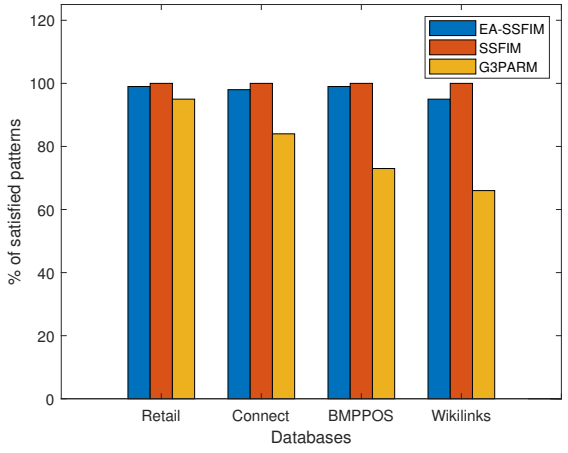


FIGURE 8. Percentage of satisfied patterns of EASSFIM, SSFIM, and G3PARM using large and big instances

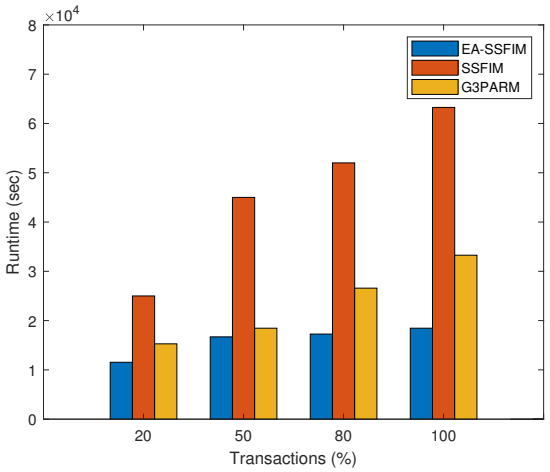


FIGURE 9. Runtime in seconds of EASSFIM, SSFIM, and G3PARM using Wikilinks

Conversely, the Apriori heuristic is an item-based approach, where the number of generated candidates inversely depends on the minimum support.

The last experiments regarding the sequential version of SSFIM test its scalability when dealing with big data instance. The particularity of this Big Data instance (Wikilinks) is that the average number of items per transaction and the maximum number of items per transaction are relatively low and do not exceed 23 and 35, respectively. Given the theoretical and statistical analysis presented in Section III-B, we anticipate that SSFIM is a suitable candidate to solve this Big Data instance in relatively reasonable time. Fig. 7 presents the runtime performance of SSFIM, Apriori, FP-Growth, Eclat for the Big Data instances by varying the number of transactions that are considered in the mining process from 10% to 100%. The results show that the run time of SSFIM remains relatively stable and the increase with the number of transaction is much smoother compared to the other algorithms. For example, to process the entire Wikilinks instance, SSFIM needs about 20 hours, while the other algorithms take more than 50 hours for just 40% of transactions. Further, all these algorithms (except SSFIM) were not able to extract more than 40% frequent itemsets from the Wikilinks after several days of running that ended with system crashing. This is in line with results reported in the literature, e.g., it has been shown in [21] that these algorithms have been executed for more than 20 days without returning any results while processing this big data instance .

The results clearly demonstrate the superiority of SSFIM when treating big data instances. However, the sequential SSFIM still needs huge time when dealing with these instances. More than 10 hours of runtime is acceptable only for analyzing delay tolerant static data, but not in delay sensitive applications such as analysis of sensorial data in IoT applications (e.g., road traffic management, energy management decisions in smart grids, etc.), realtime mining of financial data, etc.

