Computing and Informatics, Vol. 42, 2023, 1360–1377, doi: 10.31577/cai_2023_6_1360

REVIEW OF HEURISTIC ALGORITHMS FOR FREQUENT ITEMSETS MINING PROBLEM

Meryem Barik, Imad Hafidi, Yassir Rochd

Laboratory of Process Engineering, Computer Science and Mathematics (LIPIM)
University Sultan Moulay Slimane
Khouribga, Morocco
e-mail: meryem.barik@usms.ac.ma, {i.hafidi, y.rochd}@usms.ma

Abstract. Frequent Itemsets Mining (FIM), which consists of extracting frequent patterns from a transactional database, is considered one of the most successful techniques in data mining. Generally, the FIM problem can be solved by either the exact or metaheuristic-based methods. Exact methods, such as the Apriori algorithm, are highly effective for dealing with small to medium datasets. However, these methods need more temporal complexity when dealing with large datasets. Metaheuristic-based methods are becoming more rapid, but the majority still need to be more precise. Several studies were carried out to address these issues and improve metaheuristics-based approaches by combining the Apriori algorithm with several metaheuristics algorithms such as Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The result of this combination gave birth to two approaches: GA-Apriori and PSO-Apriori. Consequently, after performing several studies on different database instances, the results revealed that the two approaches outperformed the Apriori algorithm in terms of runtime. PSO-Apriori also beats GA-Apriori in terms of both runtime and solution efficiency.

Keywords: Frequent itemsets mining, genetic algorithm, particle swarm optimization, metaheuristic

1 INTRODUCTION

In the last few years, the data generated from several sources, called Big Data, has gradually exploded. Several techniques are used in Big Data, such as data mining, which is the science of extracting important information from a large data set. The application of traditional data mining techniques to large datasets is complicated. Among these techniques, we find the extraction of usable patterns from a large collection of data. This paper mainly focuses on the Frequent Itemset Mining (FIM) technique for extracting frequent items from a transactional database. This technique helps solve real-world problems by applying several algorithms; however, applying FIM algorithms on a large database can be very time-consuming. Two categories of approaches solve the FIM problem: exact and metaheuristic-based approaches.

In exact approaches of FIM, we find algorithms such as Apriori [1], FP-Growth [2], DIC [3], and AIS [4]. They aim to extract all frequent items, but it takes a long time to execute due to the multiple scans of an entire transactional database. The performance of these algorithms decreases in terms of the increase in database size, the number of items and transactions, and the high number of generated itemset, which becomes unacceptable for big data instances. To overcome the performance problem with exact methods, many FIM approaches that use metaheuristic-based techniques, such as genetic algorithms [5, 6] or swarm intelligence [7, 8], have been proposed.

Metaheuristic-based approaches extract a subset of all frequent itemsets in a short execution time but cannot find all possible frequent itemsets in a database. In other words, the efficiency of the solution obtained using metaheuristic-based approaches is lower than the optimum quality attained by exact approaches, which discover all possible frequent itemsets. The efficiency of the solutions in FIM metaheuristic-based approaches is determined by how the randomized search of the itemsets space is carried out. We contend that existing FIM metaheuristic-based methods in the literature need to take into account the inherent properties of the FIM solution space to enhance a search. The most important of these properties is that frequent itemsets are recursive, which means that if an itemset of size k is frequent, then all of its sub-itemsets of size $s = \{1, \ldots, k-1\}$ are also frequent. This function is essential to the Apriori exact algorithm, but it is seldom used by FIM metaheuristic-based approaches.

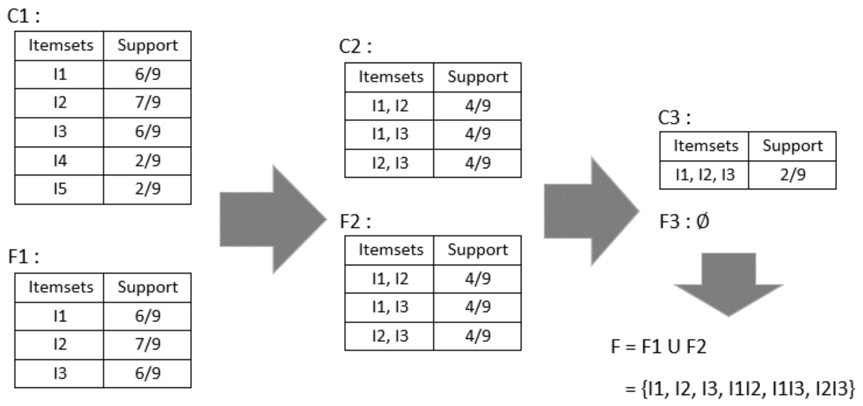
In literature, we find FIM metaheuristic-based approaches that take advantage of the recursive property of frequently occurring itemsets in a database. These approaches include GA-Apriori [9, 10, 11] and PSO-Apriori [10, 11], which use a genetic algorithm, and particle swarm optimization, respectively. The two approaches, GA-Apriori and PSO-Apriori, are an amelioration of existing algorithms GA-FIM [5] and PSO-FIM [8] by defining a new search space intensification and diversification operators, such as crossover and mutation for GA-FIM and particle positioning and velocity for PSO-FIM, that take the recursive property of frequent itemsets into account.

This paper is organized as follows: Section 2 introduces the Apriori algorithm, Section 3 reviews related work on the FIM problem, Section 4 represents the latest metaheuristic-based approaches, in particular, GA-Apriori and PSO-Apriori approaches, Section 5 represents a comparative study between the two approaches, and finally, Section 6 introduces the conclusion.

2 APRIORI ALGORITHM

The principle of the Apriori algorithm [1] is to generate all itemsets candidates of size k from frequent itemsets of size $k-1$ iteratively and recursively. This procedure is repeated until no itemsets candidates are generated during an iteration.

