STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **x**, Month 2025, pp 0–17.
Published online in International Academic Press (www.IAPress.org)

OGA-Apriori: An Optimized Genetic Algorithm Approach for Enhanced Frequent Itemset Mining

BARIK Meryem*, TOULAOUI Abdelfattah, HAFIDI Imad, ROCHD Yassir

Laboratory of Process Engineering, Computer Science and Mathematics (LIPIM), University Sultan Moulay Slimane Khouribga, Morocco

Abstract Frequent Itemset Mining (FIM) can be broadly categorized into two approaches: exact and metaheuristic-based methods. Exact approaches, such as the classical Apriori algorithm, are highly effective for small to medium-sized datasets. However, these methods face significant temporal complexity when applied to large-scale datasets. However, while capable of addressing larger datasets, metaheuristic-based approaches often struggle with precision. To overcome these challenges, researchers have explored hybrid methods that integrate the recursive properties of the Apriori algorithm with various metaheuristic algorithms, including Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). This integration has led to the development of two prominent techniques: GA-Apriori and PSO-Apriori. Empirical evaluations across diverse datasets have consistently shown that these hybrid techniques outperform the traditional Apriori algorithm in both runtime and solution quality. Building upon this foundation, this study introduces an enhanced version of the GA-Apriori algorithm, Optimized GA-Apriori (OGA-Apriori), to improve runtime efficiency and solution accuracy. Comprehensive evaluations on multiple datasets demonstrate that the proposed OGA-Apriori approach achieves superior performance compared to the original GA-Apriori in both runtime and solution effectiveness.

Keywords Data Mining, Frequent Itemsets Mining, Apriori Algorithm, Genetic Algorithm, Particle Swarm Optimization, Metaheuristic.

DOI: 10.19139/soic-2310-5070-2320

1. Introduction

Frequent Itemset Mining (FIM) approaches can be broadly categorized into exact methods and metaheuristic-based methods. Exact approaches include algorithms such as Apriori [1], FP-Growth [2], DIC [3], and AIS [4]. These algorithms aim to extract all frequent itemsets from a database, but their execution time significantly increases due to multiple scans of the entire transactional dataset. The performance of exact algorithms deteriorates with the growth in database size, the number of items and transactions, and the exponential increase in generated candidate itemsets, rendering them inefficient and impractical for large-scale datasets.

To address the performance limitations of exact methods, metaheuristic-based approaches have been proposed, such as those employing genetic algorithms (GA) [5, 6] and swarm intelligence techniques [7, 8]. Meta-heuristic-based methods aim to extract a subset of frequent itemsets within a shorter execution time. However, these approaches cannot guarantee the discovery of all possible frequent itemsets in the database. As a result, the quality of the solutions produced by metaheuristic-based approaches is generally inferior to the optimal solutions provided by exact methods, which enumerate all frequent itemsets.

The efficiency of metaheuristic-based FIM methods depends on the strategy used for randomized exploration of the itemset search space. A critical limitation in the literature is the insufficient utilization of the inherent

*Correspondence to: BARIK Meryem (Email: meryem.barik@usms.ac.ma). Laboratory of Process Engineering, Computer Science and Mathematics (LIPIM), University Sultan Moulay Slimane Khouribga, Morocco.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

properties of the FIM solution space to improve the search process. One of the most significant properties, which is fundamental to the Apriori algorithm, is the recursive nature of frequent itemsets. This property ensures that if an itemset of size k is frequent, all its sub-itemsets of size s={1,...,k-1} are also frequent. Although this recursive property is central to exact methods, it is often overlooked in metaheuristic-based approaches.

However, several metaheuristic-based FIM methods in the literature incorporate this recursive property to enhance search efficiency. Notable examples include GA-Apriori [9, 10, 11] and PSO-Apriori [10, 11], which leverage genetic algorithms and particle swarm optimization, respectively. These approaches build upon their predecessors, GAFIM [5] and PSOFIM [8], by introducing novel search space intensification and diversification operators. For example, GA-Apriori employs advanced crossover and mutation operators, while PSO-Apriori uses refined particle positioning and velocity adjustments. Both techniques incorporate the recursive property of frequent itemsets to improve performance and solution quality.

Despite the advancements introduced by GA-Apriori, there remains room for further optimization, particularly in addressing issues related to runtime efficiency and solution quality. This study proposes an enhanced algorithm, OGA-Apriori (Optimized GA-Apriori), which builds upon GA-Apriori by introducing improved genetic operators and search strategies. These enhancements aim to exploit the recursive property of frequent itemsets more effectively and balance search space exploration and exploitation. Experimental evaluations on diverse datasets demonstrate that OGA-Apriori consistently outperforms GA-Apriori regarding both runtime and solution quality, thereby advancing the state-of-the-art in metaheuristic-based FIM methods.

The structure of this paper is as follows: Section 2 reviews related work, Section 3 introduces the proposed approach, Section 4 presents experimental results, Section 5 introduced Statistical assessment, and Section 6 concludes the study.

2. Background and retaled work

2.1. Frequent Pattern Mining

Frequent Itemset Mining (FIM) is an essential part of Data Mining. FIM's goal is to discover all the frequent itemset from a massive dataset, i.e., if an itemset or several itemsets occurs/seem several times in a Data, they are interesting, this is according to a minimum frequency threshold given by a user. This activity was developed in the early 1990s by Agrawal et al. [1] to identify frequently co-occurring items in market basket analysis. In this section, formal definitions associated with this task are explained.

2.1.1. Pattern A pattern is a collection of elements, events, or things that often appear in a database. Formally, a pattern P in a database D is defined as a subset of elements P $\subseteq \{i_1,..., i_n\} \in$ D that explain important data characteristics.

2.1.2. Support In database D, the set of transactions that contain a pattern x is called its coverage. The support of a pattern x, denoted *sprt(x, D)* is the number of transactions that contain x, i.e., the cardinal of its coverage.

$$sprt(x, D)_{abs} = |t_k \in D/x \subseteq t_k| \tag{1}$$

The equation gives the absolute value of the support (a positive integer less than or equal to the size of the database), which can also be expressed in relative terms, i.e., by a real number between zero and one, representing a percentage by dividing it by the size of the data set as shown in the formula below.

$$sprt(x, D)_{abs} = \frac{|t_k \in D/x \subseteq t_k|}{|D|} \tag{2}$$

In order to lighten the writing, we simply note it *sprt(x)* without any abs sub-mention or reference to a transaction base that will be understood from the context.

2.1.3. Frequent Pattern A pattern X is frequent if and only if its support exceeds a minimal threshold s specified in advance. Formally, X is frequent in D relative to the threshold s if and only if: *sprt(X)* $>$ s.
The problem of pattern mining consists in enumerating the set F_D(s) of all frequent patterns derived from A present in the transaction base D with respect to the minimal threshold of support s. To simplify the notation, this last set will, from now on, be noted as F for short.

$$F_D(s) = x \in /sprt(x) \geq s \tag{3}$$

2.2. Related work on metaheuristic-based FIM methods

2.2.1. Exact methods This led to the development of numerous exact algorithms beyond the foundational Apriori. Early approaches include AIS [1], DHP (Dynamic Hashing and Pruning) [12], and DIC (Dynamic Itemset Counting) [3], all of which were based on or inspired by the principles established by Apriori. Additionally, FPGrowth, introduced by Han.et.al. [2], leverages a compact data structure known as the FP-Tree (Frequent Pattern Tree), a prefix-tree-like structure enhanced with support information to minimize database scans. During its mining process, FPGrowth identifies patterns along each path from the root to a node and records items along with their support counts.

Subsequent advances have focused on improving efficiency and scalability. Eclat (*Equivalence Class Clustering and bottom-up Lattice Traversal*) [13] employs a vertical data layout, associating each itemset with a set of transaction IDs (TIDs). By intersecting these TID sets, Eclat efficiently calculates the support of candidate itemsets, thereby reducing memory overhead compared to algorithms that directly materialize Apriori. CHARM (*Closed Association Rule Mining*) [14] extends vertical mining by efficiently discovering *closed* itemsets. It optimizes the search space through the use of "diffsets," which store differences between TID sets, significantly speeding up the search for frequent patterns. PrePost [15] introduces a unique "preorder" and "postorder" coding strategy within a pattern-growth framework, limiting the generation of redundant candidates. Nevertheless, when dealing with extremely large databases, these exact methods can become prohibitively expensive in both time and space. The exponential expansion of data from various sources has produced vast datasets [16], underscoring the need for more efficient solutions.

