Parallel and Distributed Multi-level Entropy-Based Approach for Adaptive Global Frequent Pattern Mining in Large Datasets

Original Scientific Paper

Houda Essalmi*
Laboratory of Engineering Sciences, Polydisciplinary Faculty of Taza,
University of Sidi Mohamed Ben Abdellah
Fez, Morocco
houda.essalmi@usmba.ac.ma

Anass El Affar
Laboratory of Engineering Sciences, Polydisciplinary Faculty of Taza,
University of Sidi Mohamed Ben Abdellah
Fez, Morocco
anass.elaffar@usmba.ac.ma

*Corresponding author

Abstract *– Frequent pattern mining in distributed settings remains a significant challenge due to predominantly high computational expenses and high communication overhead. This paper presents AGFPM (Adaptive Global Frequent Pattern Mining), a novel solution that integrates an extensible Master-Slave architecture with an advanced pruning technique that relies on binary entropy and statistical quartiles. AGFPM proposes two primary data structures: the LP-Tree (Local Prefix Tree) and the GP-Tree (Global Prefix Tree). A single pass of each local Slave site is used to build one LP-Tree, and low information value branches are pruned early on by entropy and quartile thresholds. Rather than transferring complete trees, only succinct metadata is sent to the Master site, where the GP-Tree is built from globally sorted items in order of their entropy rankings. A significant aspect of AGFPM is the flexible pruning approach: either the GP-Tree is pruned or not pruned, based on user criteria. This provides a dynamic adjustment between the performance and generality of results, thereby allowing control over the level of compression applied when generating global patterns. Global frequent patterns are then recursively mined from the GP-Tree based on conditional sub-GP-Trees. Frequent patterns are extended at each level of the hierarchy by intersecting the common prefix paths, guided by a Global Header Table. AGFM demonstrates improved performance in execution time, scalability, and robustness against low support thresholds relative to existing methods.*

Keywords: *Data mining, Distributed Datasets, FP-tree; Communication Overhead, Frequent patterns mining, Binary Entropy, Quartile-based Pruning*

Received: July 2, 2025; Received in revised form: August 30, 2025; Accepted: September 1, 2025

1. INTRODUCTION

1.1. BACKGROUND AND CHALLENGES

Frequent pattern mining over distributed, horizontally partitioned data aims to uncover meaningful co-occurrences while keeping computation and communication overheads manageable [1-3]. Candidate-generation approaches rooted in the Apriori principle typically face a rapid growth in candidates and require multiple scans of the data. Pattern-growth methods mitigate this by compressing transactions into compact prefix structures, yet they can still encounter memory pressure and expensive cross-site consolidation as datasets become large or dense [4-7]. In practice, an effective solution should combine compact and merge-friendly structures, a coherent global ordering that simplifies fusion, limited network traffic, and principled filtering to constrain structural growth without discarding informative patterns.

1.2. LIMITS OF PRIOR APPROACHES

Algorithms in the Apriori family rely on an iterative process of candidate generation and support validation, guided by the downward closure property [8]. While effective, this approach necessitates multiple database

scans and extensive synchronization, leading to scalability challenges in distributed environments [8-12]. Several distributed variants, such as Count Distribution (CD) [9], Distributed Mining Algorithm (DMA) [10], and Fast Distributed Mining (FDM) [11], operate by aggregating local counts across nodes, but often suffer from candidate set explosion and high communication overhead, especially at low minimum support thresholds. Enhanced methods like Optimized Distributed Association Mining (ODAM) [12] and Distributed Decision Miner (DDM) [13] aim to mitigate these issues by reducing inter-site message exchange and memory consumption. To improve efficiency, parallelization strategies leverage data or task decomposition, including Data Distribution (DD) [9], Intelligent Data Distribution (IDD) [14], and Hash-based Parallel Association (HPA) [15], which distribute workloads to increase throughput. Techniques such as Scalable Hybrid (SH) [16] address load imbalance by adapting to data skew. Hybrid approaches like Candidate Distribution (CaD) [9] and Hybrid Distribution (HD) [14] further combine partitioning strategies using prefix-based assignment or optimized memory-aware distribution to minimize communication and enhance locality. Despite these optimizations, the fundamental limitations of repeated data scans and coordination between nodes continue to hinder performance in large-scale or dense datasets.

FP-growth avoids explicit candidate generation by organizing transactions into a compact prefix tree and mining conditional pattern bases. Parallel variants such as PFP-tree and Multiple Local Parallel Trees (MLPT) construct local trees across sites and merge them subsequently, alleviating candidate-generation costs but introducing heavy cross-site transfers and structural heterogeneity that complicate global consolidation [17, 18]. Single-pass and compact prefix structures reduce scans and memory requirements; however, harmonizing many sizable local trees under constrained memory and bandwidth remains challenging [4-6].

In dense settings and at low support, both candidate-generation and pattern-growth families face similar stressors: intermediate structures can grow quickly (large candidate sets or deep/wide trees), local structures diverge across sites and become difficult to align, and communication scales with the number of sites, especially during counting and merging [4–16]. Because control is predominantly frequency-based, branches with limited informational value often persist, inflating consolidation costs and post-processing effort. These observations point to the need for mechanisms that are explicitly information-aware and distribution-adaptive, so that uninformative regions are pruned early, local structures are better aligned before fusion, and global aggregation is streamlined.

1.3. MOTIVATION: BINARY ENTROPY WITH QUARTILE-BASED PRUNING

Fixed frequency thresholds (e.g., MinSupp) do not always provide sufficient control over structural growth and redundancy in distributed trees. Binary Shannon entropy offers an information-theoretic signal of presence/absence uncertainty for items and paths, distinguishing informative from low-value branches [19, 20]. Using the empirical distribution of entropies, adaptive quartile thresholds (Q1, Q2, Q3) enable progressive pruning: rapid removal of minimally informative branches (below Q1), complementary reduction around the median (Q2), and preservation of the most information-rich parts (above Q3) [21, 22]. This data-adaptive strategy avoids arbitrary hyperparameters and can be applied locally to shrink LP-Trees prior to fusion and, optionally, globally to regulate GP-Tree complexity.

1.4. CONTRIBUTIONS OF AGFPM

This work presents AGFPM (Adaptive Global Frequent Pattern Mining), a distributed method built around four complementary pillars. First, it adopts a Master–Slave architecture with lightweight communication: each site constructs a (Local Prefix-Tree) LP-Tree and shares only compact metadata with the Master, thereby avoiding full-tree transfers and reducing network overhead. Second, it imposes a global, information-driven ordering: items are ranked by global entropy, and sites rebuild their LP-Trees accordingly, yielding structurally aligned trees that facilitate accurate and efficient fusion. Third, it implements multi-level adaptive pruning: branches are filtered using binary entropy and quartile thresholds (Q1, Q2, Q3), applied locally to the (Global Prefix-Tree) GP-Tree, in order to contain structural growth, reduce redundancy, and preserve the most informative paths. Fourth, it enables efficient global mining: a GP-Tree, guided by a Global Header Table and the global entropy order, supports recursive, top-down extraction via conditional sub-GP-Trees. Collectively, these design choices shorten runtime and communication, improve robustness under low support on dense datasets, and produce more focused sets of global patterns without compromising relevance findings corroborated on standard synthetic and real benchmarks [23-25].

Positioned against prior work, AGFPM extends beyond Apriori-style distributed families [8–16] by dispensing with explicit candidate generation and limiting synchronization, and beyond parallel pattern-growth approaches [4-7, 17-18] by transmitting succinct metadata and enforcing a global entropy-based order that homogenizes LP-Trees before fusion. In addition, AGFPM introduces a multi-level, quartile-guided pruning strategy applied locally and, when desired, to the GP-Tree to bound structural complexity and curb redundancy. Relative to recent distributed big-data systems [26-31], the combination of information-driven ordering and adaptive pruning jointly lowers communication overhead and runtime while strengthening robustness at low support on dense data.

The remainder of this paper is structured as follows: Section 2 presents related work. Section 3 discusses the problem definition of mining frequent patterns in

parallel and distributed contexts and presents the concept of Entropy and Quartiles. The proposed algorithm is given in detail in Section 4. The results and discussion are presented in Section 5. Finally, Section 5 draws conclusions from this work.

2. RELATED WORK

Several studies have been suggested to improve the way pattern mining algorithms process large data. We discuss important contributions in distributed frequent pattern mining in this section.

Deng *et al*. [26] proposed an optimized and distributed version of the Apriori algorithm, titled STB_Apriori, designed to operate on the Apache Spark platform. This approach stands out due to the use of BitSet structures, which allow transactions to be represented in a compressed and efficient manner, significantly reducing the memory required for processing. By fully leveraging Spark's in-memory computing model, the algorithm manages to limit disk accesses while benefiting from distributed parallelism across data partitions. The authors demonstrated, through experiments on large datasets, that STB_Apriori outperforms the classic versions of Apriori, both in terms of execution time and scalability. These results confirm the value of combining memory optimization techniques with distributed frameworks to improve the extraction of frequent itemsets in large-scale environments.

Shaikh *et al*. [27] developed DIAFM (Distributed Incremental Approximation Frequent Itemset Mining), a distributed approach designed for the incremental extraction of frequent itemsets at a large scale. This algorithm relies on the MapReduce model and introduces a strategy of successive fragment processing, allowing for the efficient integration of new data without re-scanning the entire dataset. DIAFM aims to reduce the overall computational cost while maintaining high accuracy, thanks to a controlled error tolerance mechanism. Experimental evaluations show that this method significantly improves processing time while adapting to dynamic environments such as transactional streams or continuous monitoring systems. Its modular architecture and compatibility with increasing data volumes make it a promising solution for incremental exploration in distributed contexts.