Let us consider a database contains 9 transactions: ((I1, I2, I5), (I2, I4), (I2, I3), (I1, I2, I4), (I1, I3), (I2, I3), (I1, I3), (I1, I2, I3, I5), (I1, I2, I3)). We apply the Apriori heuristic to this database. Figure 1 presents the steps of applying the Apriori heuristic with a MinSup equal to 30 %. First, the database is scanned to calculate the support of each candidate itemsets containing only one item. Then, the frequent 1-itemsets are extracted (C1). In this case, the frequent itemsets obtained are (I1, I2, I3) (F1) because their supports are greater than 0.3. The candidate itemsets of size 2 (C2) are generated in the second iteration by joining the frequent itemsets of size 1; then, the support of each candidate itemsets of size 2 is calculated (F2). The frequent itemsets obtained are (I1I2, I2I3, I2I3) (F2). By joining these three frequent itemsets, we obtain the candidate itemsets of size 3 (I1I2I3) (C3). The process is stopped since its support is less than 0.3. As an output, we obtain the set of all frequent itemsets F by the union of the frequent itemsets of size 1 and size 2. Therefore, F = (I1, I2, I3, I1I2, I1I3, I2I3). The most important disadvantage of the Apriori heuristic is scanning the entire database at each iteration to determine the support of each candidate itemsets.



C1 :

Itemsets	Support
I1	6/9
I2	7/9
I3	6/9
I4	2/9
I5	2/9

F1 :

Itemsets	Support
I1	6/9
I2	7/9
I3	6/9

C2 :

Itemsets	Support
I1, I2	4/9
I1, I3	4/9
I2, I3	4/9

F2 :

Itemsets	Support
I1, I2	4/9
I1, I3	4/9
I2, I3	4/9

C3 :

Itemsets	Support
I1, I2, I3	2/9

F3 : Ø

F = F1 U F2

= {I1, I2, I3, I1I2, I1I3, I2I3}

Figure 1. Apriori algorithm

3 RELATED WORKS

In this section, we will present the different approaches for discovering frequent itemsets existing in the literature. In Table 1, the approaches are classified into exact and bio-inspired. In the first part, we briefly introduce the Apriori algorithm, which exploits the property of frequent itemsets. Then, we review the related work using exact and metaheuristic-based approaches. Finally, we discuss the latest metaheuristic-based approaches in the literature, GA-Apriori and PSO-Apriori.

Type of approaches	Approaches	Limitation
Exact approaches	Apriori [1], AIS [4], Eclat [12], FPGrowth [2]	Multiple scans of the database. Poor runtime performance for large databases.
Meta-heuristic-based approaches	GENAR [13], GAR [14], ARMGA [15], AGA [16], PQMA [17], G3PARM [18], NICGAR [6], GAFIM [5], ACO [19], HUIM-ACS [20], PSOFIM [8], HUIM-BPSO [8], ARMBGSA [21], BSO [7], PeSOA [22], BATFIM [23]	Inefficient representation of individuals. Poor solution quality, especially with large databases. Poor runtime performance when generating new solutions.

Table 1. The different categories of FIM approaches

3.1 Exact Approaches

In addition to the Apriori algorithm, several algorithms belong to exact approaches. However, the Apriori algorithm has been introduced as an improvement of its predecessor AIS [1] – named after its inventors – which does not employ any optimization.

AIS [24] algorithm generates itemsets candidates by analyzing the database and extending the used frequent itemsets found in previous analysis to itemsets found in transactions. However, AIS performs many database analyses and consumes a lot of memory. Several algorithms in the literature are inspired by the Apriori algorithm.

AprioriTid and AprioriHybrid are two minor extensions proposed in the same Apriori paper [25]. The first reduces the number of scans of the database to just one. Indeed, AprioriTid builds, after the first pass, transaction tables reduced to candidate patterns, generated in the same way as Apriori. These structures replace the base. AprioriTid gives inferior results to Apriori in the first iterations since the new structures produce surcharges in memory (and in time because the data is replaced on disk). However, as the algorithm progresses, the results outperform Apriori's for the opposite reasons. This prompted the authors to suggest AprioriHybrid that combines the two, in which Apriori is invoked first, then once the structures AprioriTid can lodge in AprioriHybrid memory switches to AprioriTid, to benefit from the advantages of the two algorithms.

DHP (Dynamic Hashing and Pruning) [3] is introduced to reduce the number of candidates. DHP gradually reduces the size of the database and truncates transactions based on the necessary condition that any item i can be deleted if it does not appear at least k times in the candidate k-patterns of a transaction.

DIC (Dynamic Itemset Counting) [2] is motivated by the reduction in the number of passes on the database. The philosophy of this algorithm interweaves candidate generation. It supports computation by anticipating the formation of longer candidate patterns of patterns found to be frequent without even waiting to see all the transactions.

Han et al. have proposed the FPGrowth algorithm [2]. The latter uses a compact data structure called FPTree (Frequent Pattern tree) to represent the database, a sort of prefix tree or sort [26] augmented by support information. Thus, each tree node contains an item and an integer representing the support of the pattern formed from the items found on the path from the root to this node.

Exact approaches could be more efficient with large database instances in terms of time and space complexity. Due to various factors, such as the exponential growth of self-generated data from several sources, extensive databases have become modern signs [14]. This problem needs more efficient algorithms to extract frequent itemsets.

In the next section, we will present the different approaches that have been explored, such as bio-inspired approaches, which target strategies such as reducing the number of database scans.

3.2 Meta-Heuristic Based Approaches

We can categorize bio-inspired approaches as swarm intelligence-based approaches and evolutionary-based approaches.

