

An efficient frequent itemsets finding in distributed datasets with minimum communication overhead

Houda Essalmi, Anass El Affar

Laboratory of Engineering Sciences, Polydisciplinary Faculty of Taza, University of Sidi Mohamed Ben Abdellah, Fez, Morocco

Article Info

Article history:

Received Apr 13, 2024

Revised Oct 16, 2024

Accepted Oct 30, 2024

Keywords:

Apriori

Communication scheme

Computation costs

Distributed database

Generation of candidates

ABSTRACT

Finding frequent itemsets is an essential researched technique and a challenging task of data mining. Traditional approaches for distributed frequent itemsets require massive communication overhead among different distributed datasets. In this paper, we adopt a new strategy for optimizing the time of communications/synchronizations from large datasets and, we present a novel algorithm for discovering frequent itemsets in different distributed datasets on the slave sites called finding efficient distributed frequent itemsets (FEDFI). The proposed algorithm is capable of generating the important frequent itemsets by applying an efficient technique for pruning the candidate itemsets. The experimental results confirm that our algorithm FEDFI performs better than Apriori and candidate distribution (CD) algorithms in terms of communication and computation costs.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Houda Essalmi

Laboratory of Engineering Sciences, Polydisciplinary Faculty of Taza

University of Sidi Mohamed Ben Abdellah

Fez, Morocco

Email: houda.essalmi@usmba.ac.ma

1. INTRODUCTION

Data mining or KDD (knowledge discovery in databases) [1] is a discipline primarily concerned with the formulation, analysis, and implementation of processes that convert data into actionable knowledge [2]. Its core objective is to extract meaningful and interpretable information from databases [3]. The data utilized in data mining often presents challenges due to its large volume (numerous records) and high dimensionality (many attributes). As a result, there has been a growing focus on distributed data mining, which operates through distributed data analysis [4]. Distributed data mining employs a range of fast parallel and distributed techniques to uncover implicit and valuable knowledge [5], [6]. One of the most prominent techniques is association rule mining, which aims to identify significant relationships between attributes within the database. The generation of frequent itemsets is a fundamental and essential step in solving the association rule mining problem.

The Apriori algorithm, introduced by Agrawal [7], is a foundational and pioneering algorithm for mining frequent itemsets in transactional databases. The general concept of this algorithm involves iteratively scanning through itemsets level by level. At each level k , a set of candidate itemsets of size k is generated by joining the frequent itemsets identified in the previous iteration. This set of candidate itemsets is then pruned based on a statistical metric called support. Several variations of the Apriori algorithm are discussed in [8], [9] to mitigate its limitation of multiple scans over the entire transaction database, also the weaknesses of the original Apriori algorithm are cited in [10], highlighting its inefficiency in terms of time and space to search for frequent itemsets.

Many distributed algorithms have been developed for frequent itemset mining in distributed environments, designed to provide substantial computational power. These include algorithms such as CD (candidate distribution) [11], DMA (distributed mining algorithm) [12], FDM (fast distributed mining) [13], ODAM (optimized distributed association mining) [14], and DDM (distributed decision miner) [15]. These algorithms, which utilize the data parallelism paradigm, differ in their techniques for pruning new candidates or whether they employ candidate counting techniques.

In contrast, algorithms based on task parallelism, such as DD (data distribution) [11], IDD (intelligent data distribution) [16], HPA (hash-based parallel mining of association rules) [17], and others. They partition both the candidates and the database among processors. They differ in their methods for dividing candidates and the database. Some other algorithms cannot be classified into either of the two paradigms such as the CaD (candidate distribution) [11] algorithm, it represents a hybrid approach combining elements of both CD and DD. It focuses on distributing candidates to ensure consistency between the itemsets that are calculated and the transactions used for these calculations. By distributing transactions and candidates based on their prefixes, CaD enables each site to process data independently. In SH (skew handling) [18], candidate itemsets are not derived a priori from previous frequent itemsets. Instead, they are generated independently by each site by scanning its local partition of the database. The hybrid distribution (HD) algorithm [16] is a hybrid approach that integrates IDD and CD methods. The P sites are partitioned into groups of equal size, with each group functioning as a super-site. The IDD algorithm is employed within each group, while the CD algorithm is used for interactions between the groups. Zaki *et al.* [19] propose a set of algorithms (PAR-Eclat, PAR-MaxEclat, and PAR-MaxClique) that use database partitioning and calculate different candidate itemsets. The algorithms all assume that the database is in vertical format (tid-lists for each item), unlike horizontal partitioning. However, these algorithms encounter several challenges, including synchronization issues, communication overhead, load balancing problems, and a high frequency of data scans, which adversely affect their performance.

In the literature, many approaches focused on improving distributed algorithms for mining frequent itemsets in distributed environments. In this context, Mudumba and Kabir [20] proposed a novel approach for mining association rules independently from multiple data sources. This method integrates frequent patterns derived from each data source to identify frequent patterns applicable across a distributed environment. Furthermore, the model can be extended to generate rules with specified targets. The model improves transparency and memory efficiency in association rule generation. Additionally, it reveals significant relationships, allowing decision-makers to access frequent patterns from individual data sources as well as the overall dataset across the environment.

Nguyen *et al.* [21] presented three novel strategies designed to greatly improve the efficiency of multi-level high-utility itemset mining through multi-core processing. They also proposed two new algorithms, MCML+, and MCML++, which leverage these strategies. By fully utilizing the available processor cores, these approaches achieve superior performance in multi-level high utility itemset (HUI) mining.

Fernandez-Basso *et al.* [22] provide an overview of the most representative algorithms for association rule mining, particularly those designed for processing very large datasets. Recognizing the limitations of Hadoop compared to Spark, they developed new efficient frequent itemset algorithms for big data using distributed computation. Additionally, they introduced a new association rule-mining algorithm in Spark and compared these proposed algorithms with existing ones, varying the number of transactions and items.

Diop *et al.* [23] introduced the first pattern-on-demand approach for extracting patterns in a transactional distributed database. This approach involves extracting patterns instantly when the analyst requires them. They proposed a generic algorithm called distributed database sampling (DDSampling), which randomly selects a pattern from a distributed database based on an interestingness measure that combines frequency and length-based utility functions, including length constraints.