To address this challenge, recent research has turned toward distributed and parallel algorithms, capitalizing on frameworks such as Apache Hadoop and Spark. On Hadoop, notable contributions include a parallel implementation of PrePost [17], an enhanced Apriori algorithm that exploits a hybrid row-/column-oriented layout [18], and a performance-improved variant of PrePost [19]. A comprehensive survey of scalable frequent-itemset algorithms for big data on both Hadoop and Spark is provided in [20]. Leveraging Spark's in-memory processing, researchers have proposed a Spark-native parallel PrePost algorithm [21] and an efficient distributed FIM algorithm tailored to big-data workloads [22]. These studies collectively highlight the importance of scalability and demonstrate that careful data layout, in-memory computation, and parallel pattern-growth strategies can substantially reduce execution time while maintaining exactness in frequent itemset mining.

2.2.2. Metaheuristic-based methods In metaheuristic-based intelligence approaches, we distinguish between genetic algorithms and swarm-based methodologies.

For genetic algorithm-based methods, we have GAR [23], which is the first genetic algorithm-based technique for mining association rules and frequent itemsets. A known drawback of GAR is its inefficient individual representation, where individuals are encoded by the number of items they contain, causing variable-length chromosomes that hinder the efficiency of crossover and mutation operators. Two subsequent methods, AGA [24] and ARMGA [25], differ primarily in their genetic operators: ARMGA employs a two-point crossover, whereas AGA uses a simple crossover. Another alternative is G3PARM [26], which uses grammar-guided genetic programming (G3P) to reduce the likelihood of generating incorrect itemsets.

The GAFIM algorithm, introduced by [5], stands out for its deletion and decomposition mechanism that splits infrequent itemsets into pairs of frequent itemsets. More recently, [11] proposed the GA-GD approach, which is a new framework of the GA-Apriori [10, 9]. This approach integrates the recursive property of the Apriori algorithm (i.e., the downward closure principle) with genetic operators such as crossover, mutation, and selection to enhance

the exploration of larger frequent itemsets while pruning non-promising candidates. Although GA-Apriori retains the fundamental genetic operators typical of frequent itemset mining (FIM) techniques, it differs from GAFIM in its initialization strategy, as well as in the specific design of its crossover, mutation, and selection operators. The general workflow of GA-Apriori is illustrated in Figure 1.

Recent findings show that swarm intelligence techniques—including ACO (Ant Colony Optimization), PSO (Particle Swarm Optimization) and Bee-based algorithms—offer promising solutions for data mining tasks such as feature selection, clustering, and frequent itemset mining (FIM) [27]. Although approaches like ACO-based FIM [28] (for example, HUIM-ACS) can be effective, they often face runtime efficiency challenges. Meanwhile, PSO-FIM [8] and its variants aim to balance intensification and diversity in their searches. Bee-based FIM methods explore neighborhoods through dance tables and communication between bees, while BATFIM [29] uses the bat meta-heuristic, reportedly outperforming other evolutionary and swarm-based FIM algorithms.

Complementing these developments, [30] present a comprehensive Review of Heuristic Algorithms for the Frequent-Itemset-Mining Problem , and subsequently introduce a dedicated Heuristic Algorithm for Extracting Frequent Patterns in Transactional Databases [31], both of which demonstrate the continued relevance of heuristic search strategies alongside evolutionary and swarm-based methods.

More recently, PSO-GD [11] and BSO-GD [11] have incorporated the recursive Apriori principle to enhance the position updates of particles and bees, respectively. PSO-GD [11], in particular, constitutes a new framework of the PSO-Apriori [10] approach, leveraging the synergy between swarm intelligence and the downward-closure property. Meanwhile, studies continue to emphasize the superior runtime efficiency and solution consistency of both genetic algorithms and PSO-based methods over other bioinspired approaches, although their randomized search mechanisms can occasionally result in suboptimal accuracy under certain conditions.

Apart from FIM, feature selection in classification can benefit from association rule mining, as shown in a network intrusion detection system [32]. Using fuzzy ARTMAP and a gravitational search-optimized neural network, the approach reduces duplicate input, leading to improved detection rates, lower false alarms, and significant computational savings (over 8.4%). This integration of association rule mining for feature selection underscores the broad applicability of bioinspired and evolutionary methods in data mining.

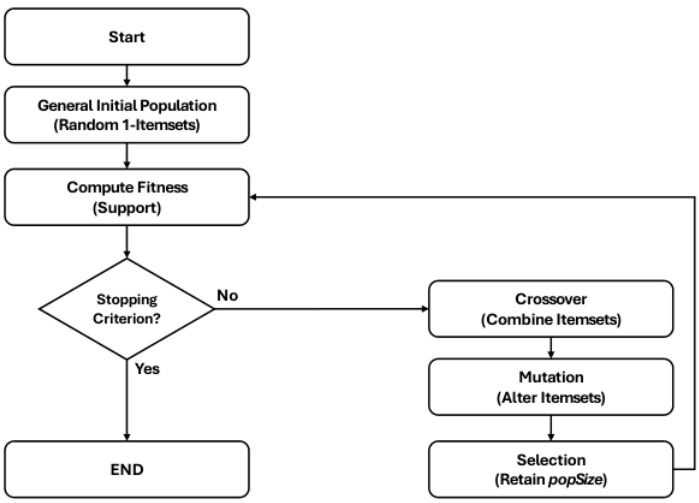

Figure 1. Genetic algorithm workflow.

3. Proposed approach

The proposed approach **OGA-Apriori**, illustrated in the Figure 2, is an improvement of the **GA-Apriori** [9, 10] approach existing in the literature. The primary distinction between **GA-Apriori** and **OGA-Apriori** lies in how they initialize their populations of candidate itemsets. *GA-Apriori* typically starts with a random *popSize* set of 1-itemsets, which can include many low-support or marginal itemsets that slow convergence. This random selection forces the algorithm to expend computational effort eliminating non-promising candidates over multiple iterations. In contrast, *OGA-Apriori* (Optimized GA-Apriori) replaces this random initialization with a **support-based** selection of 1-itemsets: it calculates the support of all possible 1-itemsets, ranks them in descending order, and retains only the top *popSize*. By beginning with itemsets that already demonstrate relatively high support, OGA-Apriori focuses its genetic search on more promising regions of the solution space right from the outset, leading to **faster convergence** and **reduced overhead**. Despite this modified initialization step, OGA-Apriori preserves the fundamental evolutionary operators—crossover, mutation, and selection—of GA-Apriori, thereby retaining the benefits of genetic optimization while mitigating the drawbacks of a randomly seeded population.