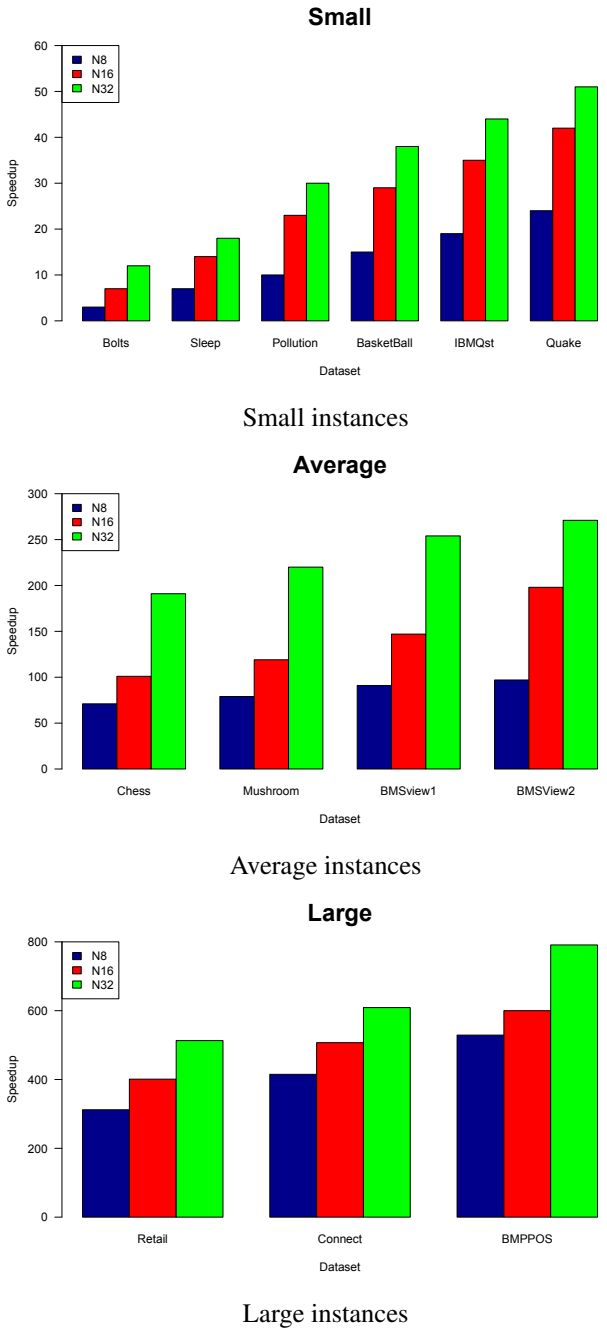


FIGURE 10. Speedup of MR-SSFIM approach with different number of nodes (8, 16, and 32)

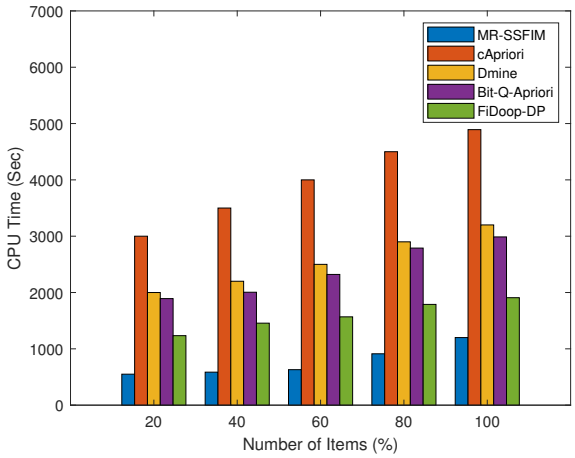


FIGURE 11. Runtime of MR-SSFIM and state of the art parallel-based approaches with different percentage of items (From 20% to 100%) using Wikilinks