This study investigated the effects of mining frequent itemsets in distributed datasets with minimum communication overhead by developing a novel algorithm for frequent itemset mining, called FEDFI (finding efficient distributed frequent itemsets), with an appropriate communication model, referred to as master/slaves. While earlier studies have explored the impact of mining distributed frequent itemsets from the parallelization of mining algorithms [24], which involve performing distributed computations on a single database or intentionally distributing the data to expedite processing. This approach aims to address the challenges identified in existing distributed algorithms. The focus is on reducing communication and synchronization costs between different sites, which are critical for evaluating the performance of distributed algorithms. Furthermore, the work contributes to minimizing data scans by reducing the number of candidates generated for global information computation in a distributed setting. Our FEDFI algorithm employs three core techniques to facilitate the search for frequent itemsets: the first technique identifies the frequent 1-itemsets and 2-itemsets using a 2-dimensional matrix, while the remaining itemsets are derived

through Trie construction followed by a final validation of frequent itemsets. To evaluate the effectiveness of our approach, we compared our algorithm with the Apriori and CD algorithms in terms of communication and computation costs. The choice of the CD algorithm for comparison is due to its foundational role in distributed algorithms, it is a simple parallelization of Apriori. It operates in a distributed context where each site processes its local portion of the transaction database, calculates local supports, and shares them with other sites for global support computation.

This paper is structured as follows. Section 2 outlines the methodology of the work by presenting the adopted communication model and the proposed algorithm. The experimental results and a discussion of the performance of the FEDFI algorithm on various datasets are given in Section 3. Finally, section 4 concludes this paper.

2. METHOD

2.1. Model of communication

Some distributed algorithms for frequent itemset mining require each site S_i to broadcast the locally computed supports to all other sites at each iteration k , resulting in significant overhead [25]. The CD algorithm, being the foundational algorithm, consists of three phases. In phase 1, the local supports for the itemsets in each site's local partition are calculated. In phase 2, the sites synchronize and each site exchanges the local supports of all itemsets with all other sites to compute the global supports of the itemsets. In phase 3, the globally frequent itemsets are identified and generated independently at each site. The CD algorithm repeats until no more itemsets can be generated. At each iteration, the same set of frequent itemsets is identified, leading to redundant calculations of their global supports at each site. This results in an increased number of communications and/or synchronizations needed for global information computation in a distributed environment. To mitigate this issue, we propose adapting the master/slaves model to significantly reduce the number of communications and synchronizations between sites, thereby minimizing redundant calculations for candidate itemsets.

The master processor segments the entire database into clusters and assigns these clusters to slave processors, as detailed in [25]. Vasoya and Koli [26] demonstrated that implementing a master/slave system significantly improved time and space complexity, facilitating optimal resource utilization in a distributed environment. In our work, the database is horizontally partitioned among the slave sites, each of which handles the processing for its data segment. The master site coordinates between the slave sites, reducing the number of messages exchanged. During each iteration kk , the slave sites send messages to the master site, which then replies with messages to all slave sites. Figure 1 shows the master/slave communication model.

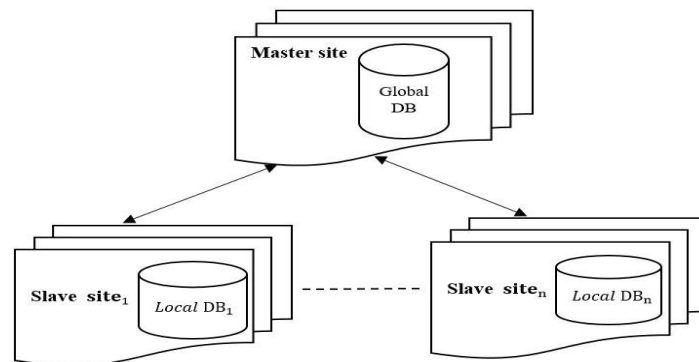


Figure 1. The model of master/slave communication

2.2. Proposed approach

The distribution aspect can be described as follows. Suppose that DB is a dataset of transactions partitioned horizontally and correspondently $\{DB_1, DB_2, \dots, DB_n\}$ on P sites $\{S_1, S_2, \dots, S_n\}$ in a distributed system. D is the total database size, and d_i is the size of each partition DB_i for $i=1, \dots, n$.

The support(X) is known as the global support counter in the entire DB and can be defined as,

$$X. sup = \frac{card(X)}{T_G}$$

where $card(X)$ is the number of records in the whole DB containing X and T_G is the total number of records in the database DB. The $Support_i(X)$ is known as the local support counter in DB_i and can be defined as,

$$X.Supp_i = \frac{card(X)}{T_i}$$

where $card(X)$ is the number of records in DB_i containing X and T_i is the total number of records in the database DB_i .

Let minimum support Sup_{min} defined by the user, X is globally frequent if $X.supp \geq Sup_{min} \times D$, correspondently, X is locally frequent at site S_i , if $X.supp_i \geq Sup_{min} \times d_i$.

Our FEDFI algorithm consists of two fundamental phases: the initialization phase and the execution phase. During the initialization phase, the master site partitions the horizontal database across all sites equitably by dividing the number of transactions by the number of sites. We are given a horizontal transaction database β illustrated below in Figure 2, where I as a group of items, the alphabet $I = \{a, b, c, d, e\}$ (with $m=5$ elements), and Tid as a set of transactions, $Tid = \{1, 2, 3, 4, 5, 6\}$ with a minimum support threshold ($Sup_{min} = 2$). The horizontal transaction database is fragmented and distributed on two slave sites as shown in Figure 3.

The execution phase begins with converting the horizontal database into a vertical data format for each slave site, generating a list of $Tids$ for all items in I . This method involves a single pass through the horizontal transaction database to identify the transaction $Tids$ for each item in I , which is then used to construct the vertical transaction database. The primary distinction between vertical and horizontal data lies in the structure of transactions. In horizontal data, a transaction includes the Tid and the Itemset, whereas in vertical data, each transaction is identified by the Itemset, enabling the support count for any frequent itemset to be calculated through the intersection of $TidSets$. This vertical transaction database comprises a list of all 1-itemsets and their corresponding $Tids$ for each slave site. Figure 4 illustrates the conversion of two slave sites in a vertical transaction database.

| Tid | Items |
|-----|-------|
| 1 | acd |
| 2 | bce |
| 3 | abce |
| 4 | be |
| 5 | abce |
| 6 | bce |