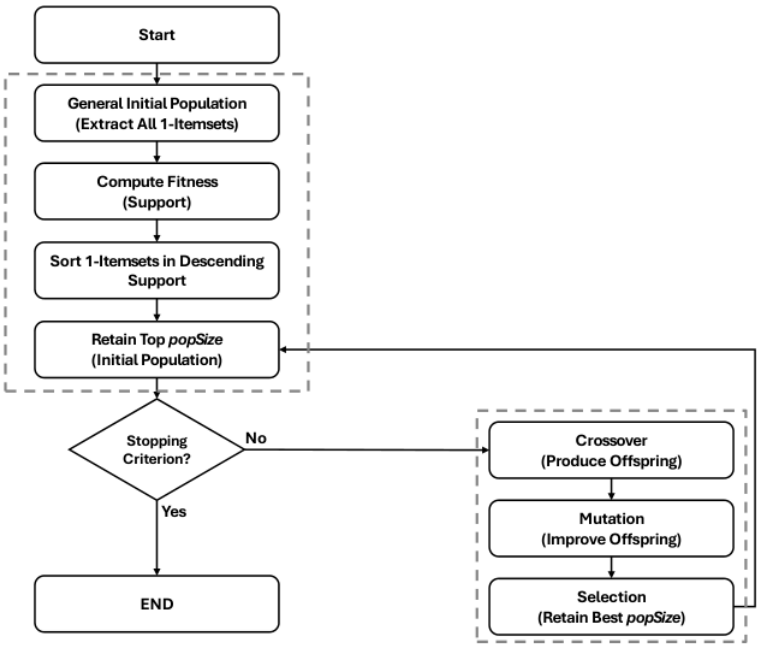

Figure 2. Optimized Genetic algorithm workflow

Algorithm 1 outlines the main steps of the **OGA-Apriori** method. Given a transactional database D as input—used to compute the support of generated itemsets—and a user-defined minimum support threshold minSup, OGA-Apriori proceeds as follows. First, it identifies frequent 1-itemsets by invoking the *FindFrequentOneItemset()* function, which computes and sorts all single items by their support in descending order, then retains only the top *popSize* items as the *initial population*. These frequent 1-itemsets are also added to the global set of frequent itemsets, denoted by F.

Next, the algorithm applies the *Crossover* operator to each pair of parent itemsets in the current population, appending any resulting itemsets to a temporary population. Subsequently, it employs the *Mutation* operator to transform any infrequent itemsets into frequent ones, adding newly discovered frequent itemsets to F. Afterward, the *Selection* procedure is executed, which filters and ranks the itemsets, preserving only the top *popSize* frequent itemsets to serve as the *new population* in the subsequent iteration.

This iterative process of crossover, mutation, and selection—supported by the Apriori-based pruning of infrequent subsets—continues until the current population becomes empty. At that point, **OGA-Apriori** terminates, having collected a comprehensive set F of frequent itemsets.

Algorithm 1 OGA-Apriori

Input: Transactional database D, minimum support minSup, population size *popSize*.
Output: Set F of frequent itemsets.
$F \leftarrow \emptyset$
population $\leftarrow$ FINDFREQUENTONEITEMSET(D, *minSup*, *popSize*)
$F \leftarrow F \cup$ *population*
while *population* is not empty **do**
  tempPop $\leftarrow$ CROSSOVER(*population*)
  newPop $\leftarrow$ MUTATION(*tempPop*, D, minSup)
  $F \leftarrow F \cup$ (frequent itemsets from *newPop*)
  population $\leftarrow$ SELECTION(*newPop*, *popSize*)
end while
return F

In what follows, we describe in detail the process of our approach.

3.1. Population Initialization:

Let $\mathcal{I}$ be the universal set of items in the dataset D. For each item $x \in \mathcal{I}$, define the *support* function as

$$\text{support}(x) \;=\; \frac{\big|\{\, t \in D : x \in t\}\big|}{|D|},$$

where each t is a transaction in D. Let minSup be the minimum support threshold. We call x *frequent* if $\text{support}(x) \geq \text{minSup}$.

1. **Compute All 1-Itemsets.** Construct the set of singleton itemsets $\{\{x\} : x \in \mathcal{I}\}$ and calculate $\text{support}(x)$ for all x.
2. **Prune Infrequent Items.** Remove each item x for which $\text{support}(x) < \text{minSup}$.
3. **Sort and Retain Top *popSize*.** Sort the remaining items in *descending order* of $\text{support}(x)$. Let *popSize* be a positive integer. The *initial population* is then the top *popSize* frequent singleton itemsets.

By selecting only the *most frequent* items at the outset, OGA-Apriori restricts the *genetic search* to a more promising subset of items, thereby *accelerating* convergence compared to random initialization approaches.

Example:

Let $\mathcal{I} = \{\, a, b, c, d\}$ and suppose the dataset D yields the following support values:

$$\text{support}(a) = 0.4, \quad \text{support}(b) = 0.5, \quad \text{support}(c) = 0.1, \quad \text{support}(d) = 0.3.$$

Given $\text{minSup} = 0.3$, the frequent items are

$$\{\, a,\ b,\ d\}, \quad \text{since} \quad \text{support}(a) \geq 0.3,\ \text{support}(b) \geq 0.3,\ \text{support}(d) \geq 0.3.$$

If we set *popSize* $= 2$, we retain only the top two items by descending support:

$$b\ (\text{support}(b) = 0.5), \quad a\ (\text{support}(a) = 0.4).$$

Hence, the *initial population* is

$$\{\, \{a\},\ \{b\}\}.$$

Representing these as 4-bit vectors (x_a, x_b, x_c, x_d), where $x_x = 1$ if item x is present and 0 otherwise, we obtain:

$$\{a\} \longrightarrow (1,\,0,\,0,\,0), \quad \{b\} \longrightarrow (0,\,1,\,0,\,0).$$

All other items ($\{c\}$ and $\{d\}$) are excluded from the initial population because either their support is below minSup or they do not rank within the top *popSize*.

3.2. Crossover

Let $I = \{\, x_1, x_2, \ldots, x_n\}$ be the universal set of items in a dataset D. Represent each *itemset* of size k as a binary vector

$$\mathbf{c} = (c_1, c_2, \ldots, c_n),$$

where

$$c_i = \begin{cases} 1, & \text{if } x_i \in \textit{itemset}, \\ 0, & \text{otherwise.} \end{cases}$$

Suppose we have two *parents* of size $(k-1)$,

$$\mathbf{p}_1 = (p_{1,1}, p_{1,2}, \ldots, p_{1,n}), \quad \mathbf{p}_2 = (p_{2,1}, p_{2,2}, \ldots, p_{2,n}),$$

each of which is assumed to be frequent under the Apriori downward-closure property. We seek to produce two *child* vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ of size k using the following crossover steps:

1. **Parent Selection:** Identify the two most frequent $(k-1)$-itemsets $\mathbf{p}_1$ and $\mathbf{p}_2$ in the current population. This selection can be based on *support*, *fitness*, or another metric.
2. **Initial Copy:**
$$\mathbf{c}_1 \leftarrow \mathbf{p}_1, \quad \mathbf{c}_2 \leftarrow \mathbf{p}_2.$$
i.e., $\mathbf{c}_1$ initially replicates $\mathbf{p}_1$, and $\mathbf{c}_2$ initially replicates $\mathbf{p}_2$.
3. **Selective Insertion:** Let e_1 be the most frequent item index such that
$$p_{2,e_1} = 1 \quad \text{and} \quad p_{1,e_1} = 0.$$
Then set
$$c_{1,e_1} = 1 \quad (\text{i.e., insert } x_{e_1} \text{ into child } 1).$$
Similarly, let e_2 be the most frequent item index such that
$$p_{1,e_2} = 1 \quad \text{and} \quad p_{2,e_2} = 0.$$
Then set
$$c_{2,e_2} = 1 \quad (\text{i.e., insert } x_{e_2} \text{ into child } 2).$$

By selecting *frequent* items e_1 and e_2 for insertion, we generate potentially larger frequent itemsets (of size k) in line with the Apriori heuristic.

Example:

Consider a dataset D of four items $\{\, a, b, c, d\}$. Define the corresponding 4-bit vector (p_1, p_2, p_3, p_4) in the order (a, b, c, d). Suppose the two most frequent parents of size $(k-1) = 2$ are

$$\mathbf{p}_1 = (1,\,1,\,0,\,0) \quad \longleftrightarrow \quad \{a, b\},$$

$$\mathbf{p}_2 = (0,\,0,\,1,\,1) \quad \longleftrightarrow \quad \{c, d\}.$$

Let us choose:

$$e_1 = c \quad (\text{index } 3), \quad e_2 = a \quad (\text{index } 1),$$

based on their respective support or frequency in the parents.