3.2.1 Evolutionary-Based Approaches

In evolutionary-based approaches, we found approaches based on genetic algorithms. The first approach based on a genetic algorithm for association rule mining and frequent itemset mining is GAR [16]. The limitation of GAR is that it uses an inefficient representation of individuals. Individuals are represented according to their size and the number of items they contain. The size of individuals may differ across the populations, and it degrades the performance of both the crossover and mutation operators. Then, genetic algorithms that rely on an improved representation of individuals have been proposed. In particular, a more efficient way to represent individuals is to use a vector of n elements, whereby the i^{th} element is set to 1 if the i^{th} item belongs to the itemset and 0 otherwise. This is the representation we use in our framework and which are also used by the algorithms discussed in the remainder of this section. Then we have the two algorithms AGA [15] and ARMGA [15]. The significant difference between ARMGA and AGA is the mutation and crossover operators. ARMGA adopts a two-point crossover, whereas AGA considers a simple crossover operator. Another algorithm is G3PARM [18], which uses G3P (Grammar Guided Genetic Programming) to avoid finding invalid individuals. G3PARM can be used with various types of data using context-free grammar. In [27], a new Niching Genetic Algorithm is proposed to obtain diversified patterns from a solution space. Patterns are mined in each niche. A selection procedure is then applied to find the best patterns representing the whole niche using similarity measures between the extracted patterns. Martín et al. [6] integrated a windowing-based learning scheme in a genetic algorithm for discovering frequent itemsets in large-scale datasets. The algorithm first partitions a data instance into disjoint subsets. Then, these subsets are used to calculate the fitness of the current population using a round-robin approach. Recently, an efficient genetic algorithm named GAFIM was introduced [5]. The main innovation in GAFIM is a delete and decomposition strategy, which is developed to divide infrequent itemsets into pairs of frequent itemsets.

3.2.2 Swarm Intelligence Based Approaches

According to recent research, swarm intelligence methods can be used successfully in data mining problems such as feature collection, clustering, and periodic itemsets mining [28]. The first swarm intelligence-based approach using ACO (Ant Colony Optimization) [29] is proposed in [30]. It combines clustering and ACO. [29] develops an extension of this method for continuous domains. HUIM-ACS [20] is a more recent ACO-based technique that adapts the TWU heuristic [31] to lead the ants in exploring the solution space. The biggest disadvantage of ACO-based approaches to FIM is runtime efficiency.

The authors of [32] suggested a particle swarm optimization method for the FIM problem. The neighborhood space is discovered by relocating each particle's front and back points. This algorithm outperforms AGA, but the search based on front and back points yields more neighborhoods, favoring the intensification search over the diversity search. PSOFIM [8] has modified this algorithm to strike a balance between intensification and diversification.

The authors suggested ARMBGSA in [21], inspired by recent metaheuristics focused on Newtonian gravity and the rule of motion. Each itemset is represented as a mass. Using the rule of motion, all masses draw each other. After each iteration, the k-heaviest masses are chosen to exert power on the new masses in the next iteration. When opposed to other evolutionary algorithms, the ARMBGSA algorithm produces fewer itemsets. The FIM problem is solved using bees swarm optimization in [7]. The bees' search area is decided first, and then each region is searched by one bee to locate regular itemsets. The bees coordinate using a dance table to overlap at each iteration. The bees interact using a dance table at each iteration to agree on the strongest collection of itemsets.

According to [33] and [34], genetic algorithms and particle swarm optimization outperform other bio-inspired methods regarding runtime efficiency and solution consistency when solving the FIM problem. However, due to the design of the randomized search mechanism used, which does not consider the complexity of the

FIM problem, the accuracy of the solutions obtained by these algorithms remains inefficient.

BATFIM [23] has recently used the bat metaheuristic [35] to solve the FIM dilemma. A community of bats searches for relevant itemsets in the same area of the solutions space. Various methods are proposed to pick up the best-repeated itemsets from the total bats. The findings reveal that BATFIM outperforms the current evolutionary and swarm intelligence-based approaches to FIM.

4 THE LATEST META-HEURISTIC BASED APPROACHES

In this part, an item is considered a vector of n elements, and n is the number of items in the database. The vector's i^{th} element takes a 1 if the item i belongs to the itemset and 0 otherwise. For example, we consider a database T containing the following items: $\{a, b, c, d\}$. The itemset $\{a, b, c\}$ is presented by the vector $\{1, 1, 1, 0\}$, and the item $\{a, d\}$ by $\{1, 0, 0, 1\}$. In [10, 11], the authors proposed a new framework presented in Figure 2 to guide FIM approaches based on metaheuristics using the recursive property of frequent itemsets. This framework is an iterative algorithm. At first, it scans the entire database to find the frequent itemsets of size 1 by calculating their support and reserving the items with support greater than the *minSup*. Then, space exploration is done by joining the frequent itemsets to obtain larger ones iteratively. In other words, itemsets of iteration k are generated by joining frequent itemsets found in iteration $k - 1$. In iteration k, the generated itemsets are of size k. To facilitate the generation of frequent itemsets, the itemsets space is divided into k regions, and each region R_k contains itemsets of size k. If k is the size of the largest transaction in the database, then there are k regions to be explored at most, and the number of iterations cannot be greater than k.

4.1 GA-Apriori

This part will present the framework's applicability to the genetic algorithm introduced in [10, 11]. At first, an initial population of itemset of size 1 is randomly generated. Then, the crossover and mutation operators are applied. At the end of each iteration, a selection operator will be applied to select the most frequent itemsets only for the next iteration. The process (crossover, mutation, selection) is repeated for a maximum number of iterations. This approach combines the recursive property of frequent itemsets with a genetic algorithm to improve the exploration of space itemsets. The GA-Apriori differs from the previous algorithms in how genetic algorithm operators (initialization, crossover, mutation, and selection) are defined according to the proposed framework. In the following, we are going to describe these operators in detail:

Step 1 (Population Initialization): In this step, we choose the first frequent popSize itemsets by computing the itemset of size 1 and calculating their sup-