Figure 2. Horizontal transaction database β

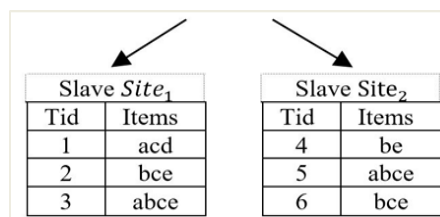


Figure 3. Horizontal transaction database for slave site₁ and slave site₂

| Vertical DB ₁ | | Vertical DB ₂ | |
|--------------------------|-------|--------------------------|-------|
| Items | TID | Items | TID |
| a | 1,3 | a | 5 |
| b | 2,3 | b | 4,5,6 |
| c | 1,2,3 | c | 5,6 |
| d | 1 | d | - |
| e | 2,3 | e | 4,5,6 |

Figure 4. Vertical database for slave site₁ and slave site₂

To efficiently extract all globally common itemsets in a distributed context, our FEDFI algorithm executes three techniques to avoid repeated database scans and reduce the cost associated with generating a large number of candidate sets, thereby minimizing execution time. The first technique establishes the CountTidList_t data structure (for t=1, .., n) for each slave site to facilitate the computation of frequent 1-itemsets and 2-itemsets, using a two-dimensional matrix of size (m×m), where m represents the number of items in the vertical transaction database. In the implementation, the itemsets are ordered lexicographically, and redundant (symmetric) elements are removed from the CountTidList_t matrix, thereby reducing its size to ½(m²+m). Furthermore, the CountTidList_t data structure is employed to optimize the support calculation of candidate itemsets without accessing the vertical database. The CountTidList_t matrix provides a projection of the vertical transaction database for each slave site.

Each cell (i, j) represents the size of Tids for the Itemset consisting of elements ‘x_i’ and ‘x_j’. The frequency of 1-itemsets is determined by the size of the Tid associated with each item; frequency (x_i)=size of (Tid (x_i)), and this is reflected in the diagonal cells of the matrix. The frequency of 2-itemsets is calculated based on the size of the intersection of Tids for the two items; frequency (x_ix_j) = size of (Tid (x_i) ∩ Tid (x_j)), and these values are located above the diagonal in the matrix. Figure 5 illustrates the construction of the CountTidList matrix for slave Site₁ and slave Site₂.

The exact supports for 1-itemsets and candidate 2-itemsets are determined through direct access to the CountTidList_t structure, and eliminate those that do not satisfy the minimum support threshold Sup_{min}. Only the frequent 1-itemsets and 2-itemsets are retained.

$$Support(x_i) = \frac{card(x_i)}{n} = \frac{CountTidList(i,1)}{n}, |x_i| = 1$$

$$Support(x_i x_j) = \frac{card(x_i x_j)}{n} = \frac{CountTidList(i,j-(i-1))}{n}, |x_i x_j| = 2, n = |\beta|$$

Significantly, frequent 1-itemsets and 2-itemsets for each slave site can be computed in a single pass through the vertical database, in contrast to the Apriori method, which requires two passes. Subsequently, each slave site transmits its data: support of frequent 1-itemset, 2-itemset and Tids of frequent 2-itemset: (Tid (x_ix_j)=Tid (x_i)∩Tid (x_j)) to the master site.

The master site then constructs a global structure CountTidList_G by summing the supports from the CountTidList_t of the different slave sites, without needing access to the vertical databases or additional communication with the slave sites. As shown in Figure 6, the total of the support values from the matrix of slave site₁ CountTidList₁ and Slave site₂ CountTidList₂. The list of 1-itemsets is {a:3, b:5, c:5, e:5}, and the list of 2-itemsets is {ab:2, ac:3, ae:2, bc:4, be:5, ce:4}. The list of Tid for 2-itemsets includes {ab:(3,5), ac:(1,3,5), ae:(3,5), bc:(2,3,5,6), be:(2,3,4,5,6), ce:(2,3,5,6)}.

The second technique of our algorithm aims to identify the set of K-itemsets (k≥3) within the slave site databases. To reduce the exchange of distributed calculations for frequent itemsets between these databases, the master site iteratively generates the list of k-itemsets (k≥3) by constructing a Trie. At the first level of the Trie, the nodes are composed of the set of frequent 2-itemsets obtained from the first technique, along with their TidSets, which are ordered lexicographically.

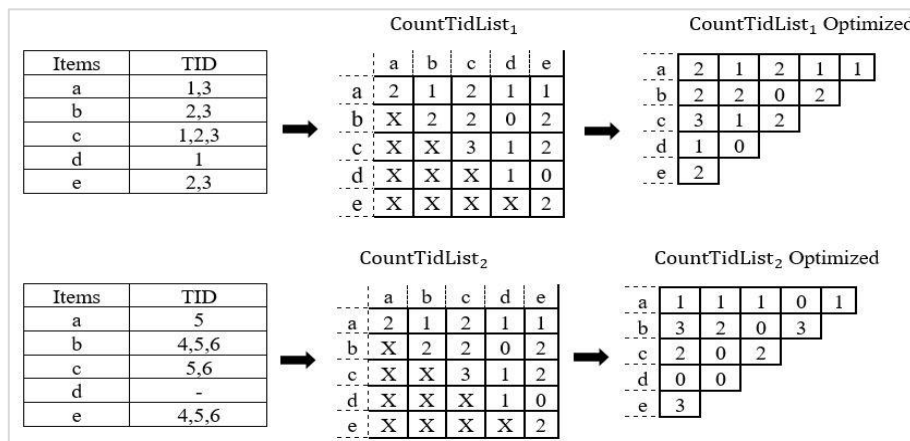


Figure 5. CountTidList matrix for slave site₁ and slave site₂

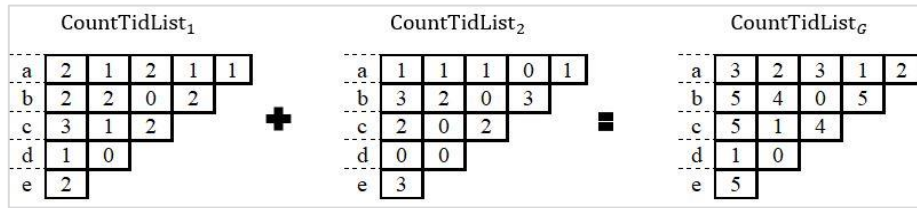


Figure 6. Calculate the Sum of CountTidList_G for slave site₁ and slave site₂

To generate candidate (k+1)-itemset nodes for the next level, the algorithm employs a self-join of the (k-1)-itemsets generated in the previous level, ensuring that (k-1) items of the two k-itemsets are identical. Additionally, the algorithm determines the Tids of (k+1)-itemsets by intersecting the Tids of the two k-itemsets that share the same (k-1) items. The algorithm continues to construct nodes at each level along with their Tids until no further nodes can be generated, thereby concluding the search for frequent k-itemsets with their TidSets using the Trie structure. Finally, the approximate support of the set of candidate k-itemsets is determined by counting the size of the TidSets for each node generated in the Trie structure.