Sun *et al*. [28] developed a distributed basket analysis system for large transaction volumes, leveraging the parallel processing capabilities of Apache Spark. This approach relies on a two-step execution. First, random portions of the data are analyzed locally using FP-Growth to identify partial frequent patterns. In a second step, these results are consolidated using an approximate method to construct a global set of representative itemsets. The system stands out for its flexibility, particularly due to the integration of Spark SQL, which facilitates the analytical querying of the extracted results. Experiments conducted on massive datasets (up to one billion transactions) demonstrate a significant reduction in execution time and excellent scalability, while maintaining accuracy close to that of exact methods.

In a recent study, Raj *et al*. [29] proposed CrossFIM, a hybrid algorithm for large-scale distributed extraction of frequent itemsets, specifically designed for the Apache Spark framework. This method dynamically alternates between the Apriori and Eclat strategies depending on the depth of the analysis, in order to combine the efficiency of candidate generation with the speed of tidset intersections. Particular attention was paid to optimizing communications between nodes through a clever partitioning mechanism, significantly reducing the volume of exchanged data. Experimental results indicate that CrossFIM offers excellent scalability and efficient memory resource management, making it particularly suitable for analyzing very large transactional datasets.

Rochd and Hafidi [30] introduce DSSS (Distributed Single Scan on Spark), an algorithm designed to optimize the extraction of frequent itemsets in a Big Data environment, fully leveraging the distributed capabilities of Apache Spark. Their method relies on a single scan strategy of the transactional database, which contrasts sharply with traditional algorithms requiring multiple successive passes. To achieve this, DSSS uses in-memory RDDs and a global hash table broadcast to all nodes. This design effectively reduces the latency caused by disk I/O and inter-node communications, which are often bottlenecks in distributed architectures. The algorithm also incorporates an optimization of the management of local support structures, dynamically filtering out irrelevant items before the aggregation phase. Experiments conducted on public datasets (notably Retail, T40I10D100K) show that DSSS outperforms several reference distributed algorithms, such as PFP-Tree, and D-Apriori, particularly on large and sparsely dense databases. In terms of scalability, the results indicate a significant reduction in execution time as the number of nodes increases, with a nearly linear behavior, demonstrating a good distribution of the processing load. Moreover, the algorithmic complexity remains manageable, as it primarily depends on the average number of items per transaction rather than the total size of the dataset.

Singla and Gandhi [31] propose an algorithm titled PDReLim ("Parallel and Distributed Recursive Elimination"). This work aims to optimize the mining of frequent itemsets in distributed environments, leveraging PySpark and Spark's RDD capabilities. PDReLim differs from traditional approaches through a recursive local pruning strategy: each node removes items deemed infrequent before generating new candidates, thereby significantly reducing the search space. This local optimization allows for limiting inter-node exchanges and controlling combinatorial complexity. The experiments, conducted on standard datasets (Chess, Mushroom, Connect) from the UCI repository, demonstrate that PDReLim significantly outperforms traditional iterative MapReduce methods such as PApriori, PFP-Growth, and PFP-Max, particularly at low support thresholds, a situation favorable for acceleration due to in-memory processing.

3. PROBLEM DEFINITION

In this section, we introduce the definitions of frequent pattern mining in a distributed computing environment, and we present the concept of Entropy and Quartiles.

3.1. FREQUENT PATTERNS

Consider an itemset with n different items, $I = \{x_1, x_2, x_3, ..., x_n\}$. A pattern or itemset is a subset of this itemset, represented as $X \subseteq I$. Since each transaction T is a subset of I and is uniquely recognized by its transaction ID (TID), a database DB is basically a collection of transactions. The amount of DB transactions that contain a pattern X, shown as support (X), indicates whether or not it is supported. The formula is calculated as follows [2]:

$$support\ (X) = \frac{frequency(X)}{N} \quad (1)$$

where $count(X)$ is the frequency of X in the database and N is the total number of transactions. If a pattern's support achieves or exceeds a minimal support level, shown by the symbol ξ, it appears to be frequent. Finding every pattern that frequently appears in a transaction database while staying within the specified support threshold ξ is the primary goal of frequent pattern mining. For many data analytics applications, this process is crucial, particularly when it comes to finding significant patterns and correlations in enormous quantities of data.

The two main approaches in frequent pattern mining are the pattern growth method and the candidate set generation method. The choice between these methods is primarily determined by the size and complexity of the dataset, even though each offers special advantages in certain circumstances.

- Candidate Set Generation Method [8]: This conventional approach is a systematic process that generates and examines candidate sets to identify recurring patterns in a dataset. One popular technique that applies this idea is Apriori. Its foundation is the idea that no pattern that is not prevalent in the database cannot also be frequent if it is formed from a pattern of length (k+1). All of a pattern's subsets must similarly satisfy the necessary frequency threshold in order for it to be considered frequent. Even though this method works well for identifying significant patterns, it can become computationally difficult, particularly when working with big datasets.
- Pattern Growth Method [7]: This method, in contrast to the candidate set generation method, uses previously found frequent patterns to avoid the requirement to build candidate sets. It concentrates on identifying local, frequent patterns that are gradually extended to produce more extensive global patterns rather than methodically integrating every potential item. Its computational efficiency is one of this method's main advantages. Processing huge datasets is rendered easier by reducing the requirement for multiple database scans by converting the database into a more compact and memory-efficient structure.

3.2. FREQUENT PATTERNS IN DISTRIBUTED DATABASES

There are n sites in a distributed system, which are designated $S_1, S_2, S_3, ..., S_n$. Within this system, the database DB is horizontally partitioned into n segments, denoted as $DB_1, DB_2, DB_3, ..., DB_n$, where each partition DB_i is allocated to a specific site S_i for processing (for i = 1, ..., n). To evaluate the occurrence of a pattern X, we define $support_i(X)$ as its local support count within, DB_i, while $support\ (X)$ represents its global support count across the entire distributed system. If a pattern X satisfies the minimal support criterion min_{sup}, which is established by the formula:

$$support\ (X) \geq min_{sup} \times |DB| \quad (2)$$

it can be considered globally frequent. In a distributed database setting, this condition is important since it assures that the pattern appears frequently across several partitions.

3.3. CONCEPT OF ENTROPY AND QUARTILES

In information theory, entropy is a measure of the uncertainty of a random variable. In this study, entropy is used to measure the distribution of items and enable the removal of branches in prefix trees. The conventional Shannon [19] formula for entropy is given by (Equation 3):

$$H(X) = -\sum_{i=1}^{n} p(X_i) log_2 p(X_i) \quad (3)$$

$P(X)$ is the probability that an item X is present in a transaction. Takes into account only the occurrence of an item. Here, we use its binary form [20], which also considers the non-occurrence; the formula can be expressed as (Equation 4):

$$H(X) = -P(X) log_2 p(X) - \big(1 - p(X)\big) log_2 \big(1 - p(X)\big) \quad (4)$$

$P(X)$ is the probability that a branch X is present in a transaction. $1-P(X)$ is the probability that X is absent. This formula measures the uncertainty or impurity related to the presence or absence of X. If the item or branch is very frequent or very rare, entropy is low (less uncertainty). If the item or branch appears about 50% of the time, entropy is high (maximum uncertainty).

Each branch is assessed using this measure to ascertain how informative it is. To filter out less relevant branches, we employ statistical quartile-based adaptive pruning. Level $Q1$ (25%): immediate elimination of branches whose entropy is lower than the first quartile ($Q1$). Median Level $Q2$ (50%): removal of the remaining branches whose entropy is lower than the median, but only partially. Level $Q3$ (75%): exclusion of branches with entropy values falling below the third quartile ($Q3$), thereby including only highly relevant and informative branches. The ranks are calculated according to the formula (Equation 5) [21]:

$$Q = q \times (N - 1) + 1 \quad (5)$$

Where q corresponds to 0.25 ($Q1$), 0.50 ($Q2$), or 0.75 ($Q3$), and N is the total number of values in the sample sorted in ascending order. When the position obtained is decimal, linear interpolation is applied in accordance with the method developed by Walpole *et al.* [22] as follows:

$$Q = V_{inf} + f \times (V_{sup} - V_{inf}) \quad (6)$$

In this case, V_{inf} is the value at rank i in the ranked list, and V_{sup} is the value at rank i+1. The letter f indicates the fractional part of the calculated rank. This approach provides a gradual reduction of branches, while still preserving those of high information content.

4. PROPOSED APPROACH

In this section, we present the AGFPM approach, developed according to a Master-Slave architecture adapted to distributed environments [32]. The process begins with the division of the transactional database into several fragments, each assigned to a slave site. Each Slave site then ensures autonomous processing of its portion of data by locally constructing a prefix tree (Local Prefix-Tree) in a parallel manner. A consolidation step then occurs to aggregate the local structures into a unified global tree (Global Prefix-Tree). Unlike classical methods that rely solely on absolute frequencies, this fusion is based on a binary entropy measure, allowing for the evaluation of the informational contribution of each subtree. The approach thus allows for the efficient extraction of global frequent patterns from this consolidated structure. The architectural representation of the entire process is presented in Fig.1.