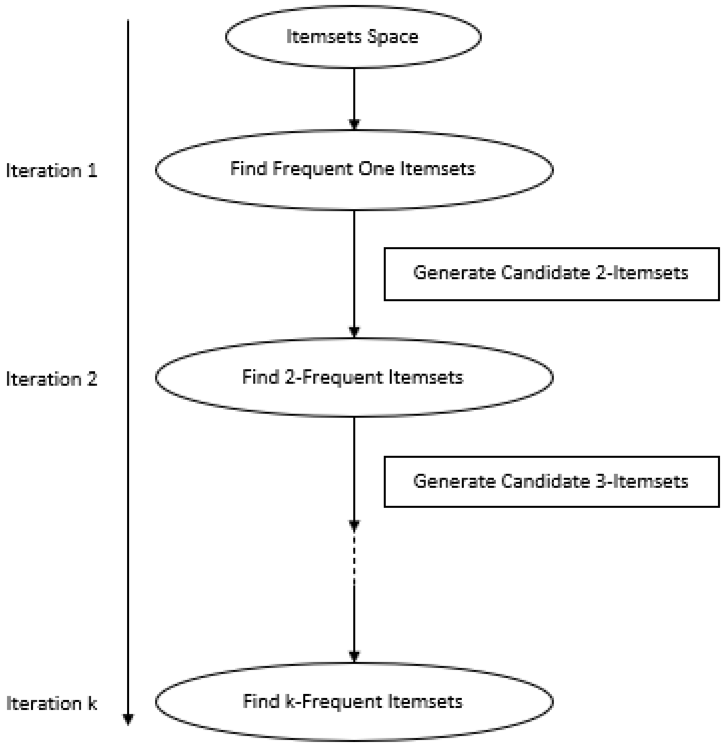


Figure 2. The main framework of the approaches

port, then sorting them according to their support. Then, the first frequent popSize itemsets are kept, and the others are removed.

For example, we have a database T containing 4 items T $= \{a, b, c, d\}$, with the following supports $\{\sup(a) = 0.4, \sup(b) = 0.5, \sup(c) = 0.1, \sup(d) = 0.3\}$. With the MinSup $= 0.3$, the frequent items found are $\{a, b, d\}$. If the popSize $=$ 2, the initial population is $\{1, 0, 0, 0\}$ and $\{0, 1, 0, 0\}$, representing items a and b, respectively.

Step 2 (Crossover): The objective of the crossover operator is to generate two itemsets child of size k from two frequent itemsets parents of size $k - 1$. The first step is to select two parents from the given population, and then we apply the crossover constraints inspired by the Apriori heuristic to generate the new children:

- Copying all items from the first parent to the first child and from the second parent to the second child;

- Choosing randomly the item e_1 equal to 1 in the second parent, which equals 0 in the first parent, then we modify his value to 1 in the first child. By the same, we randomly choose the item e_2 equal to 1 in the first parent and 0 in the second parent, then we set his value to 1 in the second child.

As an illustration, we have a database T containing 4 items T $= \{a, b, c, d\}$, we select two parents parent$_1 = \{1, 1, 0, 0\}$, that represents the itemset $\{a, b\}$ and parent$_2 = \{0, 0, 1, 1\}$, that represents the itemset $\{a, d\}$. We choose $e_1 = c$, the most frequent item in the second parent, and $e_2 = a$, the most frequent item in the first parent. So, the two children generated are child$_1 = \{1, 1, 1, 0\}$, and child$_2 = \{1, 0, 1, 1\}$, that represent the itemsets $\{a, b, c\}$, and $\{a, c, d\}$, respectively.

Step 3 (Mutation): This step aims to generate frequent itemsets of size k from infrequent itemsets of the same size for each itemset obtained from the new set of itemsets after applying the crossover operator. If we found an infrequent itemset indiv of size k, we randomly choose two items, e_1 equal to 1, and e_2 equal to 0 in indiv. The two values will be switched, meaning item e_1 will be set to 0, and e_2 to 1 in indiv. We repeat the operation until we find a frequent itemset of size k.

For instance, we have a database T containing 4 items T $= \{a, b, c, d\}$, and indiv $= \{1, 0, 1, 0\}$, that represents the itemset $\{a, c\}$. If we choose $e_1 = a$, and $e_2 = b$, then indiv becomes $\{0, 1, 1, 0\}$, representing the itemset $\{b, c\}$. If the new itemset is infrequent, the process will be repeated until a frequent itemset of size 2 is found. Otherwise, the process will be stopped.

Step 4 (Selection): This step aims to select the best frequent itemsets from itemsets generated by the crossover and the mutation operators. Then, the best popSize frequent itemsets are kept for the next iteration, and they will be the new population.

The algorithm takes as input the transactional database to calculate the support of the generated itemsets and the minimum support designated by the user to determine the frequent itemsets. Then, the algorithm looks for frequent items of size 1 with the FindFrequentOneItemset() function. This function lists all items and calculates their support. Then, it puts the frequent items in descending order, takes the first popSize frequent items, and uses them as an initial population. These frequent items are added to the set of frequent items F. Then, the crossover and mutation operators are applied. First, the crossover operator is applied to each pair of parents in the current population. The result is added to the new population. Second, the mutation operator defines the new population by transforming infrequent itemsets into frequent itemsets. Then, the new population is modified by applying the mutation operator, and the frequent itemsets will be added to the set of frequent itemsets F. Finally, the selection procedure is applied. In this step, the new population becomes the current population for the next iteration.

This process is repeated until the current population is empty.

4.2 PSO-Apriori

In the previous works of particle swarm optimization based on FIM problem [32, 36], each swarm particle represents an itemset randomly positioned in the itemset space. During the i^{th} iteration, the particle p_i explores the itemset space using the position x_i and the velocity v_i.

$$x_i(t) = x_i(t-1) + v_i(t) \tag{1}$$

with

$$v_i(t) = W * v_i(t-1) + C_1 * (x_{pi} - x_i) + C_2(x_{gi} - x_i). \tag{2}$$

In Equations (1) and (2), x_{pi} is the most frequent item position found by x_i, and x_{gi} is the best particle in the swarm, i.e., the most frequent itemset observed by all particles. W is the inertia weight of a particle, and it controls the trade-off between global and local exploration. The parameters C_1 and C_2 control the importance of the best individual and global items.

The authors of [10, 11] proposed PSO-Apriori that instantiates the framework in Figure 2 with the PSO algorithm. The application of the recursive property of frequent itemsets is used to update the particle's positions by updating the operators of the PSO algorithm. These newly updated operators are proposed instead of the old position and speed operators used in the literature to explore the solution space. In the following, we are going to describe the initialization and the new operators in detail:

Particle initialization: The initialization of the particle positions is done by choosing the frequent P items of size 1. The P most frequent items are selected. Then, each position I_i will be assigned to a particle p_i. At the beginning, the particle velocity is initialized to 0.