1. **Initial Copy:**
$$\mathbf{c}_1 \leftarrow \mathbf{p}_1 = (1, 1, 0, 0), \quad \mathbf{c}_2 \leftarrow \mathbf{p}_2 = (0, 0, 1, 1).$$
2. **Selective Insertion:**
   - Since $p_{2,3} = 1$ and $p_{1,3} = 0$, set $c_{1,3} = 1$, making
   $$\mathbf{c}_1 = (1, 1, 1, 0) \longleftrightarrow \{a, b, c\}.$$
   - Since $p_{1,1} = 1$ and $p_{2,1} = 0$, set $c_{2,1} = 1$, making
   $$\mathbf{c}_2 = (1, 0, 1, 1) \longleftrightarrow \{a, c, d\}.$$

Hence, the two child itemsets of size $k = 3$ are

$$\mathbf{c}_1 = \{a, b, c\} \quad \text{and} \quad \mathbf{c}_2 = \{a, c, d\}.$$

This procedure illustrates how the crossover operator extends frequent $(k-1)$-itemsets to generate *candidate* k-itemsets, while adhering to the Apriori principle (downward closure) by focusing on items shown to be frequent in at least one parent.

3.3. Mutation

Let $I = \{x_1, x_2, \ldots, x_n\}$ be the universal set of items in the dataset D. Suppose we represent any k-itemset as an n-dimensional binary vector

$$\mathbf{m} = (m_1, m_2, \ldots, m_n),$$

where

$$m_i = \begin{cases} 1, & \text{if } x_i \in \text{itemset}, \\ 0, & \text{otherwise.} \end{cases}$$

Goal: Given an *infrequent* k-itemset $\mathbf{m}$, we aim to *transform* it into a *frequent* k-itemset (if possible) by selectively flipping bits in $\mathbf{m}$.

1. **Identify Infrequent Itemsets.**
   - After *crossover*, gather all newly generated itemsets $\{\mathbf{m}_1, \mathbf{m}_2, \ldots\}$ of size k.
   - For each $\mathbf{m}_i$, check if $\mathbf{m}_i$ is *infrequent*, i.e.
   $$\text{support}(\mathbf{m}_i) \ < \ \text{minSup}.$$
   - If $\mathbf{m}_i$ is *frequent*, no mutation is applied.
2. **Bit-Flipping Mutation.**
   - Let $\mathbf{m} = (m_1, m_2, \ldots, m_n)$ be an *infrequent* k-itemset.
   - *Select* two indices e_1 and e_2 such that
   $$m_{e_1} = 1 \quad \text{and} \quad m_{e_2} = 0.$$
   - *Swap* their bit values:
   $$m_{e_1} \leftarrow 0, \quad m_{e_2} \leftarrow 1.$$
   - In effect, item x_{e_1} is *removed* from $\mathbf{m}$, and item x_{e_2} is *added*.
3. **Check Support.**
   - Recompute or update support$(\mathbf{m})$.
   - If $\mathbf{m}$ is still infrequent and a maximum number of bit-flips is not reached, *repeat* the bit-flipping step.
   - If $\mathbf{m}$ becomes *frequent*, the mutation step *terminates* for that itemset.

This mutation strategy attempts to replace items contributing little to support with those that might improve support, with the goal of producing a *frequent* k-itemset.

Example:

Let $I = \{\, a, b, c, d\}$, and index each item in a 4-bit vector as (m_1, m_2, m_3, m_4) corresponding to (a, b, c, d). Consider an *infrequent* itemset $\mathbf{m} = (1,\ 0,\ 1,\ 0)$, representing $\{a, c\}$. Suppose we identify:

$$e_1 = 1 \quad (\text{item } a,\ m_1 = 1), \quad e_2 = 2 \quad (\text{item } b,\ m_2 = 0).$$

1. **Before Mutation:**
$$\mathbf{m} = (1,\ 0,\ 1,\ 0) \quad \longleftrightarrow \quad \{\, a,\ c\}.$$
Assume $\{a, c\}$ is *infrequent*.
2. **Bit Flip:**
$$m_1 \leftarrow 0 \quad (\text{remove } a), \quad m_2 \leftarrow 1 \quad (\text{add } b).$$
Hence,
$$\mathbf{m} = (0,\ 1,\ 1,\ 0) \quad \longleftrightarrow \quad \{\, b,\ c\}.$$
3. **Check Support:**
   - If $\{b, c\}$ is now *frequent*, mutation *stops* immediately.
   - Otherwise, the process may repeat with different bit-flips until $\mathbf{m}$ becomes frequent (or a mutation limit is reached).

This example illustrates how *Mutation* attempts to salvage an infrequent itemset by strategically swapping bits to replace unhelpful items with potentially more frequent ones.

3.4. Selection

After *crossover* and *mutation*, let

$$\mathcal{C} = \{\, C_1, C_2, \ldots, C_m\}$$

be the set of newly generated k-itemsets. We define the following steps to select the *best* frequent itemsets from $\mathcal{C}$ and retain them for the next iteration.

1. **Filtering by Frequency:** Let minSup be the minimum support threshold. We form the subset $\mathcal{F} \subseteq \mathcal{C}$ of *frequent* itemsets as
$$\mathcal{F} \ = \ \Big\{ C_i \in \mathcal{C} \ : \ \text{support}(C_i) \ \geq \ \text{minSup} \Big\}.$$
Itemsets not meeting this condition are *discarded*.
2. **Fitness Computation:** For each frequent itemset $C_i \in \mathcal{F}$, define the *fitness* score
$$\varphi(C_i) = \text{support}(C_i).$$
The fitness function directly uses the support value, ensuring that itemsets with higher support are favored.
3. **Retain Top *popSize*:** Let $\mathcal{F}$ be sorted in *descending order* by $\varphi(\cdot)$. If we denote *popSize* by p, then the new population $\mathcal{P}_{\text{next}}$ is defined as the top p itemsets in $\mathcal{F}$:
$$\mathcal{P}_{\text{next}} \ = \ \arg\operatorname{top}_{C \in \mathcal{F}} \big(\varphi(C),\ p\big).$$
Hence, we select exactly the p frequent itemsets with the highest support values.
4. **Goal:** By choosing the *best* frequent itemsets and discarding weaker ones, the algorithm ensures that the most promising solutions form the *new population*, to be *evolved* in subsequent iterations.

3.5. Time and Space Complexity of OGA-Apriori

The overall running time of OGA-Apriori is dominated by the repeated evaluation of candidate itemsets against the database in each genetic generation. Let $|D|$ be the number of transactions, m the average transaction length, p the population size, and G the number of generations until convergence. The initial scan to find all frequent 1-itemsets takes $\mathcal{O}(|D| \cdot m)$. Each generation then performs $\mathcal{O}(p)$ crossover operations (negligible) and $\mathcal{O}(p)$ support-counting "mutations," each of which—using a straightforward scan—costs $\mathcal{O}(|D| \cdot m)$. Thus one generation costs $\mathcal{O}(p \cdot |D| \cdot m)$, and over G generations the worst-case time complexity is $\mathcal{O}(G \cdot p \cdot |D| \cdot m)$. In practice, G is bounded by the point at which no new frequent itemsets are discovered, but in the worst case it may grow until all possible combinations are explored.

In terms of space, OGA-Apriori must store the transactional database ($\mathcal{O}(|D| \cdot m)$), the evolving population of up to p candidate itemsets ($\mathcal{O}(p \cdot L)$, with L the maximum itemset length), and the set F of all discovered frequent itemsets (which in the worst case could be exponential in the number of distinct items, although genetic pruning usually keeps it much smaller). Aside from the database itself, the in-memory footprint scales linearly with the population size and average itemset length.

4. Experimental results

Several experiments have been performed to evaluate the OGA-Apriori algorithm. The algorithms have been implemented in Python and the experiments were performed under Windows 10 using a laptop equipped with an Intel I5 processor and 8 GB of memory. In the next part, we compare the performance of the OGA-Apriori algorithm with that of the GA-Apriori algorithm, using real instances of databases frequently used for benchmarking FIM search.

4.1. Description of datasets

To evaluate the performance of our proposed method, we conducted experiments on well-known benchmark datasets commonly used in the field of data mining and frequent pattern extraction. These datasets were obtained from an open-source data mining library[†] and include diverse characteristics in terms of the number of transactions, the number of items and the average transaction size. The details of the datasets are summarized in Table 1.