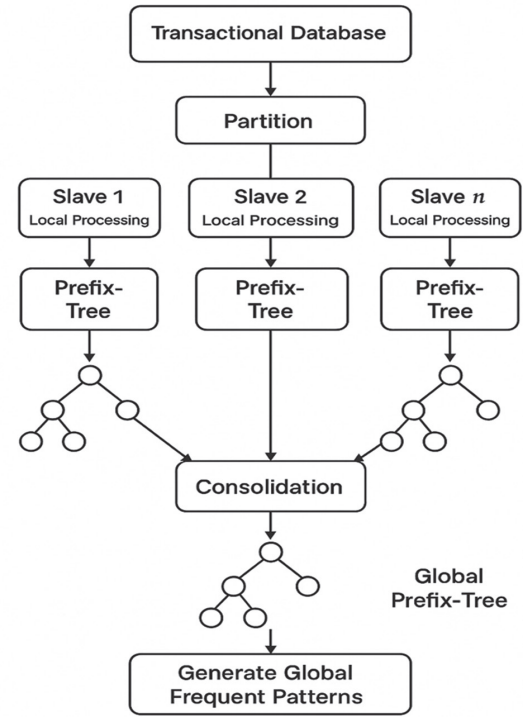

Fig. 1. General process of AGFPM algorithm

4.1. SCHEME OF COMMUNICATION

In the context of distributed algorithms, the Master-slave scheme constitutes a fundamental architecture that allows for the efficient coordination of operations between multiple processing nodes. This model relies on a clear separation of roles: the Master site assumes the responsibility of the overall orchestration of the process, while the slave sites locally execute the assigned tasks, such as data preprocessing, frequent itemset extraction, or association rule generation. This organization promotes a significant reduction in communication costs, a crucial aspect in distributed environments where bandwidth can be a limiting factor.

One of the major advantages of this scheme lies in its ability to minimize computation duplications, centralize aggregations, and promote a rapid convergence towards coherent global results, even when the data is massive and distributed heterogeneously. Moreover, the model facilitates the implementation of load balancing policies by dynamically distributing data partitions or subtasks according to the capacity or state of each Slave site. In the specific field of frequent pattern mining, this paradigm is commonly adopted for its ease of implementation, robustness, and compatibility with traditional distributed infrastructures (clusters, clouds).

Works such as those of Oliveira and Zaïane [33] or more recently Tseng *et al.* [34] have demonstrated that the Master-Slave scheme, combined with compact data structures like FP trees or prefix lattices, optimizes overall performance while ensuring low latency in the transfer of intermediate results.

Our algorithm relies on this scheme of communication for optimizing frequent pattern extraction. Each site constructs an LP-Tree locally, and the Master aggregates the supports and constructs a global GP-Tree. The use of entropy measures and adaptive thresholds (quartiles) allows for statistically significant pruning, ensuring global coherence, accuracy of extracted patterns, and minimization of exchanges between sites.

4.2. DEVELOPMENT OF PARALLEL LOCAL PREFIX-TREE

The Local LP-Tree, or Local Prefix Tree, is a parallel structure developed for each partition of the distributed database. It follows a tree-like structure modeled on the FP-Tree and has a root node along with multiple subtrees that denote prefix paths; each node contains a local support counter and a pointer to similar nodes. A Local Header Table is built to facilitate the subsequent extraction of conditional patterns. The process of construction involves three main steps. First, each slave site sorts its transactions according to a predefined lexicographic order, inserts the prefixes into an LP-Tree, and records the counters in the local header table.

Then, in the second phase, all the counters are sent to the Master site, which calculates the binary Shannon

entropy of each prefix. In order to estimate its informational value. The prefixes are then globally ranked by decreasing entropy, and this ranking is used to restructure the branches of the local trees, thereby ensuring global coherence between the sites. The LP-Trees are then reorganized according to this new order and their local header tables updated. The use of the order of global entropies in our solution guarantees a meaningful tree, homogeneous at each node, and optimum efficiency for the subsequent construction of a global prefix tree to accurately mine globally frequent patterns.

In the third step, all Slave sites compute the binary Shannon entropy of their whole branches (paths from the root to the node) to approximate the informative quality of the branches in each LP-tree. Most frequent branches carry a higher probability. Less frequent branches carry a lower probability, which is the reason for their elimination during pruning.

To conduct the multi-level pruning of the LP-tree branches by their entropies, we establish a dynamic adaptive threshold for the pruning process; this is determined using the quartiles of the entropies calculated. The values of branch entropies are sent to the Master site, where they are merged into one list [$H(B_1)$, $H(B_2)$, $H(B_n)$], in order to calculate three adaptive thresholds as quartiles ($Q1$-25%, $Q2$-50%, $Q3$-75%), we arrange the entropies in ascending order as follows:

$$H(B_1) \leq H(B_2) \leq ... \leq H(B_n) \quad (7)$$

We then use the formula of Cover and Thomas [24] cited in subsection 3 of section 3 to calculate $Q1$, $Q2$ (median), and $Q3$. The first quartile ($Q1$, 25%) allows the immediate removal of branches with very low entropy. The second quartile (Median, 50%) removes systematically branches that have medium entropy. The third quartile ($Q3$, 75%) is used as a last resort to keep only the most informative branches.

The setting of these three thresholds enables a systematic, incremental, and statistically valid pruning process by ensuring that only branches of high informational value (defined by high entropy, reflecting a high diversity of items) are retained. Therefore, branches with entropy levels below these three thresholds are eliminated, thus allowing the incremental elimination of unnecessary and less valuable branches at different levels. The first quartile ($Q1$) level involves the rapid elimination of branches with entropy values below this first quartile. Median Level ($Q2$) is the elimination of branches whose entropy levels fall below the median. Level $Q3$ is the stringent elimination of branches whose entropy is still lower than the third quartile ($Q3$), so that only the most informative and relevant branches are retained. This improved method supports the precise and efficient strengthening of the LP-tree to obtain an optimized LP-tree, avoiding overly aggressive removal of useful information and maintaining a balance between memory savings and preservation of significant patterns.

Finally, from the optimized LP-Trees, each site extracts a local database of conditional patterns (CL), containing for each prefix its ancestors and their associated supports. These databases contain the essential information and will act as a basis for the merging of patterns on a global scale in the subsequent steps. Algorithm 1 gives the phases involved in our LP-Parallel Tree construction.

Algorithm 1: Parallel Construction of LP-Tree

Input:

$BD \leftarrow$ Global transactional database

$N \leftarrow$ Number of distributed sites

$min_{sup} \leftarrow$ Minimum support threshold

Output:

LP-Tree-Optimized[i] ← Pruned local prefix trees for each site S_i

$CL[i] \leftarrow$ Local conditional pattern bases from each site

1. // *Step 1*: Horizontal partitioning of the database
2. Divide BD into N disjoint subsets $\{BD_1, BD_2, ..., BD_n\}$
3. Distribute BD_i to each corresponding site S_i
4. // *Step 2*: Local construction of *LP-Trees*
5. **for each** site S_i **do**
6. Build initial *LP-Tree* [i] from BD_i (prefix-based tree)
7. Construct *Local-Header-Table* [i] from *LP-Tree* [i]
8. Each node stores: prefix, local counter, and node links
9. Send local prefix counters to the Master site
10. **end for**
11. // *Step 3*: Global entropy computation
12. Master aggregates all local counters
13. **for each** prefix P_i **do**
14. $P(p_i) \leftarrow$ frequency $(P_i) / |BD|$
15. $H(p_i) \leftarrow -P(p_i) \cdot \log_2(P(P_i)) - (1 - P(p_i)) \cdot \log_2(1 - P(p_i))$
16. **end for**
17. Sort prefixes by decreasing entropy $H(P_i)$
18. Send the sorted prefix order to all sites
19. // *Step 4*: Tree reconstruction using global entropy order
20. **for each** site S_i **do**
21. Reorder transactions based on entropy order
22. Rebuild LP-Tree [i] accordingly
23. Rebuild Local-Header-Table [i] from updated LP-Tree [i]
24. **end for**
25. // Step 5: Compute branch entropies
26. **for each** site S_i **do**
27. **for each** branch B_j in *LP-Tree* [i] **do**
28. Compute entropy $H(B_j)$ using Shannon binary entropy

29. **end for**
30. Send all branch entropies to the master
31. **end for**
32. // *Step 6*: Global adaptive thresholds (quartiles)
33. Master aggregates all entropies and computes:
34. $Q_1 \leftarrow 25\%$; $Q_2 \leftarrow 50\%$; $Q_3 \leftarrow 75\%$
35. Send Q_1, Q_2, Q_3 to all sites
36. // *Step 7*: Multilevel pruning based on entropy
37. **for each** site S_i **do**
38. Remove branches with $H(B) <$ all thresholds (Q_1, Q_2, Q_3)
39. *LP-Tree-Optimized* [*i*] ← resulting pruned tree
40. **end for**
41. // *Step 8*: Extract local conditional patterns
42. **for each** *LP-Tree-Optimized* [*i*] **do**
43. Extract *CL* [*i*] ← Conditional pattern base for each prefix
44. **end for**

4.3. DEVELOPMENT OF GLOBAL PREFIX-TREE

In this section, we focus on the global knowledge from the local structures by constructing a global compact prefix arborescence from the pruned local LP-Trees. Our algorithm AGFPM is responsible for focusing this distributed information by building a global prefix tree known as GP-Tree. The Master site receives all the CL extracted by the local sites. For each prefix, the corresponding ancestors (and their counters) are merged. When several sites share the same prefix and ancestor associations, their counters are aggregated. This phase builds a unified structure that gathers all the local data in a condensed format. After the aggregation process is finished, the GP-Tree is developed by adding each Ancestor → Prefix path into a central tree while keeping the aggregated counters by means of a Global Header Table (GHT) that encompasses all of the counters for each prefix. The result is a compact, centralized arborescent structure that represents the global distribution of frequent patterns. This structure is the foundation of more advanced pruning and extraction processes that are part of the approach. In order to accommodate various analysis environments, the method permits the user to select one of two GP-Tree processing modes. In the first instance, pruning can optionally be used to cut down on the complexity of the tree, thereby reducing the amount of extracted frequent patterns and concentrating the results on the most important branches.