For example, we have $\{a, b, c, d\}$, the frequent items sorted by support. If P = 3, then the initial positions of the particles are: $\{1, 0, 0, 0\}$ that represent the first particle, the item a, $\{0, 1, 0, 0\}$ that represent the second particle, the item b, and $\{0, 0, 1, 0\}$ that represent the third particle, the item c.

Update positions: In [10], in an iteration k the authors define new operators joining ($\odot$) and combining ($\oplus$) to explore the solution space of the particles. The position $x_i(k)$ of the particle p_i is updated by applying the following equations during an iteration k:

$$x_i(k) = x_i(k-1) \oplus v_i(k) \tag{3}$$

with

$$v_i(k) = x_{pi}(k-1) \odot x_{gi}(k-1). \tag{4}$$

- **Joining operator $\odot$:** Equation (4) shows the velocity of each particle p_i by joining the best itemset (the most frequent) x_{pi} of the swarm p_i, and an itemset x_{gi} selected at random from the set of the most frequent elements observed until now and not belonging to x_{pi}. This step produces an itemset of size k by choosing randomly an item from x_{pi} and joining it to x_{gi}.

For instance, we consider a database T contains 4 items T $= \{a, b, c, d, e\}$ if the best itemset x_{gi} of the second iteration is $\{1, 1, 0, 0, 0\}$ and the best local itemset x_{pi} observed by the second particle p_2 is $\{0, 0, 0, 1, 1\}$. Hence, the velocity of the second particle in the third iteration is $\{1, 1, 0, 1, 0\}$, by choosing randomly the item d from x_{pi} and joining it to x_{gi}.

- **Combining operator $\oplus$:** Equation (3) shows the position of each particle p_i by combining the new velocity and the position of the current particle. By replacing the item with low support in the velocity vector with an item with high support in the current position of p_i, which does not appear in the velocity vector of this particle.

  As an illustration, we have the following items sorted by their support $\{a, b, c, d, e\}$, and the velocity of the particle p_2 is $\{1, 1, 0, 1, 0\}$ and the current position of p_2 is $\{1, 0, 1, 0, 0\}$. Then, by replacing item d with item c in the vector of velocity of p_2, we obtain the new position of p_2 $\{1, 1, 1, 0, 0\}$.

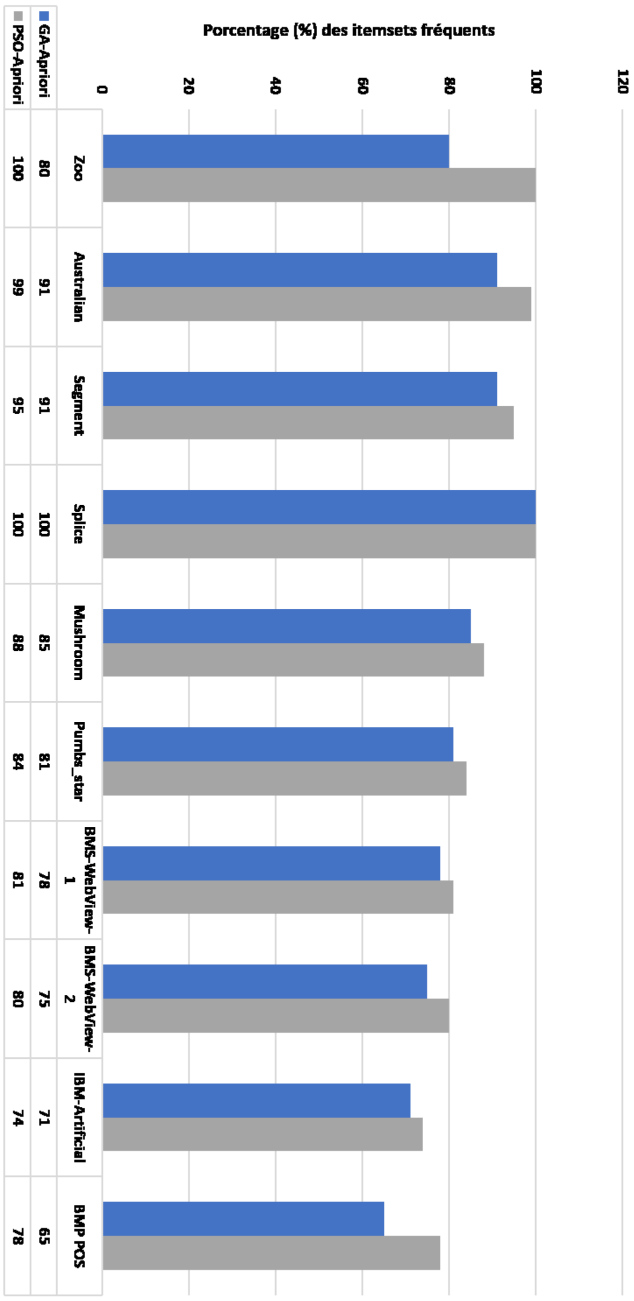
This process is repeated until the set of current itemsets is empty. The algorithm takes as input a transactional database to calculate the support of the generated itemsets and the minimum support designated by the user to determine the frequent itemsets. Then, the algorithm looks for frequent items of size 1 with the FindFrequentOneItemset() function. This function lists all items and calculates their support. Then, it puts the frequent items in descending order, takes the first P frequent items, assigns each particle to a position, and sets the initial velocity of each particle to 0. These frequent items are added to the set of frequent items F. The operators of joining and combining are applied to each particle to update the particle's velocity and position. The frequent itemsets found in the current iteration are collected, stored, sorted, and added to the set of frequent itemsets F.

This process is repeated until the current population is empty.