In Figure 7, we illustrate the second technique using the previous example from database β. For instance, at level 1 of the Trie, the two nodes "ab" and "ac" have the same (k-1) item "a", so a link will be created to a node "abc" by identifying the TidSet through the intersection of their Tids (Tid(ab)∩Tid(ac)=35). This process continues up to level 3 (node "abce": frequent 4-itemset with Tid: 35). Since there is only one 4-item node, no more nodes can be generated. The approximate support of node "abc" is equal to the size of Tid (35): 2. The set of candidate k-itemsets generated, along with their approximate support, is as follows: {abc: 2, abe: 2, ace: 2, bce: 3, abce: 2}. This was derived from the Trie structure without any exchanges with slave site₁ and slave site₂.

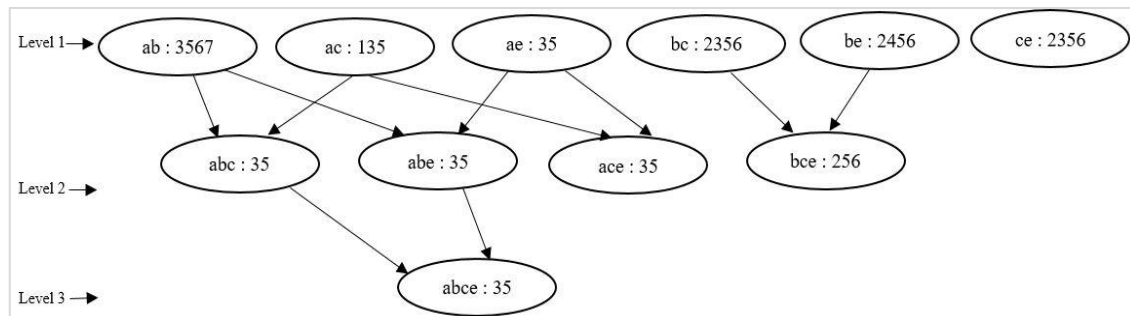


Figure 7. Illustration of Trie structure for k-itemsets (k ≥ 3)

The master site sends the set of candidate k-itemsets to the slave sites to determine the exact support for each k-itemset. Therefore, a third refinement technique is necessary to validate the support values. Each slave site must scan its local data to determine the support for each K-itemset. At this stage, The FEDFI algorithm determines the exact support of K-itemsets based on the SupportMin metric introduced in our paper [27]. This metric is used for pruning candidates without accessing the database. It estimates the actual support of itemset X based on the minimum supports of its (k-1) subsets identified in the previous iteration. This value represents the minimum possible support for a K-itemset X. At this point, the FEDFI algorithm aims to determine the exact support of the K-element sets by focusing on transactions that include the (k-1) subsets of the K-itemset that contain the minimum support. This method significantly reduces the number of transactions required to accurately determine the supports of the itemsets.

For example, at slave site₁, the support for the subsets of the itemsets "abc" is as follows: "ab": 1, "ac": 2, and "bc": 2, which have already been calculated using the CountTidList structure. Thus, the exact support of the itemset "abc" is determined within the transactions of the subsets that have the minimum support of "ab". The transactions for the itemset "ab" have been previously defined from the vertical database. Consequently, the exact support for "abc" is found in transaction Tid=3, consequently the exact Support of "abc" is equal to 1. Subsequently, the slave sites send to the master site the k-itemsets with their

exact support to calculate the global supports and identifies the frequent itemsets by discarding those that not satisfy the specified minimum support threshold (Sup_{min}).

It is widely recognized that the Apriori algorithm requires a comprehensive scan of all transactions in a database to mine frequent k -itemsets. In contrast, our refinement technique scans the database only for transactions associated with the subsets of an itemset that have the minimum support threshold. This method significantly reduces the number of database accesses required to identify frequent k -itemsets.

The proposed method is illustrated in Figure 8, which details the methodology employed by the FEDFI algorithm and its key components. The master site is responsible for partitioning and distributing the database horizontally and equitably across all sites, as well as converting each local database to a vertical format. This work is achieved by applying three major techniques for mining distributed frequent itemsets: generating the CountTidList matrix structure for each Slave site and computing the global support for 1-itemsets and 2-itemsets at the master site. The set of candidate k -itemsets ($k \geq 3$) and their supports are determined using the Trie structure. The master site then distributes the candidates to the slave sites for further refinement of the k -itemsets ($k \geq 3$). Based on the results provided by the slave sites, the master site verifies whether the k -itemsets are globally frequent and displays the results for all frequent k -itemsets.