The method employed is as follows: Each distinct path in the GP-Tree (root to node) is analyzed with Shannon's binary entropy to quantify the amount of uncertainty or information associated with this path. The objective is to identify the most important branches, i.e., those that carry the highest amount of information regarding the overall frequent patterns. Following the computation of the entropies of all the branches, the Master site proceeds to define the distribution and calculate three adaptive thresholds in the form of quartiles: Q_1 (25%), Q_2 (50% or median), and Q_3 (75%). These thresholds enable multi-level pruning to be carried out. To be precise, any branch with an entropy value that falls below these three thresholds (Q_1, Q_2, and Q_3) is assumed to be of very limited informational value and is therefore pruned. This approach enables retaining only the most informative patterns while, concurrently, minimizing the size of the tree and eliminating redundancies. The GP-Tree thus produced is not only more concise and informative but also more appropriate for mining high-quality global conditional patterns and thus reducing redundancy and increasing interpretability of global frequent patterns. In another mode, the user can opt to keep the entire GP-Tree, hence enabling an exhaustive pattern extraction without utilizing rigorous filtering techniques. Such flexibility enables the adjustment of analytical granularity to suit each application's individual requirements, either to performance or to comprehensiveness. GP-Tree is instrumental in the derivation of global conditional models, which ultimately enables the determination of common global patterns. In either mode, the last GP-Tree is utilized as the foundation for the construction of global conditional models (CG), which results in the extraction of global frequent patterns. Algorithm 2 describes the steps of constructing the GP-tree.

Algorithm 2: Construction of GP-Tree

Input:

CL [1...*N*] ← Local conditional pattern bases from N distributed sites

Output:

GP-Tree ← Global Prefix Tree constructed from merged local

CLs

CG ← Global conditional patterns

1. Initialize an empty dictionary Aggregated-*CL* ← {}
2. // *Step 1*: Aggregate local conditional patterns
3. **for** $i \leftarrow 1$ to N **do**
4. for each (prefix P, list of ancestors $\{A_1, A_2, ..., A_k\}$) in *CL* [*i*] **do**
5. **if** $P \notin$ Aggregated-CL **then**
6. *Aggregated-CL* [*P*] ← {}
7. **for each** ancestor A_j in $\{A_1, ..., A_k\}$ **do**
8. **if** $A_j \in$ *Aggregated-CL* [*P*] **then**
9. *Aggregated-CL* [*P*][A_j] ← *Aggregated-CL* [*P*][A_j] + *count* (*CL* [*i*][*P*] [A_j])
10. **else**
11. *Aggregated-CL* [*P*][A_j] ← *count* (*CL* [*i*] [*P*] [A_j])

12. **end for**
13. **end for**
14. **end for**
15. // *Step 2*: Construct the Global Prefix Tree (GP-Tree)
16. Initialize GP-Tree as an empty prefix tree with a root node
17. **for each** prefix P in $Aggregated\text{-}CL$ **do**
18. **for each** ancestor A_j in $Aggregated\text{-}CL$ [P] **do**
19. Insert the path $A_j \rightarrow P$ into GP-Tree
20. Update the node counters with $Aggregated\text{-}CL$ [P][A_j]
21. Construct GHT from GP-Tree // Global Header Table
22. **end for**
23. **end for**
24. // *Step 3*: Extract final global conditional patterns
25. CG ← Extract global conditional patterns from GP-Tree
26. **return** GP-Tree

4.4. GLOBAL FREQUENT PATTERNS MINING

In our approach, global conditional patterns (CG) are used (CG) to build GP subtrees (sub-GP-Trees) at various levels of iterations as K. Each sub-GP-Tree categorizes systematically frequent patterns at a specific level of granularity, thereby enhancing the detection of more complicated combinations, e.g., pairs or triplets. Unlike an FP-tree that explores elements from bottom to top, the sub-GP-Tree is traversed from top to bottom in the Global Header Table. This top-down exploration method allows the efficient derivation of all frequent global element sets related to X.

The procedure is iterative: At level $K = 1$, for each prefix of size (m=1) obtained from the global conditional patterns (CG) of GP-tree, a sub-GP-Tree is produced. For each frequent global pattern of size ($m = 2$) obtained from the initial level patterns, a sub-GP-Tree is built at level $K = 2$. It is done by building the sub-GP-Tree based on the intersection of the paths for subsets of global frequent patterns. The mining of common paths among subsets of X, each containing (m−1) elements and sharing common prefixes of length (m−2) at the start of the frequent global patterns X, constitutes the next step of the procedure at level K≥3. This procedure continues until no new sub-GP-Tree can be constructed, i.e., no further frequent global pattern can be identified. Likewise, the sub-GP-Tree of XY cannot be constructed if X is part of a global frequent pattern XY and does not have any ancestors, and therefore, prevents the construction of additional global frequent patterns. Algorithm 3 describes the procedure for obtaining global frequent patterns.

Algorithm 3: Extract Global Frequent Patterns

Input:

$GP\text{-}Tree$ ← Construction of $GP\text{-}Tree$ (from Algorithm 2, global header table implicitly built)

Output:

$G\text{-}Patterns$ ← Complete set of global frequent patterns

1. Initialize $G\text{-}Patterns \leftarrow \emptyset$ // Final collection of global frequent patterns
2. // *Recursive pattern mining starting at level* $K = 1$
3. **for** $K \leftarrow 1$ to ∞ do
4. New-Patterns ← ∅ // Temporary patterns discovered at level K
5. **if K** = 1 then
6. // *First level:* explore directly from GP-Tree structure
7. **for** each global frequent pattern X from $GP\text{-}Tree$ **do**
8. Build $Sub\text{-}GP\text{-}Tree$ [X] from conditional paths including X
9. **if** $Sub\text{-}GP\text{-}Tree$ [X] ≠ ∅ **then**
10. Extract frequent patterns from $Sub\text{-}GP\text{-}Tree$ [X]
11. Add them to New-Patterns and to G-Patterns
12. **end if**
13. **end for**
14. **else**
15. // *Next levels*: *explore from each global header table GHT of previous subtrees*
16. **for each** global frequent pattern X derived from $Sub\text{-}GP\text{-}Tree$ [X]. GHT **do**
17. Build $Sub\text{-}GP\text{-}Tree$ [X] from conditional paths including X
18. **if** $Sub\text{-}GP\text{-}Tree$ [X] ≠ ∅ **then**
19. Extract frequent patterns from $Sub\text{-}GP\text{-}Tree$ [X]
20. Add them to New-Patterns and to G-Patterns
21. **end if**
22. **end for**
23. **end if**
24. **if** New-Patterns = ∅ **then**
25. break
26. **end if**
27. // *Update header table of each* $Sub\text{-}GP\text{-}Tree$ [X] *for the next level*
28. Update GHT for each $Sub\text{-}GP\text{-}Tree$ [X] // Global Header Table: GHT
29. **end for**
30. **return** G-Patterns

Our proposed AGFPM algorithm is based on a succession of complementary steps, each one of which is essential for building the data structures required for the final extraction. The method permits concentrating computational resources on the most representative patterns using the techniques of the distributed organization of processing, focused use of entropy to prioritize information, and application of intelligent pruning. Both structural and informational, this optimization offers a strong basis for proceeding to the study of the results. It enables a valid assessment of the relevance of the identified patterns as well as a clear observation of the performance of the suggested approach in several experimental settings.

5. RESULTS AND DISCUSSION

We carried out comprehensive experiments on two types of datasets, as indicated in Table 1, with distinct properties in order to assess the performance of our AGFPM. The synthetic dataset T10I4D100K was generated by the IBM Almaden Quest group and obtained from [23]. The Kosarak dataset was used in its transaction format as distributed by SPMF [24]. The Chess dataset, derived from the UCI Machine Learning Repository, was used in its transaction-encoded form as distributed by SPMF [25]. We compared AGFPM against well-known methods such as PFP-Tree and CD. The tests were conducted on a PC running Windows 11 with an Intel® CoreTM i7-10875H CPU running at 2.80 GHz and 16 GB of RAM. The datasets were dispersed among 3, 5, and 7 Slave sites in order to evaluate scalability and efficiency. The Java program was generated with the NetBeans IDE. MPJ Express, a Java-based message passing framework created especially for running parallel applications on multicore machines, facilitates communication between sites.

Table 1. Dataset Characteristic

Dataset	Transaction	Items	Max TL (Maximum Tree Length)	Avg TL (Average Tree Length)
T10I4D100K	100000	870	29	10.1
Kosarak	990002	41270	2498	8.10
Chess	3196	75	38	36

5.1. ANALYSIS OF PERFORMANCES

The efficiency and performance of the PFP-Tree method, CD algorithm, and proposed AGFPM technique vary considerably in the case of the T10I4D100K, Kosarak, and Chess datasets. Fig. 2 presents a detailed comparison of the execution times of the AGFPM algorithm applied to the T10I4D100K dataset, based on different values of the minimum support threshold (MinSupp).