5 EXPERIMENTAL RESULTS

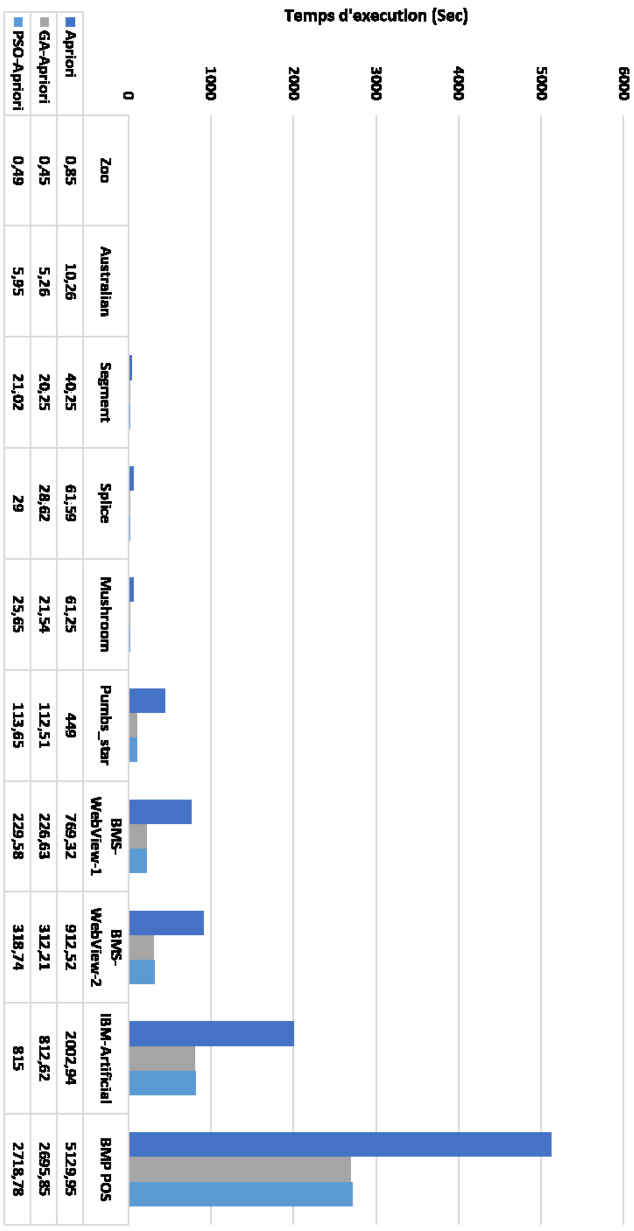
Several studies on medium, high, and large databases were done to evaluate the performance of GA-Apriori and PSO-Apriori. These instances of data are described in Table 2, they can be downloaded from `https://archive.ics.uci.edu/ml/datasets.html`, `http://fimi.ua.ac.be/data/`, and `https://sourceforge.net/projects/ibmquestdatagen/`. These experiments have been run on a laptop equipped with an Intel I5 processor and 8 GB memory, and all algorithms have been implemented in Java.

Figure 3 shows that PSO-Apriori outperforms GA-Apriori in terms of quality of solutions, i.e., percentage of frequent itemsets found. Knowing that the Apriori algorithm finds all frequent itemset (100 %) in all cases. The PSO-Apriori converges to 100 % of solutions in almost all cases.



	Zoo	Australian	Segment	Splice	Mushroom	Pumbs_star	BMS-WebView-1	BMS-WebView-2	IBM-Artificial	BMP POS
🟦 GA-Apriori	80	91	91	100	85	81	78	75	71	65
⬜ PSO-Apriori	100	99	95	100	88	84	81	80	74	78

Figure 3. Percentage (%) of frequent itemsets



	Zoo	Australian	Segment	Splice	Mushroom	Pumbs_star	BMS-WebView-1	BMS-WebView-2	IBM-Artificial	BMP POS
■ Apriori	0,85	10,26	40,25	61,59	61,25	449	769,32	912,52	2002,94	5129,95
■ GA-Apriori	0,45	5,26	20,25	28,62	21,54	112,51	226,63	312,21	812,62	2695,85
■ PSO-Apriori	0,49	5,95	21,02	29	25,65	113,65	229,58	318,74	815	2718,78

Figure 4. Runtime (Sec) of approaches

Instance Name	No. of Transactions	No. of Items	Avg. Size of Transaction
Zoo	102	17	17
Australian	690	60	60
Segment	2 310	19	19
Splice	3 190	6	6
Mushroom	8 124	119	23
Pumbs_star	40 385	7 116	50
BMSWebView1	59 602	497	2.5
BMSWebView2	77 512	3 340	5
IBMArtificial	100 000	999	10
BMP POS	515 597	1 657	2.5

Table 2. Data instances description

The runtime performance is compared in Figure 4. The results revealed that PSO-Apriori and GA-Apriori outperform the Apriori algorithm regarding computing time. When dealing with small instances (such as Zoo, Australian, and Segment), the execution time is almost the same with all small instances. Moreover, with the most significant instances, like BMP POS, the runtime of metaheuristic-based approaches is lower than that of the Apriori algorithm.

6 CONCLUSION

This paper presents exciting research directions to improve algorithms for extracting frequent items for Big Data. As seen in this paper, various algorithms have been proposed to discover frequent items. Metaheuristic-based algorithms are ones of those. Recently, two metaheuristic-based methods have been proposed to solve the FIM problem. The itemset space is explored by integrating the recursive property of frequent itemsets and the stochastic search mechanism of metaheuristics. Two new metaheuristic-based methods for FIM have been established using this method. In the first (GA-Apriori), the crossover operator generates itemsets of size k from frequent itemsets of size $k-1$ for each iteration k. In contrast, the mutation operator finds frequent itemsets from the itemsets generated by the crossover. In the second (PSO-Apriori), the recursive property of frequent itemsets directs the direction and velocity of the particles exploring the itemsets solution space. Several studies on medium, high, and extensive database instances were done to evaluate the efficiency of these approaches. The results revealed that PSO-Apriori outperforms GA-Apriori regarding both runtime and solution efficiency. As a perspective, we aim to facilitate the GA-Apriori approach by modifying the crossover operator or the mutation operator to have a good result in terms of runtime and number of frequent itemsets found. As far as we know some have been designed to work with distributed frameworks such as Hadoop, MapReduce, and Spark. In future work, we plan to apply distribution using Apache Spark to the approach GA-Apriori and PSO-Apriori.