Table 1. Data instances description

Instance name	N. of transactions	N. of items	Avg. size of transactions
Chess	3 196	75	37
Mushrooms	8 416	119	23
c20d10k	10 000	192	20
c73d10k	10 000	1 592	73
connect	67 557	129	43
accidents	340 183	468	33.8
RecordLink	574 913	29	10
kddcup99	1 000 000	135	16
PAMAP	1 000 000	141	23.93
PowerC	1 040 000	140	7

The selected datasets cover a wide range of application domains, ensuring a comprehensive evaluation of our approach. For instance, the Mushrooms dataset contains information about different mushroom species and their

[†]https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

attributes, while Chess and Connect are structured datasets commonly used in game-related analysis. The Accidents dataset, which records road accidents, provides a real-world scenario with a large number of transactions and high-dimensional itemsets.

Additionally, we included large-scale datasets such as KDDCup99, PAMAP, RecordLink, and PowerC, which consist of millions of transactions and varying itemset sizes, allowing us to assess the scalability of our method. The c20d10k and c73d10k datasets, known for their high item dimensionality, present additional challenges related to data sparsity and computational efficiency.

By testing these diverse datasets, we aim to provide a robust evaluation of our proposed approach, ensuring its effectiveness across different data distributions, transaction sizes, and itemset complexities.

4.2. Parameters setting

In this paper, we considered the following FIM algorithm: Apriori [1], GAFIM [5], GA-Apriori [9, 10, 11], PSO-Apriori [10, 11], and the proposed approach OGA-Apriori.

We ran a benchmark experiment using the previous data instances to determine the population's size. We consider the average support of frequent itemsets discovered to choose the population's size concerning the runtime. The results of the tests are presented as follows:

- **GAFIM:** Through experimentation, we varied the population size from 10 to 1000, the number of generations from 10 to 100, and adjusted the crossover and mutation probabilities across the full range (0% to 100%). The optimal configuration identified consists of a population size of 80 individuals, a maximum of 20 generations, with crossover and mutation probabilities set at 50% and 80%, respectively.
- **GA-Apriori:** By exploring population sizes ranging from 10 to 1000 and setting the maximum number of iterations between 20 and 100, the optimal configuration was determined to be 80 individuals with a maximum of 40 iterations.
- **PSO-Apriori:** By exploring population sizes ranging from 10 to 1000 and setting the maximum number of iterations between 20 and 100, the optimal configuration was determined to be 80 individuals with a maximum of 40 iterations.
- **OGA-Apriori:** By varying the population size from 10 to 1000 and the number of iterations from 20 to 100, the optimal configuration was identified as 80 individuals and a maximum of 40 iterations.

For each algorithm, we conducted 10 independent runs on each dataset and reported the average runtime to account for stochastic variability. To determine the optimal population size, we performed additional experiments by varying the popSize parameter from 10 to 1000 (keeping other parameters fixed) and plotted the average runtime against popSize Figure 4.

The Apriori algorithm employs exhaustive search strategies to identify all frequent itemsets that meet the specified minimum support threshold. Notably, these algorithms do not require the tuning of additional parameters beyond the support constraint.

In all experimental evaluations, the minimum support (MinSup) was uniformly applied across all algorithms to ensure a fair and consistent comparison.

4.3. Experimental results

To evaluate the effectiveness of our proposed approach, OGA-Apriori, we conducted an extensive experimental study using ten benchmark datasets of varying sizes and complexities. These datasets, as described in Table 1, differ in the number of transactions, number of items, and average transaction length, allowing for a comprehensive assessment of scalability, efficiency, and accuracy. We compared OGA-Apriori with three other frequent item mining algorithms, Apriori, GAFIM, PSO-Apriori, and GA-Apriori, using two key performance indicators: runtime and the percentage of frequent item sets discovered.

The runtime analysis in Table 2 reveals that the Apriori algorithm consistently exhibits the highest computational cost, particularly on large-scale or high-dimensional datasets such as Kddcup99 (1,000,000 transactions, 135 items), PAMAP (1,000,000 transactions, 141 items) and PowerC (1,040,000 transactions). This is due to Apriori's exhaustive candidate generation and repeated database scans. Although GAFIM offers improved performance

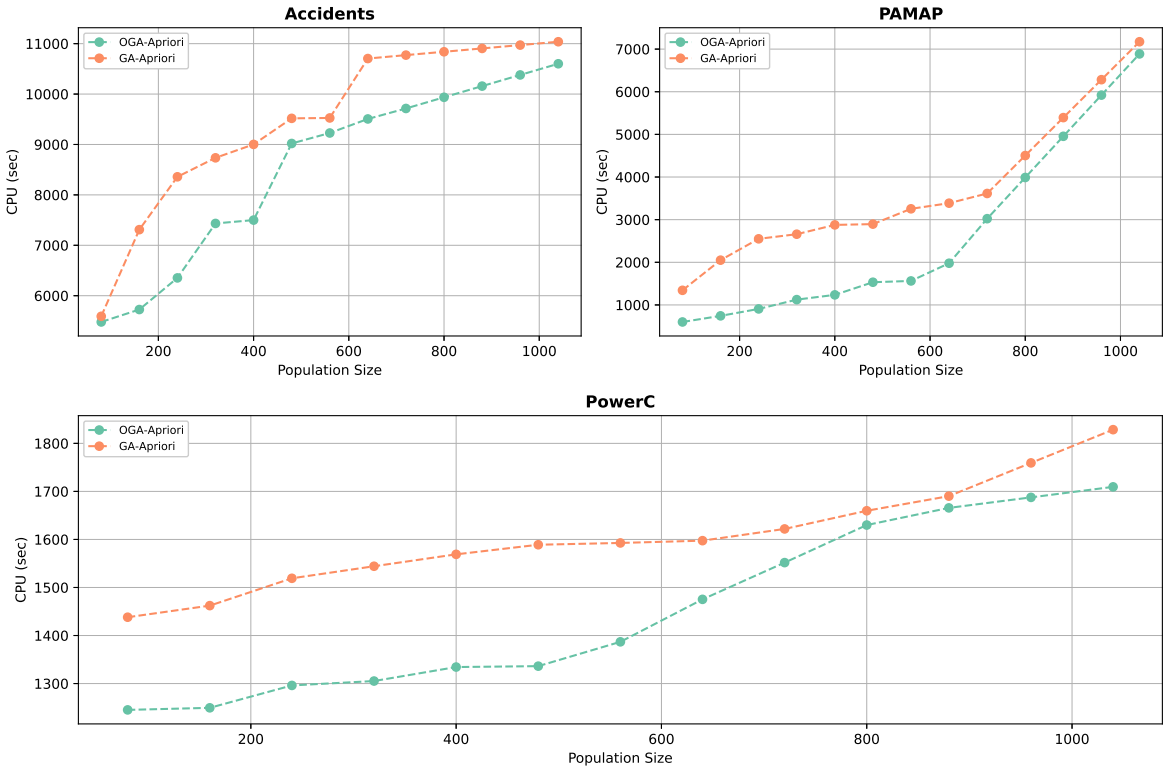

Figure 3. Runtime (Sec) vs Population Size for GA-Apriori and OGA-Apriori on Accidents, PAMAP & PowerC

Table 2. Comparison of Runtime (in seconds) between Apriori, GAFIM, GA-Apriori, and OGA-Apriori on various datasets

Dataset	**Apriori**	**GAFIM**	**GA-Apriori**	**OGA-Apriori**	**PSO-Apriori**
Chess	15.96	10.90	9.03	6.11	6.50
Mushrooms	61.31	23.74	10.01	9.88	10.92
c20d10k	200.95	100.20	35.81	32.11	36.45
c73d10k	3089.07	2875.95	2676.13	2274.02	2386.8
Connect	1549.87	1402.63	909.83	788.69	790.65
Accidents	3989.79	2596.07	1364.73	760.31	762.46
RecordLink	223.23	203.04	182.42	127.19	130.98
Kddcup99	4303.12	3966.81	3626.89	2415.63	2520.96
PAMAP	6175.05	4976.09	3327.93	2220.87	2390.87
PowerC	1551.67	915.55	612.52	315.76	318.67

through evolutionary search, it remains relatively inefficient on dense or high-volume datasets, taking more than 4976 seconds on PAMAP and 3966 seconds on Kddcup99. GA-Apriori exhibits considerable improvement by integrating genetic algorithms, achieving runtime reductions of more than 50% on several datasets compared to GAFIM. However, our proposed approach, OGA-Apriori, delivers the best runtime results across all datasets. On PowerC, for instance, OGA-Apriori completes in 315.76 seconds, outperforming GA-Apriori (612.52 s) and even PSO-Apriori (318.67 s), while on Accidents (340,183 transactions, 468 items), it runs in 760.31 seconds, more than 600 seconds faster than its GA-based predecessor. PSO-Apriori also demonstrates competitive performance

and is often close to OGA-Apriori in speed on smaller datasets, such as RecordLink and Chess, but tends to lag slightly behind as dataset complexity increases.