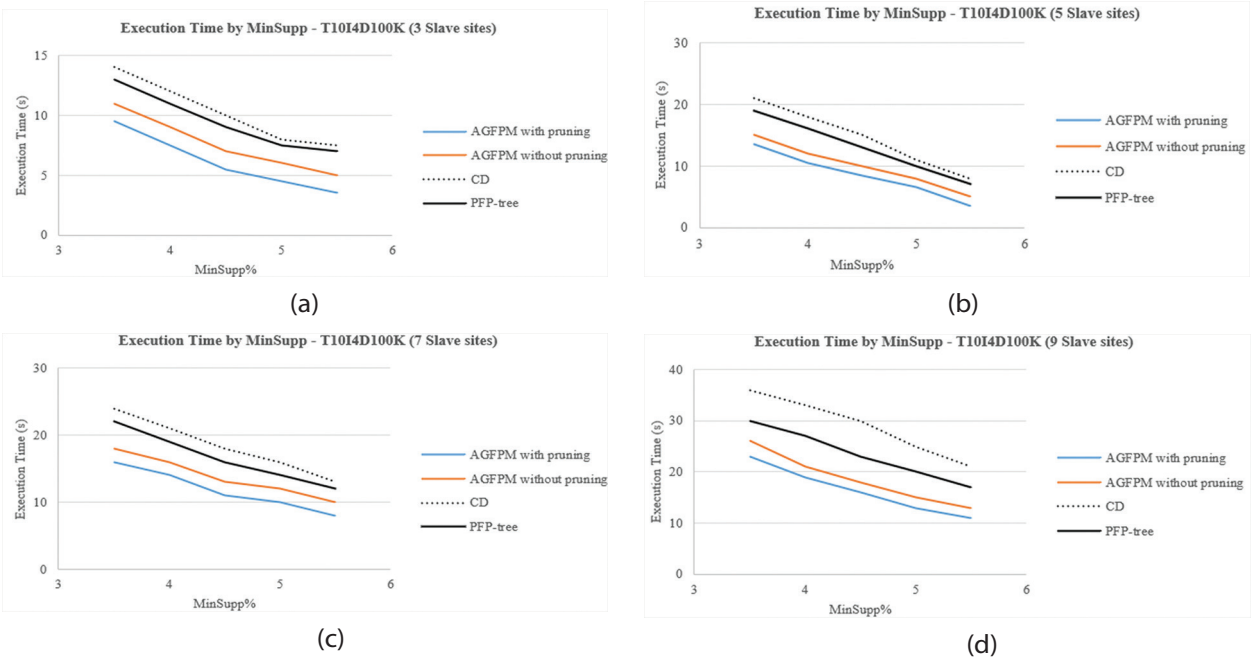

Fig. 2. The excution time of T10I4D100K with (a) 3 numbers of Slave sites, (b) 5 numbers of Slave sites, (c) 7 numbers of Slave sites, (d) 9 numbers of Slave sites

The study covers three distributed configurations, involving 3, 5, 7, and 9 slave sites respectively, in order to evaluate the impact of the degree of parallelism on performance. The displayed curves allow for a comparative analysis of the behavior of four algorithms: AGFPM with pruning, AGFPM without pruning, CD (Count Distribution), and PFP-tree (Parallel FP-tree), for MinSupp values ranging from 3.5% to 5.5%. This visualization provides a detailed analysis of the gains achieved by AGFPM compared to classical methods, while highlighting the influence of the number of slave nodes on execution speed and the processing capacity of the different models.

For a configuration with 3-slaves, the AGFPM algorithm with pruning shows the best results, with execution time decreasing from 9.5 s to 3.5 s. This improvement demonstrates the effectiveness of the entropy-based pruning mechanism, which reduces the complexity of the global tree by removing uninformative branches. In comparison, the version without pruning is slightly slower (between 11 s and 5 s), which confirms the value of adaptive filtering. The reference approaches are less effective: CD varies between 14 s and 7.5 s, while PFP-tree drops from 13 s to 7 s, revealing their limitations in contexts where parallelism is moderate.

When the number of slave sites is increased to 5, performance improves significantly. AGFPM with pruning reaches 13.5 s at MinSupp = 3.5%, and drops to 3.5 s at MinSupp = 5.5%. The gain becomes even more notable compared to other methods: CD remains above 21 seconds, and PFP-tree around 19 seconds at the minimal MinSupp. The stability of the differential between the two versions of AGFPM confirms that pruning maintains its effectiveness, regardless of the level of parallelism. This behavior validates the adaptability of the model to a medium-sized distributed architecture.

In the 7-slave configuration, although execution times continue to decrease (for example, AGFPM with pruning goes from 16 s to 8 s), the marginal gain diminishes. This is due to the increase in synchronization costs, particularly in the global aggregation phase. The CD algorithm caps at 24 seconds for the lowest threshold, while the PFP-tree remains at 22 seconds. These results show that excessive parallelism can sometimes induce an overhead that reduces the overall efficiency of the system, especially when the size of the partitions becomes too fine relative to the communication load between the master and the slaves.

Under the 9-slave configuration, the ordering is stable, and runtimes shrink as MinSupp increases. At 3.5%, AGFPM with pruning is 23s, clearly ahead of PFP-tree (30s) and CD (36s); by 5.0%, it falls to 13s while competitors remain higher. This pattern reflects a design that aligns local structures before aggregation and filters low-yield branches early, reducing both search space and cross-site reconciliation advantages that become more salient as coordination costs rise at this degree of parallelism.

Across all tested configurations, AGFPM with pruning stands out clearly, combining speed, scalability, and relevance of the extracted patterns. The use of pruning based on entropy quartiles allows for the dynamic regulation of the growth of the global tree, which lightens the processing without compromising the quality of the results. These observations highlight the relevance of the proposed approach in distributed environments, particularly when the goal is to efficiently process large volumes of data with low frequency thresholds.

Fig. 3 illustrates the evolution of the execution time of the various distributed algorithms applied to the Kosarak dataset, known for its high density.

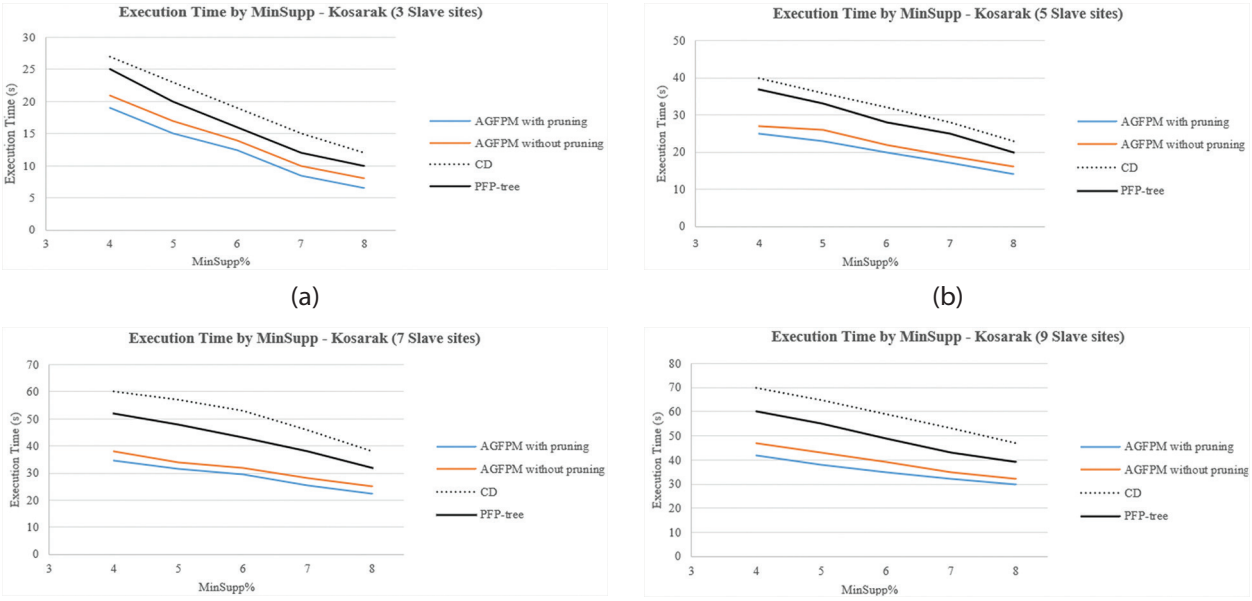

Fig. 3. The execution time of Kosarak with (a) 3 numbers of Slave sites, (b) 5 numbers of Slave sites, (c) 7 numbers of Slave sites, (d) 9 numbers of Slave sites

The results are analyzed for five increasing values of the minimum support threshold (from 4% to 8%) and at three levels of parallelism, involving 3, 5, and 7 slave sites. The performances of five algorithms are compared: AGFPM with pruning, AGFPM without pruning, CD, and PFP-tree.

In the configuration with 3 slave sites, the AGFPM algorithm with pruning stands out for its speed, with execution time decreasing from 19 seconds at MinSupp = 4% to 6.5 seconds at 8%. This efficiency is explained by the impact of the pruning strategy, which significantly reduc-

es the volume of patterns to be aggregated. The version without pruning is slower, reaching 21 seconds at 4% and 8 seconds at 8%, which confirms the value of informational filtering. The CD and PFP-tree algorithms show significantly higher execution times: for example, CD starts at 27 seconds at 4% and remains at 12 seconds at 8%, revealing a low adaptability to transaction density.

With 5 slaves, the gap between the algorithms becomes more pronounced. AGFPM with pruning remains the most efficient, with a time of 25 seconds at 4%, dropping to 14 seconds at 8%. In comparison, CD requires 40 seconds at 4% and 28 seconds at 8%, proving its low scalability in high-density environments. PFP-tree, although improved by parallelism, remains above 20 seconds even for high MinSupp values. The stability of AGFPM's performance with pruning confirms its ability to exploit parallelism while maintaining low computational overhead.

When the approach is deployed on 7 slaves, the observed trends are confirmed. The execution times remain controlled for AGFPM with pruning, which goes from 34.5 seconds at MinSupp = 4% to 22.5 seconds at 8%. The competing algorithms, on the other hand, show obvious limitations. CD caps at 60 seconds at 4% and struggles to go below 38 seconds, despite the increase in the number of nodes. PFP-tree follows a similar trend (from 52s to 32s).These results highlight the structural limitations of these methods when they have to manage a high density in a massively distributed context.