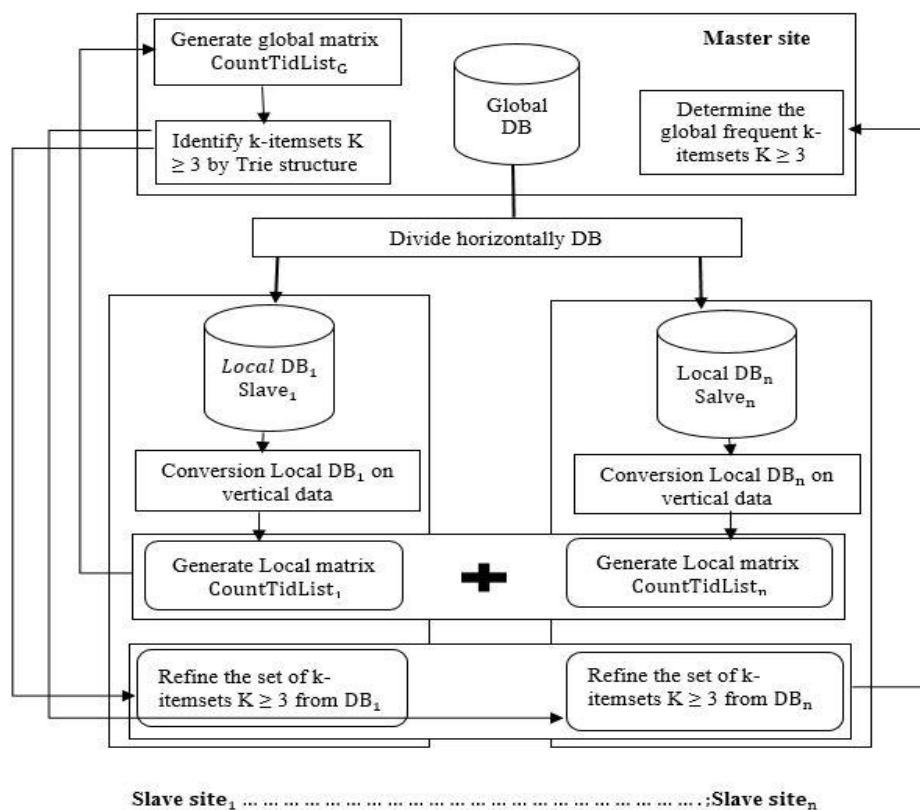


Figure 8. The main process of the proposed FEDFI

In this method, our FEDFI algorithm develops a master/slave (sites) communication model by partitioning the database across all sites in an equitable manner, dividing the number of transactions by the number of sites. Conversely, in the basic CD data parallelism algorithm and the basic DD task parallelism algorithm, partitioning is based on a random distribution of data by performing a horizontal division of the data (using an "all-to-all" broadcast approach). However, it suffers from generating a large number of candidates, leading to increased communication overhead.

In the CD algorithm, sites synchronize after computing the local support of candidates, and each site exchanges the local support of all candidates with all other sites to calculate the global support. In the DD algorithm, each site processes the received partitions to obtain global support for the entire database. After that, each site calculates the frequent itemsets from its candidate set, exchanges these frequent itemsets with all other sites to obtain the complete set of frequent itemsets, and then generates the new candidates,

partitions them, and distributes them across all sites. Consequently, our model greatly reduces the number of communications necessary for the distributed computation of global frequent itemsets compared to the conventional parallel/distributed algorithms previously described.

The CD algorithm focuses on the Apriori algorithm to extract frequent k-itemsets. By applying the Apriori algorithm to our master/slaves model with the example above, we find the following results:

- In the first iteration, both slave sites calculate the local supports of the candidate 1-itemsets ($C1(s_1)=\{a,b,c,d,e\}$, $C1(s_2)=\{a,b,c,d,e\}$) and send these calculations to the Master site.
- The master site merges the received candidates ($C1(s_1)$ and $C1(s_2)$), determines the frequent 1-Itemsets F1, computes the candidates C2, and sends them back to slave sites 1 and 2.
- This process is repeated for iterations 2, 3, and 4 with the respective candidates $C2=\{ab, ac, ae, bc, be, ce\}$, $C3=\{abc, abe, ace, bce\}$, and $C4=\{abce\}$.

The Apriori algorithm thus performs four iterations to calculate frequent itemsets, resulting in four phases of communication between the master site and the slave sites. In contrast, our algorithm requires only two accesses to the local database in the first technique of the execution phase to obtain the frequent 1-itemsets and 2-itemsets and in the third technique during the refinement phase to validate the supports of the k-itemsets ($k \geq 3$). Therefore, our FEDFI algorithm requires only two exchanges between the master site and the slave sites to compute all the frequent itemsets.

3. RESULTS AND DISCUSSION

This section presents both the experimental results and a discussion of the performance of the FEDFI algorithm, benchmarked against Apriori and CD algorithms. We evaluated the algorithms using two datasets: T40110D100K and Chess. The experiments were conducted in a distributed environment using a master/slaves scheme, where datasets were partitioned horizontally across multiple slave sites. The primary objective is to assess the efficiency, scalability, and communication overhead associated with the FEDFI algorithm, and how these aspects compare to those of the Apriori and CD algorithms.

3.1. Dataset overview

The datasets used were selected to provide a range of complexities:

- T40110D100K: sourced from FIMI [28] and studied by Fournier-Viger *et al.* [29], this large-scale dataset contains 1,000 distinct items and 100,000 transactions. It presents a high-dimensional challenge that tests the scalability and communication efficiency of mining frequent itemset algorithms.
- Chess: also from FIMI [28], this dataset, explored by Fournier-Viger *et al.* [29], consists of 75 items and 3196 transactions. It offers a more structured environment for assessing algorithm performance in scenarios with fewer transactions and less data complexity.

3.2. Experimental setup

The experiments were conducted on a system with an Intel® Core™ i7 processor at 2.80 GHz, 4 GB of RAM, and Windows 10. We distributed the data sets to 3, 5, and 7 slave nodes (sites) to measure scalability and efficiency. The algorithms were implemented in Java using the NetBeans IDE. The Apriori and FEDFI algorithms use a bidirectional communication scheme between sites following the master/slaves model, whereas the CD algorithm employs the classic models of usual communication between sites.

3.3. Results

3.3.1. Execution time analysis

The performance of the FEDFI algorithm was compared to that of the Apriori and CD algorithms in terms of execution time. Figures 7 and 8 present the execution time as a function of the minimum support threshold for Chess and T40110D100K datasets, respectively.

- Chess dataset: the results for the Chess dataset, depicted in Figure 9, also indicate that FEDFI outperforms the other algorithms. Even in a smaller dataset with fewer transactions, the algorithm's efficiency is evident. For instance, at a support threshold of 10%, FEDFI required approximately 30% fewer iterations than Apriori and 35% fewer than CD. This reduction in the number of iterations directly translates to fewer communication phases, underscoring FEDFI's ability to scale efficiently even in less complex environments.
- T40110D100K dataset: as shown in Figure 10, the FEDFI algorithm consistently outperformed both Apriori and CD across all support thresholds. For example, at a support threshold of 50%, FEDFI completed the mining process approximately 20% faster than Apriori and 25% faster than CD. The primary reason for this superior performance is FEDFI's ability to generate fewer candidate itemsets, which reduces the number of communication rounds between slave nodes. This reduction in

communication significantly lowers the overall execution time, especially for larger datasets where communication cost can be a major bottleneck.