Table 3. Percentage (%) of frequent itemsets discovered by GAFIM, GA-Apriori, and OGA-Apriori on various datasets

Dataset	GAFIM	GA-Apriori	OGA-Apriori	PSO-Apriori
Chess	63	89	91	93
Mushrooms	66	85	86	88
c20d10k	71	90	95	98
c73d10k	50	78	85	90
Connect	61	91	95	97
Accidents	62	92	95	97
RecordLink	56	71	80	88
Kddcup99	70	81	89	92
PAMAP	58	70	80	85
PowerC	40	65	79	82

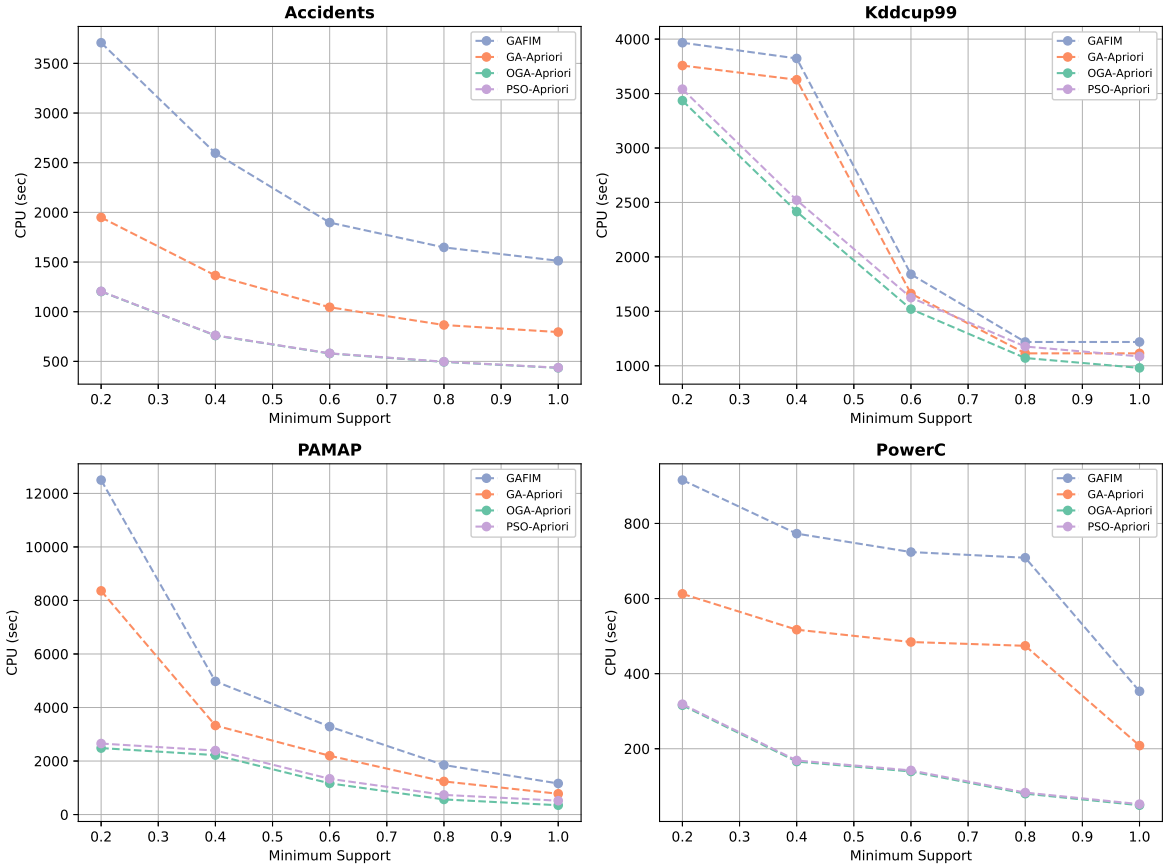

Figure 4. Runtime (Sec) of OGA-Apriori, GA-Apriori, PSO-Apriori and GAFIM using the instances: Accidents, PAMAP, Kddcup99, and PowerC.

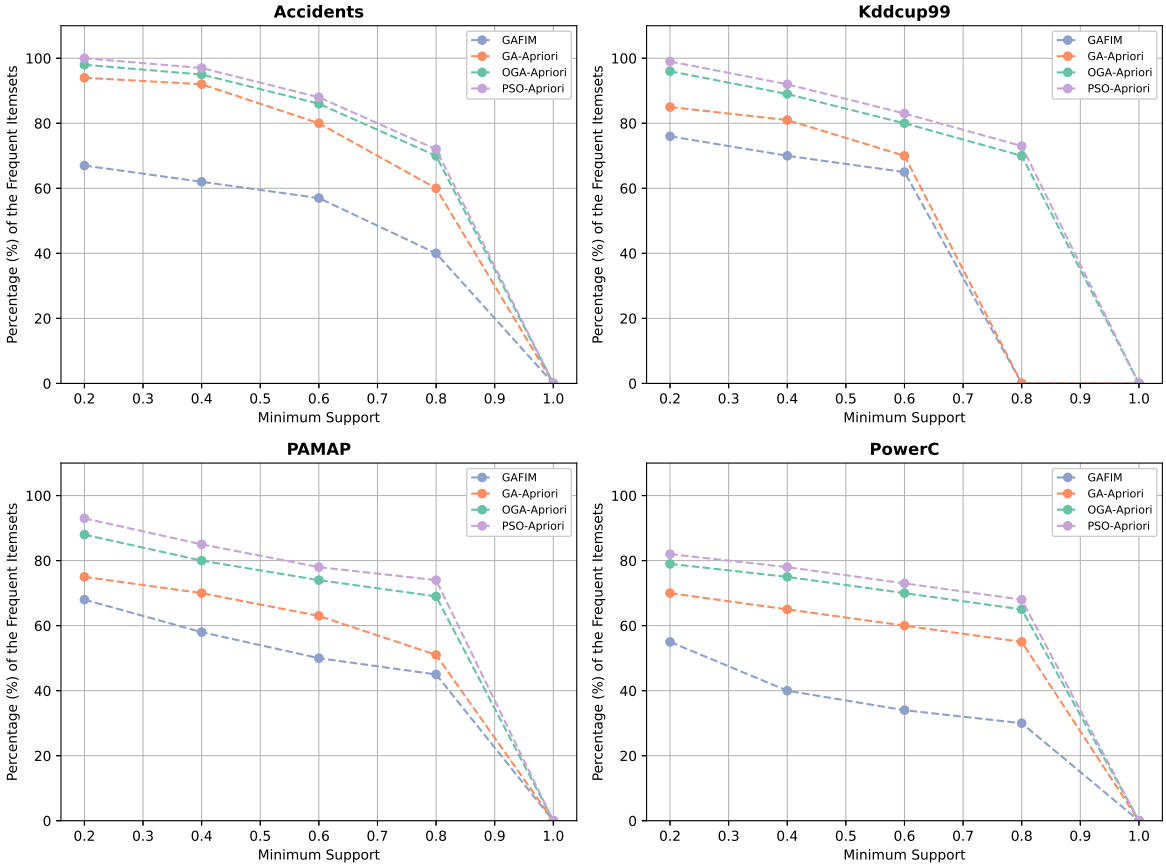

Figure 5. Percentage (%) of the number of the frequent itemsets of OGA-Apriori, GA-Apriori, PSO-Apriori and GAFIM using the instances: Accidents, PAMAP, Kddcup99, and PowerC.