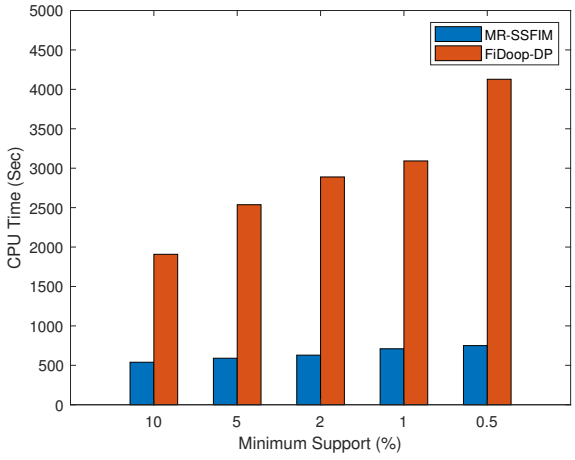


FIGURE 12. Runtime of MR-SSFIM and FiDoop-DP with different minimum support values (From 10% to 0.5%) using Wikilinks

B. EA-SSFIM PERFORMANCE

In this experiment, we compare EA-SSFIM with the SSFIM (accurate version), and the improved version of G3PARM [40]. Note that for small and average data instances, percentage of satisfaction for all algorithms is 100%. Fig. 8 presents the percentage of frequent itemsets on large and big data instances for the three algorithms. The results reveal that EA-SSFIM outperforms the baseline algorithms G3PARM and it is able to find more than 95% of frequent itemsets in all cases. This result is explained by the fact that the designed EA-SSFIM has the ability to deal with large and big solution space using both SSFIM and evolutionary procedures, where the solution space is well reduced by SSFIM (only the itemsets of the transactions are considered), and well explored by the evolutionary procedures. This is contrary to the G3PARM, where the solution space is defined by all items. Fig. 9 presents the execution time (in seconds) of EA-SSFIM, SSFIM, and G3PARM using Wikilinks instance.

By varying the percentage of transactions from 20% to 100%, EA-SSFIM outperforms the G3PARM and SSFIM. For mining all the transactions (100%), the runtime of SSFIM exceeds 60000 seconds, and the runtime of G3PARM exceeds 30000 seconds, whereas the runtime of EA-SSFIM below 19000 seconds. These results are justified by the combination of SSFIM and evolutionary procedures to explore big solution space. However, the runtime is still high for big data instances such as Wikilinks. In the next section, we show

the performance the parallel implementation (MR-SSFIM) for dealing with big Wikilinks instance.

C. MR-SSFIM PERFORMANCE

Fig. 10 shows the speed up of MR-SSFIM compared to the sequential version using small, medium and large instances, by varying the number of nodes from 8 to 32, and with different datasets. The results show that the speed up of MR-SSFIM considerably increases with the number of nodes. For large instances, in particular, the speed up reaches up to 700. These results confirm the significant improvement by the parallel approach over the sequential SSFIM. They also demonstrate that MR-SSFIM is more efficient when increasing the number of nodes and the database sizes. In the last experiments on Wikilinks instance that will be presented in the next section, we evaluate MR-SSFIM using 32 nodes. The last experiment considers big data instances (Wikilinks) and compares MR-SSFIM to some state-of-the-art HPC-based FIM approaches such as cApriori [21], Dmine [20], Bit-Q-Apriori [31] and FiDoop-DP [35]. Fig. 11 presents the runtime by varying the percentage of items from 20% to 100%. The Results show that MR-SSFIM outperforms all the other HPC-based algorithms. MR-SSFIM extracts all frequent itemsets in the Wikilinks instance in less than 20 minutes, while the sequential version of SSFIM performed that in about 10 hours (Fig. 7), and the best performing HPC-based FIM approach takes about 2 hours. These results confirm the efficient design of MR-SSFIM, whose complexity does not depend on the number of items in a database but only on the number of items per transaction. Fig. 12 presents the runtime performance of MR-SSFIM, and FiDoop-DP [35] for Wikilinks instance by varying the minimum support from 10% to 0.5%. The results show that the execution time of MR-SSFIM remains relatively stable compared to the FiDoop-DP with the decrease of minimum support value. These results confirm again the non sensitivity of the designed approach vs. the minimum support constraint.