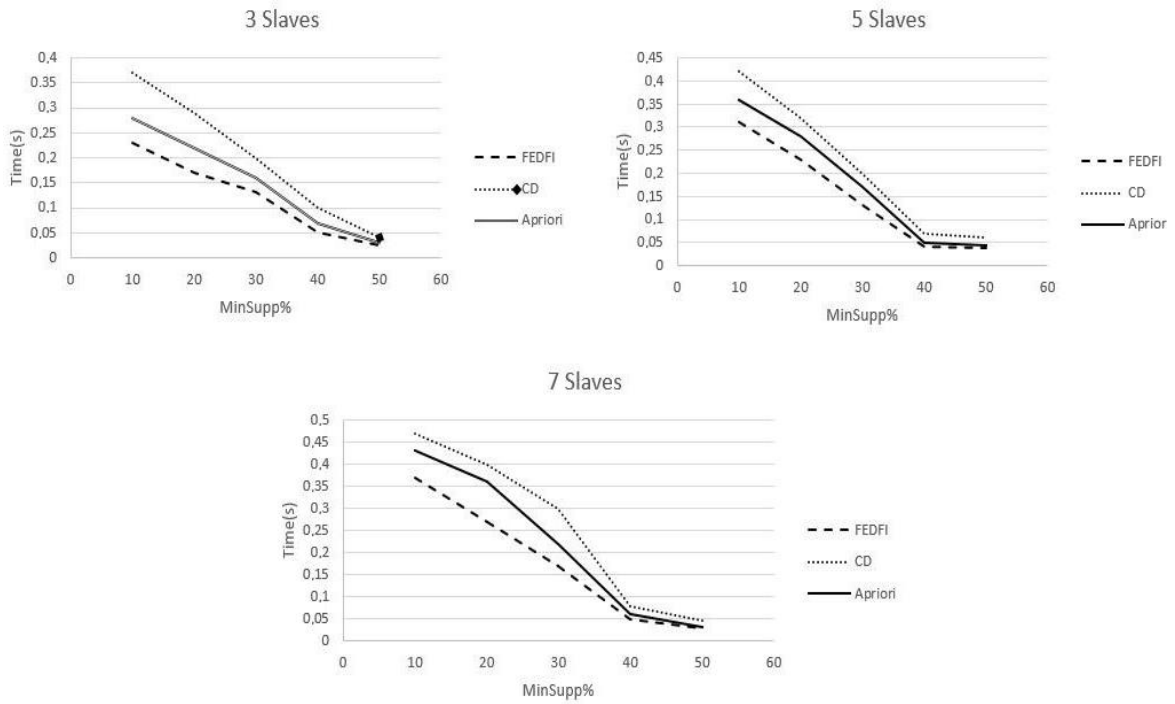


Figure 9. Comparison of runtime for Chess datasets

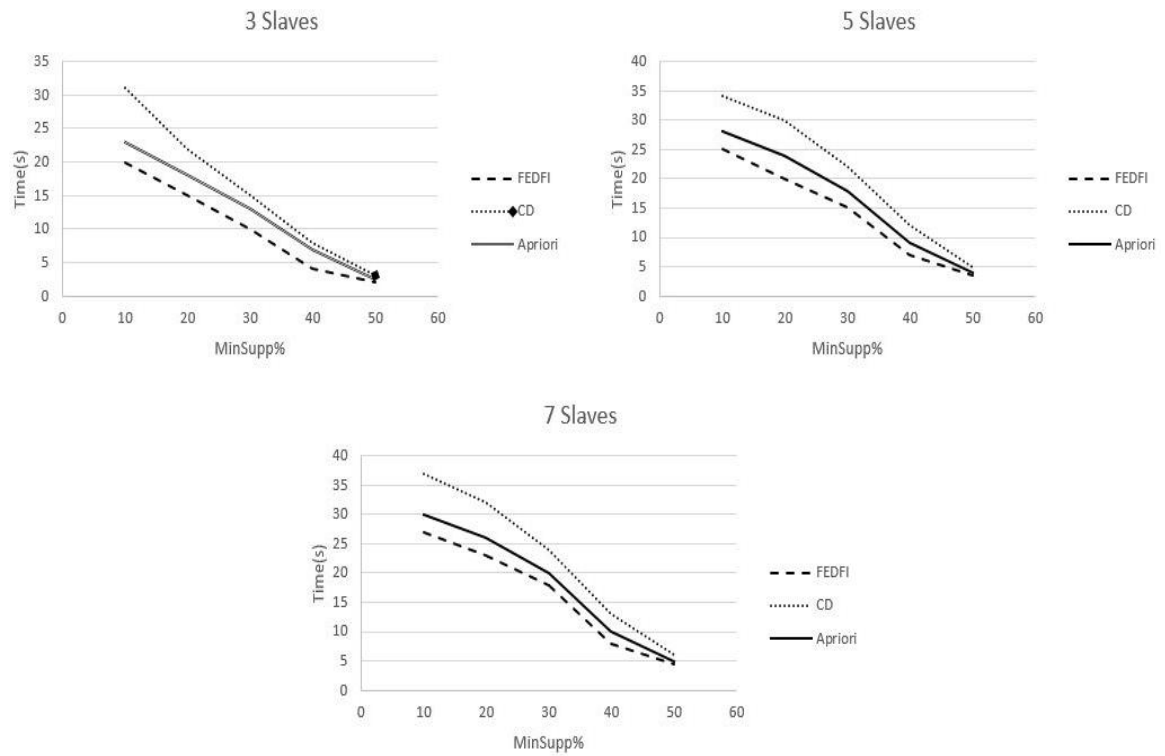


Figure 10. Comparison of runtime for T40110D100K datasets

3.3.2. Scalability analysis

To evaluate the scalability of the FEDFI algorithm, we tested it with an increasing number of slave nodes (3, 5, and 7) and larger dataset sizes. Figure 11 illustrates how the performance of FEDFI scales with more slave nodes.

- Impact of node increase: as we increased the number of slave nodes, the FEDFI algorithm continued to outperform Apriori and CD. At 7 nodes, FEDFI still exhibited better execution times, while Apriori and CD showed more noticeable increases in execution time due to higher communication overhead. FEDFI’s ability to minimize communication phases, even as the number of nodes rises, is a critical factor in its superior scalability.
- Convergence at high node count: interestingly, as the number of nodes increased, we observed that the performance of all algorithms began to diverge. This is particularly noticeable at 7 nodes, where the communication overhead became a dominant factor. However, even at this level of parallelism, FEDFI maintained an edge over Apriori and CD, reflecting its more efficient data distribution and reduced need for communication rounds.