The effectiveness of the algorithms, in terms of the percentage of frequent itemsets discovered (Table 3), offers additional insights. While GAFIM suffers from low coverage—only 40% on PowerC and 50% on c73d10k—both GA-Apriori and OGA-Apriori show marked improvements. OGA-Apriori demonstrates robust discovery performance, achieving 95% on Accidents and Connect (both containing high transaction lengths), and 89% on Kddcup99, which has many short but numerous transactions. PSO-Apriori, although slightly slower, achieves the highest overall discovery rates, reaching 98% on c20d10k and 97% on Accidents and Connect, thanks to its strong global exploration capabilities. However, this improvement in completeness comes with a trade-off in execution time, especially on large-scale datasets like PAMAP.

These findings are visually reinforced in Figure 4 and Figure 5. Figure **??** illustrates the runtime evolution across different minimum support thresholds for the datasets Accidents, PAMAP, Kddcup99, and PowerC. The figure shows that OGA-Apriori maintains the most stable and lowest runtime curve across all support values, particularly outperforming others on PAMAP and PowerC, where traditional approaches experience exponential runtime increases as the support decreases. In contrast, Figure 5 focuses on the discovery performance under the same conditions. It confirms that OGA-Apriori consistently delivers high percentages of discovered itemsets, remaining robust even at low support thresholds where search space is large and exhaustive methods like Apriori become infeasible. Although PSO-Apriori leads slightly in terms of completeness, its performance gain is marginal compared to its runtime cost when benchmarked against OGA-Apriori.

In summary, the results confirm that OGA-Apriori offers the best trade-off between runtime and completeness, particularly excelling on large, complex, and high-dimensional datasets. Its superior scalability, efficient use of

genetic operations, and effective parameter tuning make it a reliable and efficient choice for practical frequent pattern mining tasks across a wide range of data environments.

5. Statistical assessment

The statistical assessment was employed to evaluate the significant differences between the proposed approach and state-of-the-art models. To ensure the robustness and validity of our comparative analysis, we utilized the Friedman and Nemenyi tests, which are among the most commonly employed non-parametric statistical tests for comparing multiple algorithms across multiple datasets.

The Friedman test is particularly well-suited for this scenario as it assesses the null hypothesis that there is no significant difference in the performance of the compared approaches. It does so by ranking the performance of each model across different datasets and analyzing whether the observed differences in rankings could have occurred by chance. If the Friedman test indicates significant differences, this suggests that at least one approach consistently outperforms the others across the datasets. Originally introduced by Friedman [33, 34, 35]. In this test, each algorithm is ranked per dataset, with the top-performing algorithm receiving a rank of 1, the second-best a rank of 2, and so forth. Ties are resolved by assigning average ranks. The rank for the j-th algorithm on the i-th dataset among k algorithms is denoted as r_i^j. The test computes the mean ranks of the algorithms, defined as $R_j = \frac{1}{N}\sum_i r_i^j$.

Under the null hypothesis, which asserts that all algorithms perform equivalently (hence, their ranks R_j should be similar), the Friedman statistic is calculated using the formula:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right].$$

For large values of N and k, the distribution of this statistic closely approximates a χ^2 distribution with $k-1$ degrees of freedom.

Following the Friedman test, a Nemenyi post-hoc test [36] will be used to pinpoint which specific pairs of approaches differ significantly from each other. The Nemenyi test compares all possible pairs of approaches by calculating the critical difference (CD) between their mean ranks. If the difference in ranks between any two approaches exceeds this critical value, it is considered statistically significant.

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}.$$

In this context, q_α represents the critical value obtained from the Studentized range distribution, modified by a factor of $\sqrt{2}$. The hypotheses being considered are as detailed below:

- H_0: No significant differences exist over the evaluated models.
- H_1: Significant differences are present among the evaluated models.

5.1. Statistical assessment application based on runtime

In term of runtime results, we test side by side $k = 5$ methods over $N = 10$ datasets. The time is second was used to rank all models in each data-set. Table 4 presents the results of the rankings of the proposed approach against SOA methods.

Table 4. Ranking results based on runtime.

	Apriori		**GAFIM**		**GA-Apriori**		**PSO-Apriori**		**OGA-Apriori**	
	Score	Rank	Score	Rank	Score	Rank	Score	Rank	Score	Rank
Chess	15.96	5	10.90	4	9.03	3	6.50	2	6.11	1
Mushrooms	61.31	5	23.74	4	10.01	2	10.92	3	9.88	1
c20d10k	200.95	5	100.20	4	35.81	2	36.45	3	32.11	1
c73d10k	3089.07	5	2875.95	4	2676.13	3	2386.80	2	2274.02	1
Connect	1549.87	5	1402.63	4	909.83	3	790.65	2	788.69	1
Accidents	3989.79	5	2596.07	4	1364.73	3	762.46	2	760.31	1
RecordLink	223.23	5	203.04	4	182.42	3	130.98	2	127.19	1
Kddcup99	4303.12	5	3966.81	4	3626.89	3	2520.96	2	2415.63	1
PAMAP	6175.05	5	4976.09	4	3327.93	3	2390.87	2	2220.87	1
PowerC	1551.67	5	915.55	4	612.52	3	318.67	2	315.76	1
$\sum_i r_i^j$	50		40		28		22		10	
R_j	5		4		2.8		2.2		1	
R_j^2	25		16		7.84		4.84		1	

Based on the ranking results of the proposed method against SOA approaches across all datasets, as shown in Table 4, we derive the outcomes of the Friedman and Nemenyi tests. These results include the $Chi-square$ (χ^2), p-value, critical difference (CD), and confidence interval (CI), all of which are presented in Table 5.

Table 5. Friedman and Nemenyi test results.

Approach	χ^2	P_value	CD	CI
OGA-Apriori	28.81	7.95×10^{-8}	1.92	[6.110, 2415.630]

Since the calculated $p-value$ is less than 0.05, we reject the null hypothesis H_0. The Friedman test, accompanied by its ameliorated statistic, indicates significant differences over the three methods applied to 10 datasets.

Then, the Nemenyi test is employed to check the pair-wise statistical difference across all methods. After using the Nemenyi test we fond that the critical value $q_\alpha = 2.728$ and the corresponding $CD = 1.92$. Since the difference between the best and the worst performing algorithm is already greater than that, we can conclude that the $post-hoc$ test is powerful enough to detect any significant differences between the methods. The $post-hoc$ results are effectively conveyed through a clear graphical representation.

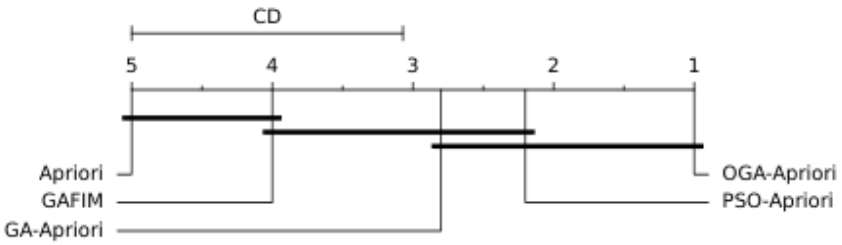

Figure 6. The critical difference of all methods against each other based on runtime.

As shown in Figure 6,The runtime evaluation across ten datasets clearly shows that OGA-Apriori significantly outperforms all baseline methods, achieving the lowest execution time and the best average rank. The Friedman test

confirms this with a highly significant p-value (7.95×10^{-8}), rejecting the null hypothesis of equal performance. The Nemenyi post-hoc test further validates that OGA-Apriori's improvements are statistically significant, as the rank differences exceed the critical difference (CD = 1.92). These results highlight OGA-Apriori as a highly efficient and scalable approach.