VII. CONCLUSIONS AND PERSPECTIVES

SSFIM, a new and efficient frequent itemset mining approach for big databases is proposed in this paper. It aims at discovering frequent itemsets with only one scan of the transactional database. Two variants EA-SSFIM and MR-SSFIM are also developed respectively based on evolutionary appraoch and MapReduce model. Experimental evaluation reveals that SSFIM, and EA-SSFIM outperform the baseline FIM algorithms for different database sizes, and that EA-SSFIM is able to discover the required information within a reasonable time. Moreover, the results reveal that the MR-SSFIM reaches up to 800 speed up and outperforms the existing HPC-based FIM solutions when dealing with big sparse databases.

Motivated by the promising results shown in this paper, we plan to extend SSFIM for solving domain-specific complex problems requiring the mining of big data. This can be found, for instance, in the context of business intelligence applications or in the context of mining financial data. Adapting and incorporating SSFIM with the existing incremental pattern mining approaches are also considered as our future agenda.

ACKNOWLEDGMENT

This research was partially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61503092, by the Shenzhen Technical Project under JCYJ20170307151733005 and KQJSCX20170726103424709

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in ACM SIGMOD Record, vol. 22, no. 2. ACM, 1993, pp. 207–216.
[2] M. A. Bhuiyan and M. Al Hasan, "An iterative mapreduce based frequent subgraph mining algorithm," IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 3, pp. 608–620, 2015.
[3] F. Pan, G. Cong, A. K. Tung, J. Yang, and M. J. Zaki, "Carpenter: Finding closed patterns in long biological datasets," in Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2003, pp. 637–642.
[4] Y. Djenouri, Z. Habbas, and D. Djenouri, "Data mining-based decomposition for solving the maxsat problem: toward a new approach," IEEE Intelligent Systems, vol. 32, no. 4, pp. 48–58, 2017.
[5] U. Yun, H. Ryang, G. Lee, and H. Fujita, "An efficient algorithm for mining high utility patterns from incremental databases with one database scan," Knowledge-Based Systems, vol. 124, pp. 188–206, 2017.
[6] S. Su, S. Xu, X. Cheng, Z. Li, and F. Yang, "Differentially private frequent itemset mining via transaction splitting," IEEE transactions on knowledge and data engineering, vol. 27, no. 7, pp. 1875–1891, 2015.
[7] J. Leskovec and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," ACM Transactions on Intelligent Systems and Technology, vol. 8, no. 1, p. 1, 2016.
[8] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li et al., "New algorithms for fast discovery of association rules." in KDD, vol. 97, 1997, pp. 283–286.
[9] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," ACM SIGMOD Record, vol. 26, no. 2, pp. 255–264, 1997.
[10] M. J. Zaki and C. J. Hsiao, "Charm: An efficient algorithm for closed itemset mining," in Proceedings of the 2002 SIAM international conference on data mining. SIAM, 2002, pp. 457–473.
[11] 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.
[12] C. Borgelt, "An implementation of the fp-growth algorithm," in Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations. ACM, 2005, pp. 1–5.
[13] K. Wang, L. Tang, J. Han, and J. Liu, "Top down fp-growth for association rule mining," in Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2002, pp. 334–340.
[14] 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.
[15] G. Lee and U. Yun, "Single-pass based efficient erasable pattern mining using list data structure on dynamic incremental databases," Future Generation Computer Systems, vol. 80, pp. 12–28, 2018.
[16] J. M. T. Wu, J. Zhan, and J. C. W. Lin, "An aco-based approach to mine high-utility itemsets," Knowledge-Based Systems, vol. 116, no. 15, pp. 102–1136, 2017.
[17] J. C. W. Lin, L. Yang, P. Fournier-Viger, T. P. Hong, and M. Voznak, "A binary pso approach to mine high-utility itemsets," Soft Computing, vol. 21, no. 17, pp. 5103–5121, 2017.
[18] Y. Djenouri and M. Comuzzi, "Combining apriori heuristic and bio-inspired algorithms for solving the frequent itemsets mining problem," Information Sciences, vol. 420, pp. 1–15, 2017.
[19] W. Gan, J. C. W. Lin, H. C. Chao, and J. Zhan, "Data mining in distributed environment: a survey," Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 7, no. 6, p. e1216, 2017.
[20] Y. Yuan, G. Wang, J. Y. Xu, and L. Chen, "Efficient distributed subgraph similarity matching," The VLDB Journal, vol. 24, no. 3, pp. 369–394, 2015.