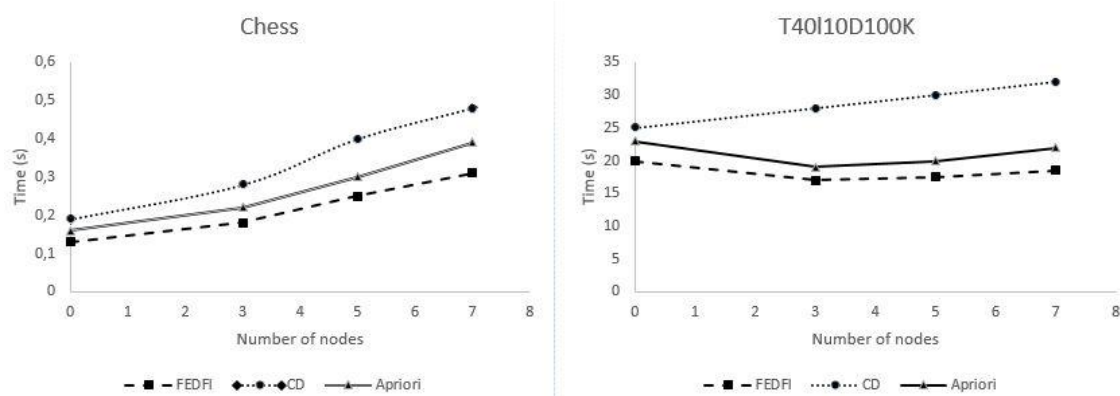


Figure 11. Scalability of FEDFI by number of nodes with $Sup_{min}=20\%$

3.4. Discussion

The results of this study highlight several important insights about the FEDFI algorithm and its performance in a distributed context. Efficiency gains: FEDFI’s efficiency in generating fewer candidate itemsets directly leads to faster execution times and reduced communication overhead. This is particularly evident in the T40I10D100K dataset, where FEDFI consistently outperformed both Apriori and CD. The ability to minimize communication rounds is a significant advantage in distributed systems, where communication costs can quickly escalate. Superior scalability: the scalability of FEDFI is one of its standout features. As the dataset size and the number of slave nodes increased, FEDFI maintained superior performance compared to Apriori and CD. This scalability is crucial for real-world applications, particularly in big data environments where datasets are often distributed across multiple sites. The algorithm’s ability to handle larger datasets without a significant increase in communication cost is a major strength. Comparison with related research: our results align with findings from prior studies, which have noted the limitations of Apriori in distributed settings due to managing a large volume of candidate sets, particularly when dealing with numerous frequent itemsets, low minimum support thresholds, or extensive itemsets. Apriori’s performance drastically decreases and becomes inefficient when the memory capacity is constrained and the transaction count is high. CD algorithm uses a simple communication system to exchange local supports, relying on a broadcast all-to-all communication model. However, it suffers from generating a large number of candidate sets, resulting in increased communication overhead. The performance of count distribution is limited, similar to Apriori, as this algorithm is based on the frequent itemset mining method. Consequently, it generates a substantial number of candidate sets, much like those produced by Apriori. The proposed method FEDFI addresses these limitations by reducing the number of candidate itemsets and communication phases required to generate frequent itemsets. This is especially critical for large datasets where communication costs can become a major bottleneck. The reduction in communication rounds observed in our experiments mirrors similar improvements reported in other research on distributed frequent itemset mining.

Implications of findings: the superior performance of FEDFI suggests that it is a viable alternative to Apriori and CD in distributed environments. Its ability to minimize both execution time and communication

overhead makes it particularly suitable for large-scale, real-world applications. The scalability demonstrated by FEDFI also indicates that it can be effectively used in environments with a high degree of parallelism, where data is distributed across numerous sites. The experimental results strongly support the conclusion that FEDFI offers substantial improvements in both efficiency and scalability compared to the traditional Apriori and CD algorithms. Its ability to reduce communication phases while maintaining high performance across varying dataset sizes and numbers of slave nodes positions it as a highly effective solution for mining distributed frequent itemsets in large-scale data environments.

4. CONCLUSION

This research introduces a new algorithm, FEDFI, for the efficient extraction of frequent itemsets in a distributed context. The proposed method in this study employs a master/slave communication model, distinguishing between a master site and multiple slave sites. We found that the model significantly reduces the communication and synchronization costs required for distributed global frequent itemset computation compared to the communication schemes (such as broadcasting) used in previously presented parallel/distributed algorithms. The FEDFI algorithm relies on three techniques to extract interesting frequent k-itemsets. In the first step, each Slave site identifies all frequent 1-itemsets and 2-itemsets using a matrix derived from a vertical database. In the second technique, candidate k-itemsets and their approximate supports are discovered using a Trie structure, built from nodes representing the frequent 2-itemsets sent by slave sites to the master site. The final step is necessary to refine the approximate supports of the frequent k-itemsets by accessing transactions of (k-1) subsets itemset that contain the minimum support threshold within the local database.

Our study demonstrates that our FEDFI algorithm is more resilient than the Apriori and CD algorithms and confirms the main role of using the master site with the proposed technique for effective validation of results at each iteration of obtaining frequent k-itemsets. Our findings provide conclusive evidence that this approach is more efficient and effective than other distributed algorithms Apriori and CD regarding the number of candidate itemsets generated where data is distributed across different sites by reducing communication and synchronization costs between sites. Our approach is valuable for businesses dealing with vast amounts of data, often spread across various media. It facilitates the discovery of new relationships from the accumulated data, enabling the extraction of hidden, valuable insights within distributed databases. Future studies may explore the problem of mining distributed association rules within a distributed context. The goal is to propose new solutions and design a novel algorithm that is well-adapted to distributed environments based on our FEDFI algorithm.