Across 9 slave sites, the ordering remains consistent, and runtimes decrease steadily as MinSupp rises. At 4%, AGFPM with pruning records 42s, clearly below PFP tree (60s) and CD (70s); by 8%, it reaches 30s while competing methods remain higher (PFP tree 39s, CD 47 s). This trajectory indicates that the pruned variant sustains a stable margin across the range, with the gap most visible at lower supports where the search space and coordination overheads are largest. The advantage arises from a design that harmonizes local structures before aggregation and suppresses low-yield branches early, thereby reducing both conditional growth and cross-site reconciliation effects that are particularly beneficial on a large, sparse workload such as Kosarak under high parallelism.

In summary, the results of Fig. 3 demonstrate the consistent superiority of the AGFPM approach with pruning, which combines algorithmic efficiency, informational compression, and intelligent exploitation of parallelism. This method adapts better than its competitors to data density and support variations, while maintaining robust scalability in the face of an increasing number of slave sites. It thus constitutes a relevant solution for the mining of global frequent patterns in complex distributed environments.

Fig. 4 depicts how execution time varies with MinSupp across four algorithms: AGFPM with pruning, AGFPM without pruning, CD (Count Distribution), and PFP-tree on the Chess dataset as the number of slave sites increases. Across all subfigures, higher MinSupp thresholds shorten runtime, while the performance ranking remains consistent: AGFPM with pruning is fastest, followed by its no-pruning variant, with PFP-tree and CD trailing gaps widening at low MinSupp where dense co-occurrences inflate intermediate structures.

Under the 3-slave setup, method separations are most visible at low MinSupp and narrow as the threshold rises. At MinSupp=0.12%, AGFPM with pruning is about 0.085s and keeps a lead that persists through mid supports (≈0.20).

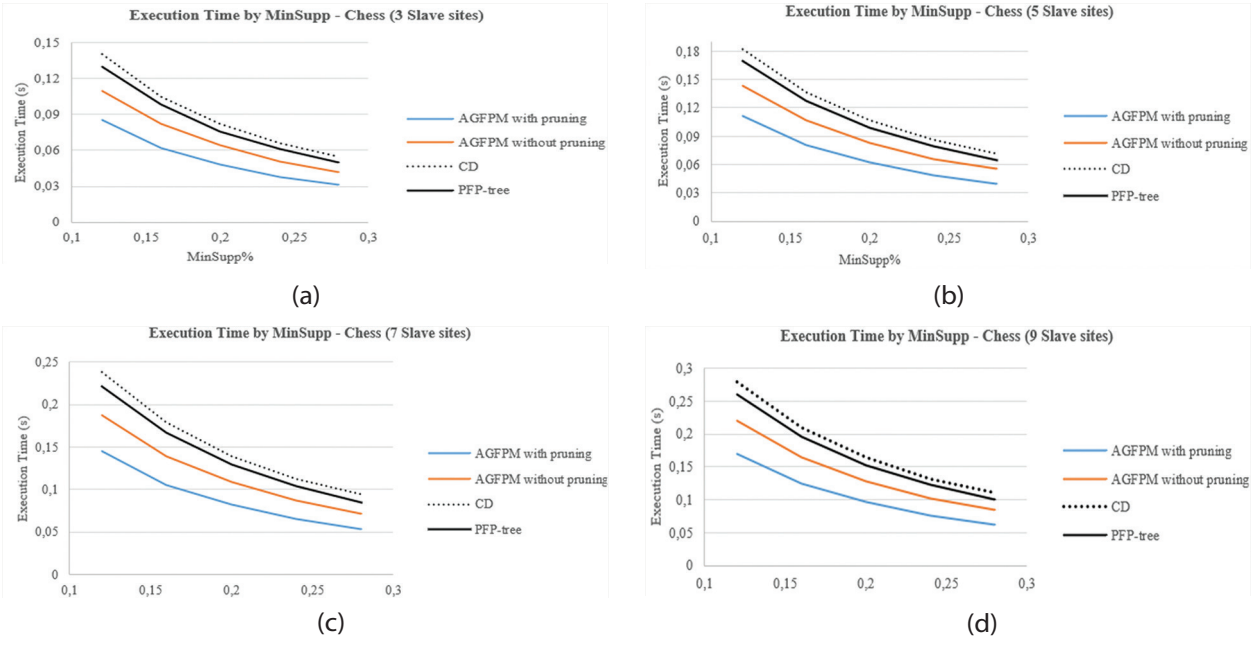

Fig. 4. The execution time of Chess with (a) 3 numbers of Slave sites, (b) 5 numbers of Slave sites, (c) 7 numbers of Slave sites, (d) 9 numbers of Slave sites

The ordering remains stable, pruned variant first, then no-pruning, with PFP-tree and CD behind. This reflects an information-guided item order that reduces cross-site mismatch and multi-level pruning that curbs conditional growth, lowering counting and merge costs, while PFP-tree and CD incur, respectively, higher merging and candidate/synchronization overheads at the smallest supports.

With 5 slave sites, the ranking remains stable and the gaps widen at low MinSupp, indicating rising structural and coordination costs as parallelism increases. At MinSupp=0.12%, AGFPM with pruning is clearly ahead (≈0.11s) and maintains a sizeable margin over its no-pruning variant and both PFP-tree and CD. The advantage persists at mid supports (e.g., around 0.20%), where the pruned version remains the fastest and the no-pruning variant stays competitive, while PFP-tree and CD trail. This efficiency is explained by information-guided item ordering that aligns local structures before aggregation and multi-level pruning that restrains conditional growth; together, they reduce merging and synchronization overheads that otherwise escalate for PFP-tree (structure merging) and CD (candidate generation and repeated coordination).

Under the 7-slave setting, separations are largest at low MinSupp and narrow as the threshold increases. At MinSupp=0.12%, AGFPM with pruning is around 0.145s and clearly ahead of its no-pruning variant, PFP-tree, and CD; the ordering remains unchanged at mid supports (e.g., 0.20%). This advantage stems from information-guided item ordering that aligns local structures before combination and multi-level pruning that limits conditional growth. By contrast, PFP-tree's merging cost and CD's candidate/synchronization overheads scale poorly at low supports and higher parallelism, keeping their curves steeper in this regime.

With 9 slave sites, the relative ranking is unchanged, and the advantage of AGFPM with pruning is most evident under low MinSupp. Communication and coordination increasingly dominate overall cost at this scale; by transmitting compact metadata and imposing a consistent global item order, AGFPM curtails the information that must be reconciled across sites. The multi-level, quartile-guided filtering further limits conditional expansions, sustaining favorable runtime trends as parallelism grows.

On dense transactional data such as Chess, especially at low MinSupp and higher degrees of parallelism, AGFPM with pruning provides the most stable and efficient behavior. Its combination of information-guided ordering, compact exchanges, and adaptive filtering translates into reduced merging and synchronization costs, more predictable scaling from 3 to 9 slave sites, and a consistently lower execution-time profile than PFP-tree and CD.

The volume of motifs can be analyzed in Table 2, Table 3, and Table 4 according to five different support thresholds and for each algorithm (CD, PFP-Tree, AGFPM with and without pruning), for the T10I4D100K, Kosarak, and Chess datasets.

Table 2. Number of Frequent Patterns Mined for Kosarak

Algorithm	4% Min Supp	5% Min Supp	6% Min Supp	7% Min Supp	8% Min Supp
AGFPM with pruning	5228	5107	4485	4110	3698
AGFPM without pruning	5311	5283	4555	4275	3475
CD	5728	5314	5151	5018	4800
PFP-Tree	5546	5123	5000	4767	4105

Table 3. Number of Frequent Patterns Mined for T10I4D100K

Algorithm	3.5% Min Supp	4% Min Supp	4.5% Min Supp	5% Min Supp	5.5% Min Supp
AGFPM with pruning	2373	2254	2119	1822	1476
AGFPM without pruning	2420	2339	2275	2120	1653
CD	3532	3258	3100	2508	2003
PFP-Tree	2891	2670	2450	2254	1854

Table 4. Number of Frequent Patterns Mined for Chess

Algorithm	0.12% Min Supp	0.16% Min Supp	0.2% Min Supp	0.24% Min Supp	0.28% Min Supp
AGFPM with pruning	5210	4620	3980	3410	2890
AGFPM without pruning	5480	4840	4210	3620	3050
CD	5728	5110	4450	3880	3280
PFP-Tree	5610	4960	4320	3730	3160

Table 2 shows that AGFPM with pruning effectively limits the number of patterns extracted on a dense dataset like Kosarak. Compared to other algorithms, particularly CD and PFP-tree, it produces a more compact set of patterns, which facilitates interpretation and reduces redundancy. Entropy-based pruning thus allows targeting the most relevant patterns while maintaining good data coverage.

On the T10I4D100K dataset, Table 3 confirms the previously observed trend: AGFPM with pruning extracts fewer patterns than the other methods, particularly at low support thresholds. This ability to contain the size of the results is valuable for optimizing analysis, especially in distributed environments. In comparison, CD and PFP-tree generate a higher volume of rules, which can lead to an overload during post-processing.

Table 4 summarizes the number of frequent patterns identified in the Chess dataset across a range of minimum support thresholds. As the threshold increases, the volume of discovered patterns declines, consistent with a stricter inclusion criterion. Across methods, AGFPM with pruning consistently yields the most compact pattern sets, reflecting the effectiveness of early elimination of redundant or low-utility branches. The variant without pruning systematically retains more patterns,

illustrating the cost of deferring filtering. Classical baselines such as CD and PFP-tree produce larger outputs at all thresholds, indicating greater retention of redundant structures and, consequently, higher computational and storage overheads.