REFERENCES

[1] Agrawal, R.—Imieliński, T.—Swami, A.: Mining Association Rules Between Sets of Items in Large Databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93), 1993, pp. 207–216, doi: 10.1145/170035.170072.

[2] Han, J.—Pei, J.—Yin, Y.: Mining Frequent Patterns Without Candidate Generation. ACM SIGMOD Record, Vol. 29, 2000, No. 2, pp. 1–12, doi: 10.1145/335191.335372.

[3] Brin, S.—Motwani, R.—Ullman, J. D.—Tsur, S.: Dynamic Itemset Counting and Implication Rules for Market Basket Data. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD '97), 1997, pp. 255–264, doi: 10.1145/253260.253325.

[4] Hart, E.—Timmis, J.: Application Areas of AIS: The Past, the Present and the Future. Applied Soft Computing, Vol. 8, 2008, No. 1, pp. 191–201, doi: 10.1016/j.asoc.2006.12.004.

[5] Djenouri, Y.—Nouali-Taboudjemat, N.—Bendjoudi, A.: Association Rules Mining Using Evolutionary Algorithms. The 9th International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2014), 2014.

[6] Martín, D.—Alcalá-Fdez, J.—Rosete, A.—Herrera, F.: NICGAR: A Niching Genetic Algorithm to Mine a Diverse Set of Interesting Quantitative Association Rules. Information Sciences, Vol. 355-356, 2016, pp. 208–228, doi: 10.1016/j.ins.2016.03.039.

[7] Djenouri, Y.—Drias, H.—Habbas, Z.: Bees Swarm Optimisation Using Multiple Strategies for Association Rule Mining. International Journal of Bio-Inspired Computation, Vol. 6, 2014, No. 4, pp. 239–249, doi: 10.1504/IJBIC.2014.064990.

[8] Lin, J. C. W.—Yang, L.—Fournier-Viger, P.—Wu, J. M. T.—Hong, T. P.—Wang, L. S. L.—Zhan, J.: Mining High-Utility Itemsets Based on Particle Swarm Optimization. Engineering Applications of Artificial Intelligence, Vol. 55, 2016, pp. 320–330, doi: 10.1016/j.engappai.2016.07.006.

[9] Djenouri, Y.—Comuzzi, M.: GA-Apriori: Combining Apriori Heuristic and Genetic Algorithms for Solving the Frequent Itemsets Mining Problem. In: Kang, U., Lim, E. P., Yu, J. X., Moon, Y. S. (Eds.): Trends and Applications in Knowledge Discovery and Data Mining (PAKDD 2017). Springer, Cham, Lecture Notes in Computer Science, Vol. 10526, 2017, pp. 138–148, doi: 10.1007/978-3-319-67274-8_13.

[10] Djenouri, Y.—Comuzzi, M.: Combining Apriori Heuristic and Bio-Inspired Algorithms for Solving the Frequent Itemsets Mining Problem. Information Sciences, Vol. 420, 2017, pp. 1–15, doi: 10.1016/j.ins.2017.08.043.

[11] Djenouri, Y.—Djenouri, D.—Belhadi, A.—Fournier-Viger, P.—Lin, J. C. W.: A New Framework for Metaheuristic-Based Frequent Itemset Mining. Applied Intelligence, Vol. 48, 2018, pp. 4775–4791, doi: 10.1007/s10489-018-1245-8.

[12] Zaki, M. J.—Parthasarathy, S.—Ogihara, M.—Li, W.: Parallel Algorithms for Discovery of Association Rules. Data Mining and Knowledge Discovery, Vol. 1,

1997, No. 4, pp. 343–373, doi: 10.1023/A:1009773317876.

[13] Mata, J.—Alvarez, J. L.—Riquelme, J. C.: Mining Numeric Association Rules with Genetic Algorithms. In: Kůrková, V., Neruda, R., Kárný, M., Steele, N. C. (Eds.): Artificial Neural Nets and Genetic Algorithms. Springer, Vienna, 2001, pp. 264–267, doi: 10.1007/978-3-7091-6230-9_65.

[14] Mata, J.—Alvarez, J. L.—Riquelme, J. C.: An Evolutionary Algorithm to Discover Numeric Association Rules. Proceedings of the 2002 ACM Symposium on Applied Computing (SAC '02), 2002, pp. 590–594, doi: 10.1145/508791.508905.

[15] Yan, X.—Zhang, C.—Zhang, S.: Genetic Algorithm-Based Strategy for Identifying Association Rules Without Specifying Actual Minimum Support. Expert Systems with Applications, Vol. 36, 2009, No. 2, pp. 3066–3076, doi: 10.1016/j.eswa.2008.01.028.

[16] Alataş, B.—Akin, E.: An Efficient Genetic Algorithm for Automated Mining of Both Positive and Negative Quantitative Association Rules. Soft Computing, Vol. 10, 2006, No. 3, pp. 230–237, doi: 10.1007/s00500-005-0476-x.

[17] Liu, D.: Improved Genetic Algorithm Based on Simulated Annealing and Quantum Computing Strategy for Mining Association Rules. Journal of Software, Vol. 5, 2010, No. 11, pp. 1243–1249.

[18] Romero, C.—Zafra, A.—Luna, J. M.—Ventura, S.: Association Rule Mining Using Genetic Programming to Provide Feedback to Instructors from Multiple-Choice Quiz Data. Expert Systems, Vol. 30, 2013, No. 2, pp. 162–172, doi: 10.1111/j.1468-0394.2012.00627.x.

[19] Kuo, R. J.—Shih, C. W.: Association Rule Mining Through the Ant Colony System for National Health Insurance Research Database in Taiwan. Computers & Mathematics with Applications, Vol. 54, 2007, No. 11-12, pp. 1303–1318, doi: 10.1016/j.camwa.2006.03.043.