REFERENCES

- [1] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, "Knowledge discovery in databases: an overview," *AI magazine*, vol. 13, no. 3, p. 57, 1992, doi: 10.1609/aimag.v13i3.1011.
- [2] E. Shakhshuki, "A methodology for evaluating agent toolkits," in *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, 2005, pp. 391-396 Vol. 1, doi: 10.1109/ITCC.2005.15.
- [3] S. Suba and T. Christopher, "A study on milestones of association rule mining algorithms in large databases," *International Journal of Computer Applications*, vol. 47, no. 3, pp. 12-19, Jun. 2012, doi: 10.5120/7167-9674.
- [4] H. Kargupta, C. Kamath, and P. Chan, "Distributed and parallel data mining: emergence, growth, and future directions," *Advances in Distributed and Parallel Knowledge Discovery*, pp. 409-416, 2000.
- [5] A. Cuzzocrea, "Models and algorithms for high-performance distributed data mining," *Journal of Parallel and Distributed Computing*, vol. 73, no. 3, pp. 281-283, Mar. 2013, doi: 10.1016/j.jpdc.2012.11.002.
- [6] Y. Fu, "Distributed data mining: an overview," *Newsletter of the IEEE Technical Committee on Distributed Processing*, vol. 4, no. 3, pp. 5-9, 2001.
- [7] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 207-216, Jun. 1993, doi: 10.1145/170036.170072.
- [8] J. Han, M. Kamber, and J. Pei, "Data mining: concepts, models and techniques," *Choice Reviews Online*, vol. 49, no. 04, p. 2107, Dec. 2011, doi: 10.5860/choice.49-2107.
- [9] Rupali and G. Gupta, "Apriori based algorithms and their comparisons," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 7, pp. 79-84, 2013.
- [10] A. Bhandari, A. Gupta, and D. Das, "Improvised Apriori algorithm using frequent pattern tree for real time applications in data mining," *Procedia Computer Science*, vol. 46, pp. 644-651, 2015, doi: 10.1016/j.procs.2015.02.115.
- [11] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962-969, 1996, doi: 10.1109/69.553164.
- [12] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. Fu, "Efficient mining of association rules in distributed databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 911-922, 1996, doi: 10.1109/69.553158.
- [13] D. W. Cheung, Jiawei Han, V. T. Ng, A. W. Fu, and Yongjian Fu, "A fast distributed algorithm for mining association rules," in *Fourth International Conference on Parallel and Distributed Information Systems*, 1996, pp. 31-42, doi: 10.1109/PDIS.1996.568665.
- [14] M. Z. Ashrafi, D. Taniar, and K. Smith, "ODAM: an optimized distributed association rule mining algorithm," *IEEE Distributed Systems Online*, vol. 5, no. 3, pp. 1-18, Mar. 2004, doi: 10.1109/MDSO.2004.1285877.




- [15] A. Schuster and R. Wolff, "Communication-efficient distributed mining of association rules," *ACM SIGMOD Record*, vol. 30, no. 2, pp. 473–484, Jun. 2001, doi: 10.1145/376284.375728.
- [16] E.-H. Han, G. Karypis, and V. Kumar, "Scalable parallel data mining for association rules," *ACM SIGMOD Record*, vol. 26, no. 2, pp. 277–288, Jun. 1997, doi: 10.1145/253262.253330.
- [17] T. Shintani and M. Kitsuregawa, "Hash based parallel algorithms for mining association rules," in *Fourth International Conference on Parallel and Distributed Information Systems*, 1996, pp. 19–30, doi: 10.1109/pdis.1996.568664.
- [18] L. Harada, N. Akaboshi, K. Ogihara, and R. Take, "Dynamic skew handling in parallel mining of association rules," in *Proceedings of the seventh international conference on Information and knowledge management*, Nov. 1998, pp. 76–85, doi: 10.1145/288627.288634.
- [19] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," in *Proceedings - 3rd International Conference on Knowledge Discovery and Data Mining, KDD 1997*, 1997, pp. 283–287.
- [20] B. Mudumba and M. F. Kabir, "Mine-first association rule mining: an integration of independent frequent patterns in distributed environments," *Decision Analytics Journal*, vol. 10, p. 100434, Mar. 2024, doi: 10.1016/j.dajour.2024.100434.
- [21] T. D. D. Nguyen, N. T. Tung, T. Pham, and L. T. T. Nguyen, "Parallel approaches to extract multi-level high utility itemsets from hierarchical transaction databases," *Knowledge-Based Systems*, vol. 276, p. 110733, Sep. 2023, doi: 10.1016/j.knsys.2023.110733.
- [22] C. Fernandez-Basso, M. D. Ruiz, and M. J. Martin-Bautista, "New Spark solutions for distributed frequent itemset and association rule mining algorithms," *Cluster Computing*, vol. 27, no. 2, pp. 1217–1234, Apr. 2024, doi: 10.1007/s10586-023-04014-w.
- [23] L. Diop, C. T. Diop, A. Giacometti, and A. Soulet, "Pattern on demand in transactional distributed databases," *Information Systems*, vol. 104, p. 101908, Feb. 2022, doi: 10.1016/j.is.2021.101908.
- [24] M. J. Zaki, "Parallel and distributed association mining: a survey," *IEEE Concurrency*, vol. 7, no. 4, pp. 14–25, Oct. 1999, doi: 10.1109/4434.806975.
- [25] T. Tassa, "Secure Mining of association rules in horizontally distributed databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 970–983, Apr. 2014, doi: 10.1109/TKDE.2013.41.
- [26] A. Vasoya and N. Koli, "Mining of association rules on large database using distributed and parallel computing," *Procedia Computer Science*, vol. 79, pp. 221–230, 2016, doi: 10.1016/j.procs.2016.03.029.
- [27] H. Essalmi, M. El Far, M. El Mohajir, and M. Chahhou, "A novel approach for mining frequent itemsets: AprioriMin," in *2016 4th IEEE International Colloquium on Information Science and Technology (CISIT)*, Oct. 2016, pp. 286–289, doi: 10.1109/CISIT.2016.7805057.
- [28] B. Goethals and M. J. Zaki, "Advances in frequent itemset mining implementations," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 109–117, Jun. 2004, doi: 10.1145/1007730.1007744.
- [29] P. Fournier-Viger *et al.*, "The SPMF open-source data mining library version 2," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9853 LNCS, pp. 36–40, doi: 10.1007/978-3-319-46131-1_8.

BIOGRAPHIES OF AUTHORS



Houda Essalmi    received her Master degree in Computer Science and Engineering from the University of Abdelmalek Essaâdi Faculty of Science Tétouan Morocco. She is currently a Ph.D. student; focused on data mining approaches at University Sidi Mohamed Ben Abdellah (Polydisciplinary Faculty of Taza, Fez, Morocco). Her research interests include big data, data mining, artificial intelligence, and machine learning. She can be contacted at email: houda.essalmi@usmba.ac.ma.



Anass El affar    received his Ph.D. degree in Computer Science and Image Processing from Sidi Mohamed Ben Abdellah University (Fez, Morocco) in 2009, prior to joining the Department of Computer Science of Sidi Mohamed Ben Abdellah University (Polydisciplinary Faculty of Taza, Morocco), he worked as a professor at Department of Computer Science in Sidi Mohamed Ben Abdellah University. His current research interests are image processing, big data, e-learning, and machine learning. He participated in and coordinated several research projects. He can be contacted at email: anass.elaffar@usmba.ac.ma.