6. Conclusion

This paper proposed OGA-Apriori, a new improvement to the GA-Apriori metaheuristic for mining frequent itemsets. OGA-Apriori speeds up convergence and lowers computational overhead by focusing the evolutionary search on high-potential areas of the solution space. It does this by combining a support-guided initialization of candidate 1-itemsets with an improved Apriori-aware crossover operator.

Comprehensive tests on a wide range of benchmark instances show that OGA-Apriori consistently has the fastest execution times of all the methods tested. It also consistently extracts frequent patterns with high completeness. It's important to note that OGA-Apriori finds as many or more solutions as other heuristic methods, including swarm-based optimizers, but it does so with a lot less runtime. The Friedman and Nemenyi tests show that these performance gains are statistically significant when they are looked at closely.

Overall, the suggested OGA-Apriori framework strikes a good balance between efficiency and effectiveness, making it a great choice for use in real-world data-mining applications.

In future work, we will first focus on porting OGA-Apriori to distributed architectures like Apache Spark. This will let us scale horizontally by running support counting and genetic operations in parallel on partitioned data. We also want to look into adaptive parameter control, which automatically adjusts crossover and mutation rates at runtime based on how diverse the population is and how quickly it converges. This will make manual tuning even less necessary. We will show that OGA-Apriori can be used in the real world by combining it with clustering methods to group transactions into similar groups before mining. This should make both the runtime and the relevance of the patterns even better. Lastly, we want to make the algorithm work with streaming data so that frequent itemsets can be updated as new transactions come in without having to reprocess the whole dataset.

REFERENCES

1. Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.
2. Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12, 2000.
3. Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 255–264, 1997.
4. Emma Hart and Jon Timmis. Application areas of ais: The past, the present and the future. *Applied soft computing*, 8(1):191–201, 2008.
5. Youcef Djenouri, Nadia Nouali-Taboudjemat, and Ahcène Bendjoudi. Association rules mining using evolutionary algorithms. In *The 9th International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2014)*. LNCS, 2014.
6. Diana Martín, Jesús Alcalá-Fdez, Alejandro Rosete, and Francisco Herrera. Nicgar: a niching genetic algorithm to mine a diverse set of interesting quantitative association rules. *Information Sciences*, 355:208–228, 2016.
7. Youcef Djenouri, Habiba Drias, and Zineb Habbas. Bees swarm optimisation using multiple strategies for association rule mining. *International Journal of Bio-Inspired Computation*, 6(4):239–249, 2014.
8. Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, Jimmy Ming-Thai Wu, Tzung-Pei Hong, Leon Shyue-Liang Wang, and Justin Zhan. Mining high-utility itemsets based on particle swarm optimization. *Engineering Applications of Artificial Intelligence*, 55:320–330, 2016.
9. Youcef Djenouri and Marco Comuzzi. Ga-apriori: Combining apriori heuristic and genetic algorithms for solving the frequent itemsets mining problem. In *Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2017 Workshops, MLSDA, BDM, DM-BPM Jeju, South Korea, May 23, 2017, Revised Selected Papers 21*, pages 138–148. Springer, 2017.
10. Youcef Djenouri and Marco Comuzzi. Combining apriori heuristic and bio-inspired algorithms for solving the frequent itemsets mining problem. *Information Sciences*, 420:1–15, 2017.
11. Youcef Djenouri, Djamel Djenouri, Asma Belhadi, Philippe Fournier-Viger, and Jerry Chun-Wei Lin. A new framework for metaheuristic-based frequent itemset mining. *Applied Intelligence*, 48:4775–4791, 2018.
12. Jong Soo Park, Ming-Syan Chen, and Philip S Yu. An effective hash-based algorithm for mining association rules. *Acm sigmod record*, 24(2):175–186, 1995.
13. Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390, 2000.

14. Mohammed J Zaki and Ching-Jui Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM international conference on data mining*, pages 457–473. SIAM, 2002.
15. Jianyong Wang, Jiawei Han, and Xifeng Zhu. Mining frequent itemsets by prepost. In *Advances in Knowledge Discovery and Data Mining (PAKDD 2009)*, volume 5476 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2009.
16. CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.
17. Yassir Rochd and Imad Hafidi. Parallel implementation of PrePost algorithm based on Hadoop for big data. In *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)*, pages 24–31, Marrakech, Morocco, 2018. IEEE.
18. Yassir Rochd and Imad Hafidi. An enhanced apriori algorithm using hybrid data layout based on Hadoop for big data processing. *International Journal of Network Security*, 18(6):161–170, 2018.
19. Yassir Rochd and Imad Hafidi. Performance improvement of PrePost algorithm based on Hadoop for big data. *International Journal of Intelligent Engineering and Systems*, 11(5):226–235, 2018.
20. Yassir Rochd, Imad Hafidi, and Bajil Ouartassi. A review of scalable algorithms for frequent itemset mining for big data using Hadoop and Spark. In *Lecture Notes in Real-Time Intelligent Systems (Proceedings of RTIS 2017)*, volume 756 of *Advances in Intelligent Systems and Computing*, pages 90–99. Springer, Cham, 2018.
21. Yassir Rochd, Imad Hafidi, and Bajil Ouartassi. Parallel implementation of PrePost algorithm based on Spark for big data. In *Big Data and Smart Digital Environment (ICBDSDE 2018)*, volume 53 of *Studies in Big Data*, pages 322–332. Springer, Cham, 2019.
22. Yassir Rochd and Imad Hafidi. An efficient distributed frequent itemset mining algorithm based on Spark for big data. *International Journal of Intelligent Engineering and Systems*, 12(4):367–377, 2019.
23. Jacinto Mata, José-Luis Alvarez, and José-Cristobal Riquelme. An evolutionary algorithm to discover numeric association rules. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 590–594, 2002.
24. Bilal Alataş and Erhan Akin. An efficient genetic algorithm for automated mining of both positive and negative quantitative association rules. *Soft Computing*, 10:230–237, 2006.
25. Xiaowei Yan, Chengqi Zhang, and Shichao Zhang. Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert Systems with Applications*, 36(2):3066–3076, 2009.
26. Cristóbal Romero, Amelia Zafra, Jose María Luna, and Sebastián Ventura. Association rule mining using genetic programming to provide feedback to instructors from multiple-choice quiz data. *Expert Systems*, 30(2):162–172, 2013.
27. Simon Fong, Raymond Wong, and Athanasios V Vasilakos. Accelerated pso swarm search feature selection for data stream mining big data. *IEEE transactions on services computing*, 9(1):33–45, 2015.
28. RJ Kuo, SY Lin, and CW Shih. Mining association rules through integration of clustering analysis and ant colony system for health insurance database in taiwan. *Expert Systems with Applications*, 33(3):794–808, 2007.
29. Meng-Hua Yang, Lei Li, Ying-Shen Hung, Cheng-Shen Hung, Jean-Pierre Allain, Kuo-Sin Lin, and Su-Jen Lin Tsai. The efficacy of individual-donation and minipool testing to detect low-level hepatitis b virus dna in taiwan. *Transfusion*, 50(1):65–74, 2010.
30. Meryem Barik, Imad Hafidi, and Yassir Rochd. Review of heuristic algorithms for frequent itemsets mining problem. *Computing and Informatics*, 42(6):1360–1377, 2023.
31. Meryem Barik, Imad Hafidi, and Yassir Rochd. Heuristic algorithm for extracting frequent patterns in transactional databases. In *Advances in Machine Intelligence and Computer Science Applications (ICMICSA 2022)*, volume 656 of *Lecture Notes in Networks and Systems*, pages 361–371. Springer, Cham, 2023.
32. Mansour Sheikhan and Maryam Sharifi Rad. Gravitational search algorithm–optimized neural misuse detector with selected features by fuzzy grids–based association rules mining. *Neural Computing and Applications*, 23:2451–2463, 2013.
33. Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.
34. Milton Friedman. A correction. *Journal of the American Statistical Association*, 34(205):109–109, 1939.
35. Milton Friedman. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*, 11(1):86 – 92, 1940.
36. Peter Bjorn Nemenyi. *Distribution-free multiple comparisons*. Princeton University, 1963.