Overall, the three tables consistently demonstrate the superiority of AGFPM with pruning in controlling the volume of extracted frequent patterns. Across the evaluated datasets—Kosarak (large and sparse), T10I4D100K (lower density), and Chess (dense)—AGFPM produces markedly more compact pattern sets than classical baselines such as CD and PFP-tree. This conciseness does not come at the expense of result quality: the pruning strategy is guided by an entropy-based criterion that preserves the most informative branches of the global tree, ensuring that salient patterns are retained while redundant structures are discarded.

Compared with the non-pruned variant, the integrated filtering mechanism prevents the extraction of irrelevant or redundant patterns, an advantage that is especially valuable in distributed mining, where unnecessary candidates amplify communication, memory, and merge overheads. By contrast, traditional baselines, while sometimes competitive on specific datasets, tend to generate voluminous pattern sets that complicate interpretation and increase post-processing effort.

In summary, the quantitative results indicate that information-based adaptive pruning improves both computational efficiency and the clarity of the outcomes, yielding more compact and actionable pattern sets. This makes the approach well-suited to large transactional databases in distributed environments. Crucially, it reduces complexity without materially sacrificing informative content, which is a meaningful benefit for decision-support pipelines and time-sensitive analytics.

5.2. SCALABILITY ANALYSIS

In this section, we assess the scalability of AGFPM in a distributed setting by measuring execution time as the number of computing nodes varies (3, 5, 7, 9). The minimum support is set to 4% for T10I4D100K and Kosarak and 0.20% for Chess. The datasets span contrasting regimes: T10I4D100K (lower density), Kosarak (larger and sparser), and Chess (dense). We compare AGFPM with and without pruning against two classical baselines: CD (Count Distribution) and PFP-tree (Parallel FP-tree). The figures depict how increasing parallelism affects runtime and reveal each method's capacity to control coordination, merging, and candidate-handling overheads, thereby indicating their efficiency in a distributed environment.

Fig. 5 (a): On the T10I4D100K dataset, execution time grows steadily as the number of nodes increases from 3 to 9. The proposed AGFPM method, enhanced with multi-level pruning, consistently achieves the best performance, requiring only 7.5 seconds with 3 nodes and scaling efficiently to 19 seconds with 9 nodes. In contrast, the same method without pruning starts at 9 seconds and reaches 21 seconds, demonstrating the overhead caused by unfiltered, redundant branches. The performance gap widens further when compared to classical approaches: CD increases from 12 to 33 seconds, and PFP-tree from 11 to 27 seconds over the same range. These results underscore the effectiveness of adaptive pruning in reducing both computational load and communication volume, particularly as the system expands to larger cluster configurations. By eliminating non-promising paths early and limiting data exchange, AGFPM maintains efficiency even under increased parallelism.

Fig. 5 (b): The high-density and high-volume Kosarak dataset enhances performance gaps between the algorithms. The AGFPM with pruning continues to provide the best performance, going from 19 s with 3 nodes to 42s with 9 nodes. Without pruning, the durations rise more abruptly, from 21s to up to 47s, validating the overhead imposed by processing unfiltered data. The traditional approaches have much higher costs: CD rises sharply from 27s at 3 nodes to 70s at 9 nodes, and PFP-tree goes from 25s to 60s. The increasing performance gap between AGFPM with pruning and the others demonstrates the scalability of the pruning approach, which becomes ever more essential when the number of nodes increases and inter-node communications accelerate.

Fig. 5 (c): On the dense Chess dataset (MinSupp=0.20%), absolute runtimes are sub-second, yet the performance ordering is clear and consistent across node counts. AGFPM with pruning remains the fastest, roughly doubling from a few hundredths of a second at 3 nodes to under a tenth at 9 nodes, while the unpruned variant, PFP-tree, and CD occupy progressively higher bands at each configuration. Although the gaps are measured in milliseconds, they are systematic, indicating that pruning and a consistent item order curb conditional expansions and reduce reconciliation overhead. This stability shows that the approach is effective not only under heavy load but also in lighter regimes, where disciplined pruning still yields measurable, reproducible gains.

Overall, the findings indicate that AGFPM with pruning is particularly well suited to distributed settings. Curbing unnecessary expansions and harmonizing structures across nodes, it reduces computational load while scaling effectively with additional resources, which helps sustain low and stable runtimes even on large or highly correlated datasets.

In contrast to classical baselines that are prone to communication overhead and structural redundancy as parallelism increases, AGFPM combines efficiency, robustness, and adaptability. These properties make it a strong candidate for large-scale frequent pattern mining in modern distributed environments.

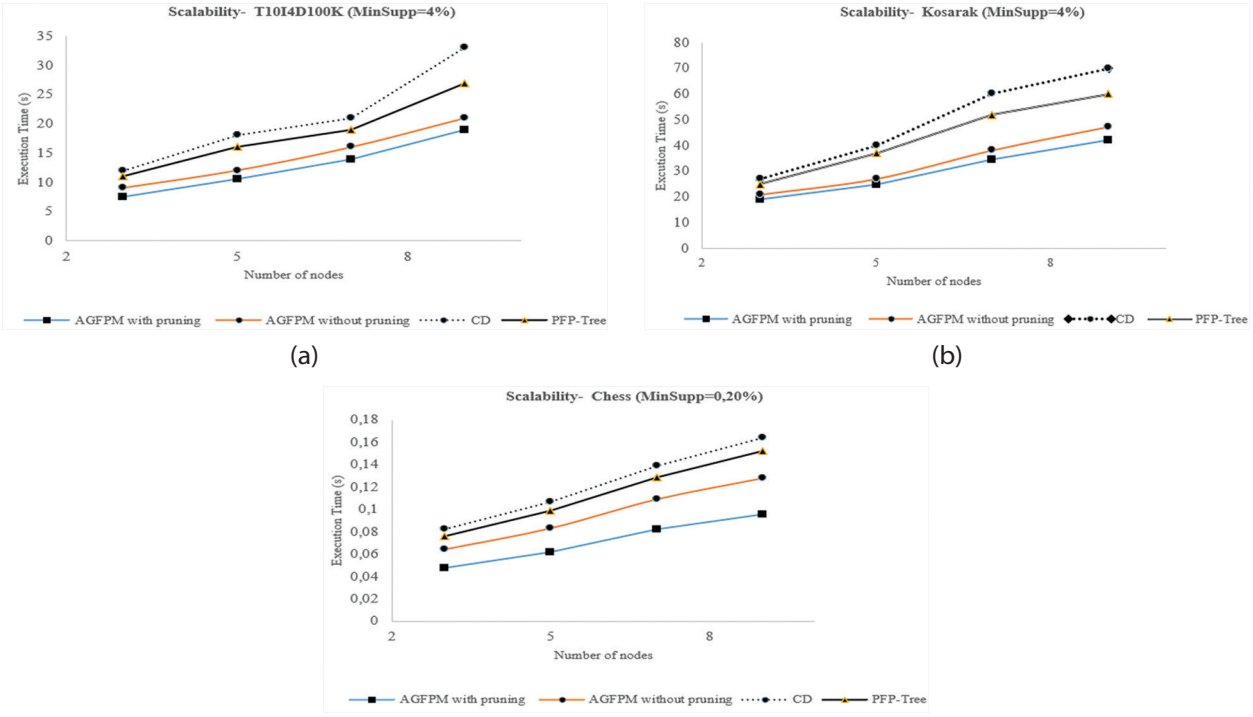

Fig. 5. Scalability of AGFPM by various number of nodes for (a) T10I4D100K, (b) Kosarak with MinSupp = 4% and (c) Chess with MinSupp = 0.20%

5.3. COMPARISON WITH RELATED METHOD ANALYSIS

We conduct a structured comparison of the main studied frequent pattern extraction methods, paralleling their performances according to several relevant evaluation criteria. The objective is to identify the strengths and limitations of each approach within a distributed and performance-oriented framework.

Table 5 summarizes this analysis based on four fundamental axes: the adopted architecture (centralized or distributed), the execution time observed during the experiments, the quality of the extracted patterns (in terms of relevance and reduction of redundancy), as well as scalability, measured through the algorithms' ability to maintain good performance when increasing the number of computing nodes.

Table 5. Detailed Comparison of Frequent Pattern Mining Methods

Criterion	CD	PFP-Tree	AGFPM without pruning	AGFPM with pruning
Architecture	Peer-to-peer or centralized, with inter-node communication repeated.	Mainly decentralized, yet synchronization is costly at the stage of merging.	Master-Slave structure; one master collects full, unfiltered local trees.	Master-Slave with metadata fusion and pruning optimized; reduces communication and central load.
Execution Time	High runtime, particularly at low support, because of repeated scanning of the database.	Moderate, but the growth of trees is expensive for large datasets.	Faster than CD/PFP, but time increases with dataset size and complexity.	Fastest overall, due to early pruning of low-entropy branches.
Pattern Quality	Correct patterns, but may not be redundant or noisy.	Robust patterns, sensitive to support threshold.	Very high-accurate pattern coverage, but possibly with irrelevant or marginal patterns.	Maintains accurate pattern quality and removes redundant or low-interest patterns.
Scalability (with more nodes)	Limited scalability; synchronization overhead rises with the number of nodes.	Moderate scalability; local tree merging is dependent.	Good scalability, but it can be affected by memory use.	Very good scalability; pruning decreases global tree size and inter-node communication.

In light of this comparison, the AGFPM algorithm, particularly in its version incorporating an adaptive pruning mechanism, confirms its superiority. It stands out for its execution speed, the informative quality of the results produced, and its ability to effectively adapt to large-scale distributed environments, making it a robust and efficient alternative to classical methods.

5.4. DISCUSSION

The experiments conducted on the T10I4D100K, Kosarak, and Chess datasets in a distributed setting (3, 5, 7, and 9 nodes) provide a comprehensive assessment of the AGFPM algorithm in its pruned and unpruned variants. The evaluation focuses on two dimensions: execution time and the number of extracted patterns, each

examined under varying minimum support thresholds. This design enables a balanced analysis of scalability and result compactness across heterogeneous data regimes.