[21] B. Schlegel, T. Kiefer, T. Kissinger, and W. Lehner, "Pcapriori: scalable apriori for multiprocessor systems," in Proceedings of the 25th International Conference on Scientific and Statistical Database Management. ACM, 2013, p. 20.
[22] Y. Djenouri, M. Comuzzi, and D. Djenouri, "SS-FIM: Single scan for frequent itemsets mining in transactional databases," in Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2017, pp. 644–654.
[23] P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and K. Thomas, "A survey of sequential pattern mining," Data Science and Pattern Recognition, vol. 1, no. 1, pp. 54–77, 2017.
[24] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, V. S. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," arXiv:1805.10511, 2018.
[25] L. Cerf, J. Besson, C. Robardet, and J.-F. Boulicaut, "Closed patterns meet n-ary relations," ACM Transactions on Knowledge Discovery from Data, vol. 3, no. 1, p. 3, 2009.
[26] C. K. S. Leung, M. A. F. Mateo, and D. A. Brajczuk, "A tree-based approach for frequent pattern mining from uncertain data," in Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2008, pp. 653–661.
[27] H. Huang, X. Wu, and R. Relue, "Association analysis with one scan of databases," in IEEE International Conference on Data Mining, 2002, pp. 630–632.
[28] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, F. Pulvirenti, and L. Venturini, "Frequent itemsets mining for big data: a comparative analysis," Big Data Research, vol. 9, pp. 67–83, 2017.
[29] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, and Y. Shi, "Parallel data mining techniques on graphics processing unit with compute unified device architecture (cuda)," The Journal of Supercomputing, vol. 64, no. 3, pp. 942–967, 2013.
[30] Y. Li, J. Xu, Y. H. Yuan, and L. Chen, "A new closed frequent itemset mining algorithm based on gpu and improved vertical structure," Concurrency and Computation: Practice and Experience, vol. 29, no. 6, 2017.
[31] X. Gu, Y. Zhu, S. Zhou, C. Wang, M. Qiu, and G. Wang, "A real-time fpga-based accelerator for ecg analysis and diagnosis using association-rule mining," ACM Transactions on Embedded Computing Systems, vol. 15, no. 2, p. 25, 2016.
[32] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in Big Data, 2013 IEEE international conference on. IEEE, 2013, pp. 111–118.
[33] N. Talukder and M. J. Zaki, "A distributed approach for graph mining in massive networks," Data Mining and Knowledge Discovery, vol. 30, no. 5, pp. 1024–1052, 2016.
[34] Y. Xun, J. Zhang, and X. Qin, "Fidoop: Parallel mining of frequent itemsets using mapreduce," IEEE transactions on Systems, Man, and Cybernetics: systems, vol. 46, no. 3, pp. 313–325, 2016.
[35] Y. Xun, J. Zhang, X. Qin, and X. Zhao, "Fidoop-dp: data partitioning in frequent itemset mining on hadoop clusters," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 1, pp. 101–114, 2017.
[36] M. Hegland, "The apriori algorithm–a tutorial," in Mathematics and computation in imaging science and information processing. World Scientific, 2007, pp. 209–262.
[37] Y. Djenouri, D. Djenouri, and Z. Habbas, "Intelligent mapping between gpu and cluster computing for discovering big association rules," Applied Soft Computing, vol. 65, pp. 387–399, 2018.
[38] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
[39] T. Friedrich, T. Kötzing, and M. Wagner, "A generic bet-and-run strategy for speeding up stochastic local search." in AAAI, 2017, pp. 801–807.
[40] 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.