[20] Wu, J. M. T.—Zhan, J.—Lin, J. C. W.: An ACO-Based Aproach to Mine High-Utility Itemsets. Knowledge-Based Systems, Vol. 116, 2017, pp. 102–113, doi: 10.1016/j.knosys.2016.10.027.

[21] Sheikhan, M.—Sharifi Rad, M.: Gravitational Search Algorithm-Optimized Neural Misuse Detector with Selected Features by Fuzzy Grids-Based Association Rules Mining. Neural Computing and Applications, Vol. 23, 2013, No. 7, pp. 2451–2463, doi: 10.1007/s00521-012-1204-y.

[22] Gheraibia, Y.—Moussaoui, A.—Djenouri, Y.—Kabir, S.—Yin, P. Y.: Penguins Search Optimisation Algorithm for Association Rules Mining. Journal of Computing and Information Technology, Vol. 24, 2016, No. 2, pp. 165–179, doi: 10.20532/cit.2016.1002745.

[23] Heraguemi, K. E.—Kamel, N.—Drias, H.: Multi-Swarm Bat Algorithm for Association Rule Mining Using Multiple Cooperative Strategies. Applied Intelligence, Vol. 45, 2016, pp. 1021–1033, doi: 10.1007/s10489-016-0806-y.

[24] Agrawal, R.—Srikant, R.: Fast Algorithms for Mining Association Rules. Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), Vol. 1215, 1994, pp. 487–499.

[25] Park, J. S.—Chen, M. S.—Yu, P. S.: An Effective Hash-Based Algorithm for

Mining Association Rules. ACM SIGMOD Record, Vol. 24, 1995, No. 2, pp. 175–186, doi: 10.1145/568271.223813.

[26] Chen, C. L. P.—Zhang, C. Y.: Data-Intensive Applications, Challenges, Techniques and Technologies: A Survey on Big Data. Information Sciences, Vol. 275, 2014, pp. 314–347, doi: 10.1016/j.ins.2014.01.015.

[27] Parpinelli, R. S.—Lopes, H. S.—Freitas, A. A.: Data Mining with an Ant Colony Optimization Algorithm. IEEE Transactions on Evolutionary Computation, Vol. 6, 2002, No. 4, pp. 321–332, doi: 10.1109/TEVC.2002.802452.

[28] Fong, S.—Wong, R.—Vasilakos, A. V.: Accelerated PSO Swarm Search Feature Selection for Data Stream Mining Big Data. IEEE Transactions on Services Computing, Vol. 9, 2016, No. 1, pp. 33–45, doi: 10.1109/TSC.2015.2439695.

[29] Kuo, R. J.—Lin, S. Y.—Shih, C. W.: Mining Association Rules Through Integration of Clustering Analysis and Ant Colony System for Health Insurance Database in Taiwan. Expert Systems with Applications, Vol. 33, 2007, No. 3, pp. 794–808, doi: 10.1016/j.eswa.2006.08.035.

[30] Olmo, J. L.—Luna, J. M.—Romero, J. R.—Ventura, S.: Mining Association Rules with Single and Multi-Objective Grammar Guided Ant Programming. Integrated Computer-Aided Engineering, Vol. 20, 2013, No. 3, pp. 217–234, doi: 10.3233/ICA-130430.

[31] Tseng, V. S.—Shie, B. E.—Wu, C. W.—Yu, P. S.: Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. IEEE Transactions on Knowledge and Data Engineering, Vol. 25, 2012, No. 8, pp. 1772–1786, doi: 10.1109/TKDE.2012.59.

[32] Kuo, R. J.—Chao, C. M.—Chiu, Y. T.: Application of Particle Swarm Optimization to Association Rule Mining. Applied Soft Computing, Vol. 11, 2011, No. 1, pp. 326–336, doi: 10.1016/j.asoc.2009.11.023.

[33] del Jesus, M. J.—Gamez, J. A.—Gonzalez, P.—Puerta, J. M.: On the Discovery of Association Rules by Means of Evolutionary Algorithms. WIREs Data Mining and Knowledge Discovery, Vol. 1, 2011, No. 5, pp. 397–415, doi: 10.1002/widm.18.

[34] Krishna, G. J.—Ravi, V.: Evolutionary Computing Applied to Customer Relationship Management: A Survey. Engineering Applications of Artificial Intelligence, Vol. 56, 2016, pp. 30–59, doi: 10.1016/j.engappai.2016.08.012.

[35] Yang, M. H.—Li, L.—Hung, Y. S.—Hung, C. S.—Allain, J. P.—Lin, K. S.—Tsai, S. J. L.: The Efficacy of Individual-Donation and Minipool Testing to Detect Low-Level Hepatitis B Virus DNA in Taiwan. Transfusion, Vol. 50, 2010, No. 1, pp. 65–74, doi: 10.1111/j.1537-2995.2009.02357.x.

[36] Sarath, K. N. V. D.—Ravi, V.: Association Rule Mining Using Binary Particle Swarm Optimization. Engineering Applications of Artificial Intelligence, Vol. 26, 2013, No. 8, pp. 1832–1840, doi: 10.1016/j.engappai.2013.06.003.

Meryem Barik is a Ph.D. student at the National School of Applied Sciences in Khouribga, Sultan Moulay Slimane University, Morocco. She received her Master's degree in big data and decision making in 2020 from the same school. Her main research interests include big data, data mining, frequent itemsets mining, and heuristics.

Imad Hafidi is currently serving as Professor at the National School of Applied Science (ENSA), Khouribga, Morocco. He is the Head of the Department of Mathematics and Computer Engineering and the Director of the Laboratory of Process Engineering, Computer Science and Mathematics (LIPIM) of ENSA, Khouribga. His main research interests include big data, data mining, frequent itemsets mining, and heuristics.

Yassir Rochd is currently Professor at the National School of Applied Science (ENSA), Khouribga, Morocco, affiliated to the Mathematics and Computer Science Department, member of the LIPIM Laboratory (Process Engineering, Computer Science and Mathematics). His research interests include data mining, big data, and artificial intelligence.