The results clearly demonstrate that the integration of a pruning mechanism based on binary entropy and quartiles offers a significant advantage to AGFPM. By filtering out less informative branches before the global merging phase, the algorithm significantly reduces the size of the global tree to be processed. This reduction results in a significant decrease in execution times, particularly noticeable when the number of nodes increases or when the support is low. In some cases, the observed processing time is reduced by 30 to 40% compared to classical approaches such as CD or PFP-tree.

From a quantitative perspective, pattern extraction follows an expected logic: lower support thresholds generate more frequent patterns. Without a filtering mechanism, the unpruned version of AGFPM systematically produces the largest number of patterns, which can burden the analysis process. On the other hand, the pruned version retains most of the informative patterns while eliminating those with low added value. For example, for a support of 4%, the difference in the number of patterns between the two AGFPM variants is relatively small, while the gain in computation time is significant.

The CD and PFP-tree algorithms, although effective in certain contexts, show their limitations here: their lack of adaptability to low thresholds and their inability to efficiently handle data density result in longer response times and more cumbersome pattern sets.

In terms of efficiency, scalability, and overall performance on three datasets, the AGFPM algorithm particularly in its pruning version, stands out as the most robust solution. Its Master-slave scheme, combined with an optimized bidirectional communication strategy, reduces synchronization overhead and avoids redundant calculations. Thanks to this design, the approach manages to extract globally frequent patterns in a targeted manner, while maintaining short execution times and excellent scalability. These results confirm that the AGFPM algorithm represents a significant advancement for the efficient and parallel exploration of frequent patterns in large-scale distributed environments.

6. CONCLUSION

In large-scale data mining, extracting frequent patterns from distributed databases is constrained by communication overhead and input/output coordination. To address these limitations, we propose AGFPM (Adaptive Global Frequent Pattern Mining), a master–slave distributed algorithm that integrates two complementary structures: the LP-Tree for local discovery and the GP-Tree for global consolidation. This design reduces inter-site communication and enables efficient detection of frequent patterns at scale. The GP-Tree supports systematic mining of globally frequent patterns by iteratively building conditional sub-GP-Trees across hierarchical levels. This recursive process examines candidate itemsets in a targeted manner, allowing the method to adapt to data and pattern complexity without incurring unnecessary computational or memory costs. To further enhance relevance and control the complexity of both LP-Tree and GP-Tree, AGFPM employs an adaptive pruning strategy that couples binary entropy with quartile-based thresholds. This mechanism automatically filters low-informative branches while preserving highly correlated patterns, improving the quality and compactness of results in distributed environments.

Empirical results show that AGFPM, augmented with the pruning mechanism, delivers superior scalability and runtime performance relative to benchmark methods, while preserving high fidelity in identifying globally frequent patterns. Future work will extend this framework to the discovery of distributed association rules within a fully decentralized architecture. The aim is to refine the methodology and design an optimized algorithm that builds on AGFPM's foundations to further improve performance in distributed computing settings.

7. REFERENCES

[1] H. Kargupta, C. Kamath, P. Chan, "Distributed and Parallel Data Mining: Emergence, Growth, and Future Directions", Advances in Distributed and Parallel Knowledge Discovery, AAAI/MIT Press, 2000, pp. 409-416.

[2] R. Agrawal, T. Imieliński, A. Swami, "Mining Association Rules Between Sets of Items in Large Databases", Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, USA, 25-28 May 1993, pp. 207-216.

[3] P.-N. Tan, M. Steinbach, V. Kumar, "Association Analysis: Basic Concepts and Algorithms", Introduction to Data Mining, Pearson Addison Wesley, 2005, pp. 327-386.

[4] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, Y.-K. Lee, "Efficient Single-Pass Frequent Pattern Mining Using a Prefix-Tree", Information Sciences, Vol. 179, No. 5, 2009, pp. 559-583.

[5] H. Huang, X. Wu, R. Relue, "Association Analysis with One Scan of Databases", Proceedings of the IEEE International Conference on Data Mining, Maebashi, Japan, 9-12 December 2002, pp. 629-632.

[6] G. Grahne, J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 10, 2005, pp. 1347-1362.

[7] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns Without Candidate Generation", Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 16-18 May 2000, pp. 1-12.

[8] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", Proceedings of the 20th

International Conference on Very Large Data Bases, Santiago de Chile, Chile, 12-15 September 1994, pp. 487-499.

[9] R. Agrawal, J. C. Shafer, "Parallel Mining of Association Rules", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, 1996, pp. 962-969.

[10] D. W. Cheung, V. T. Ng, A. W. Fu, Y. Fu, "Efficient Mining of Association Rules in Distributed Databases", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, 1996, pp. 911-922.

[11] D. W. Cheung, J. Han, V. T. Ng, A. W. Fu, Y. Fu, "A Fast Distributed Algorithm for Mining Association Rules", Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, USA, 18-20 December 1996, pp. 31-42.

[12] M. Z. Ashrafi, D. Taniar, K. Smith, "ODAM: An Optimized Distributed Association Rule Mining Algorithm", IEEE Distributed Systems Online, Vol. 5, No. 3, 2004, pp. 1-18.

[13] A. Schuster, R. Wolff, "Communication-Efficient Distributed Mining of Association Rules", ACM SIGMOD Record, Vol. 30, No. 2, 2001, pp. 473-484.

[14] E.-H. Han, G. Karypis, V. Kumar, "Scalable Parallel Data Mining for Association Rules", ACM SIGMOD Record, Vol. 26, No. 2, 1997, pp. 277-288.

[15] T. Shintani, M. Kitsuregawa, "Hash Based Parallel Algorithms for Mining Association Rules", Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, USA, 18-20 December 1996, pp. 19-30.

[16] L. Harada, N. Akaboshi, K. Ogihara, R. Take, "Dynamic Skew Handling in Parallel Mining of Association Rules", Proceedings of the 7th ACM International Conference on Information and Knowledge Management, Bethesda, MD, USA, 3-7 November 1998, pp. 76-85.

[17] A. Javed, A. Khokhar, "Frequent Pattern Mining on Message Passing Multiprocessor Systems", Distributed and Parallel Databases, Vol. 16, 2004, pp. 321-334.

[18] O. R. Zaïane, M. El-Hajj, P. Lu, "Fast Parallel Association Rule Mining Without Candidacy Generation", Proceedings of the IEEE International Conference on Data Mining, San Jose, CA, USA, 29 November - 2 December 2001, pp. 665-668.

[19] C. E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, Vol. 27, No. 3, 1948, pp. 379-423.

[20] T. M. Cover, J. A. Thomas, "Elements of Information Theory", 2nd Edition, Wiley, 2006.

[21] D. Freedman, R. Pisani, R. Purves, "Statistics", 4th Edition, W. W. Norton & Company, 2007.

[22] R. E. Walpole, R. H. Myers, K. Ye, "Probability and Statistics for Engineers and Scientists", 9th Edition, Pearson, 2011.

[23] IBM Almaden Research Center, "Quest Synthetic Data Generation Code and Datasets", http://cvs.buu.ac.th/mining/Datasets/synthesis_data/ (accessed: 2025)

[24] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V. S. Tseng, "The SPMF Open-Source Data Mining Library Version 2", Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, Riva del Garda, Italy, 19-23 September 2016, pp. 36-49.

[25] D. Dua, C. Graff, "UCI Machine Learning Repository", https://archive.ics.uci.edu/ (accessed: 2025)

[26] D. Fan, J. Wang, S. Lv, "Optimization of Frequent Item Set Mining Parallelization Algorithm Based on Spark Platform", Discover Computing, Vol. 27, No. 1, 2024, p. 38.

[27] M. Shaikh, S. Akram, J. Khan, S. Khalid, Y. Lee, "DIAFM: An Improved and Novel Approach for Incremental Frequent Itemset Mining", Mathematics, Vol. 12, No. 24, 2024, p. 3930.

[28] X. Sun, A. Ngueilbaye, K. Luo, Y. Cai, D. Wu, J. Z. Huang, "A Scalable and Flexible Basket Analysis System for Big Transaction Data in Spark", Information Processing & Management, Vol. 61, No. 2, 2024, p. 103577.

[29] S. Raj, D. Ramesh, P. Gantela, "CrossFIM: A Spark-Based Hybrid Frequent Itemset Mining Algorithm for Large Datasets", Cluster Computing, Vol. 28, 2025, pp. 231-245.

[30] Y. Rochd, I. Hafidi, "Frequent Itemset Mining in Big Data with Efficient Distributed Single Scan Algorithm Based on Spark", International Journal of Intelligent Engineering and Systems, Vol. 18, No. 2, 2025, p. 101908.

[31] A. Singla, P. Gandhi, "An Algorithm to Optimize Frequent Pattern Mining in Parallel and Distributed Environment", Engineering, Technology & Applied Science Research, Vol. 15, No. 3, 2025, pp. 22252-22256.

[32] T. Tassa, "Secure Mining of Association Rules in Horizontally Distributed Databases", IEEE Transactions on Knowledge and Data Engineering, Vol. 26, No. 4, 2014, pp. 970-983.

[33] R. Oliveira, O. R. Zaïane, "A Framework for Efficient Distributed Mining of Association Rules", Proceedings of the 2nd SIAM International Conference on Data Mining, Arlington, VA, USA, 11-13 April 2002, pp. 1-11.

[34] V. S. Tseng, C. H. Lin, Y. Y. Lin, C. W. Chen, "A Scalable Approach to Mining Frequent Itemsets over Big Data", Expert Systems with Applications, Vol. 42, No. 21, 2015, pp. 7876-7888.