YOUCEF DJENOURI obtained the PhD in Computer Engineering from the University of Science and Technology USTHB Algiers, Algeria, in 2014. From 2014 to 2015, Dr Djenouri was a permanent teacher-researcher at the university of Blida, in Algeria, where he is member of LRDSI Lab. He was granted a post-doctoral fellowship from the Unist university on South Korea, and he worked on BPM project supported by Unist university in 2016. In 2017, he was postdoctoral research at Southern Denmark University, where he has working on urban traffic data analysis. Currently, he is granted a post-doctoral fellowship from the European Research Consortium on Informatics and Mathematics (ERCIM), and he worked at the Norwegian University of Science and Technology (NTNU), in Trondheim, Norway. He is working on topics related to artificial intelligence and data mining, with focus on association rules mining, frequent itemsets mining, parallel computing, swarm and evolutionary algorithms and pruning association rules. Dr Youcef Djenouri participated in many international conferences worldwide, and he has been granted short-term research visitor internships to many renown universities including ENSMEA in Poitiers, university of Poitiers, and the university of Lorraine. He has published over 24 refereed conference papers, 20 international journal articles, 2 book chapter, and 1 tutorial paper in the areas of data mining, parallel computing and artificial intelligence. Current information can be found at https://sites.google.com/site/youcefdjenouri/.

DJAMEL DJENOURI Djamel Djenouri obtained the PhD in Computer Science from the University of Science and Technology (USTHB) Algiers, Algeria, in 2007. From 2008 to 2009, he has been granted a post-doctoral fellowship from the European Research Consortium on Informatics and Mathematics (ERCIM), and he worked at the Norwegian University of Science and Technology (NTNU), Norway. He is currently a senior research scientist (Director of Research) at the CERIST research center in Algiers, where he is leading the wireless sensor networking group, and adjunct full professor at the University of Blida. He is working on topics related to wireless networking, internet of things, mining of sensorial data for smart environment/city applications. He has been conducting several research projects with international collaborations on these topics, as the principal investor for many of these projects. He participated in many international conferences worldwide and gave many keynotes and plenary-session talks. He has been granted short-term research visitor internships to many renowned universities. He published more than 80 papers in international peer-reviewed journals and conference proceedings, and two books. He organised several editions workshops held in conjunction with DCOSS and GlobeCom, and he served as track chair for IEEE smart city conference, as a TPC member of many international conferences (e.g., GlobeCom, LCN, WiMob, etc.), guest editor of a special issue with Int. J. Communication Networks and Distributed Systems, and reviewer for many Journals. He is a senior member of the ACM, member of the Arab-German Academy of Sciences and Humanities (AGAVV), and the H2020 Algerian national contact point (NPC) for ICT.

JERRY CHUN-WEI LIN received his Ph.D. in Computer Science and Information Engineering from National Cheng Kung University, Tainan, Taiwan, in 2010. He is now working as an associate professor at Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway. His research interests include data mining, privacy-preserving and security, Big Data analytics, and social networks. He has published more than 200 research papers in peer-reviewed international conferences and journals, which have received more than 1900 citations. He is the co-leader of the popular SPMF open-source data-mining library and the Editor-in-Chief of *Data Mining and Pattern Recognition* (DSPR), Associate Editor of *Journal of Internet Technology* and *IEEE Access*, and a member of the Editorial Board of *Intelligent Data Analysis*.

ASMA BELHADI Asma Belhadi obtained the PhD in Computer Engineering from the University of Science and Technology USTHB Algiers, Algeria, in 2016. She is working on topics related to artificial intelligence and data mining, with focus on logic prpogramming. Dr Asma Belhadi participated in many international conferences worldwide, and she has been granted short-term research visitor internships to many renowned universities including IRIT in Toulouse. She has published over 10 refereed research articles in the areas of artificial intelligence