IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 February 2024, accepted 7 March 2024, date of publication 12 March 2024, date of current version 19 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3376477

RESEARCH ARTICLE

Probabilistic Support Prediction: Fast Frequent Itemset Mining in Dense Data

MUHAMMAD SADEEQULLAH[1], AZHAR RAUF[1], SAIF UR REHMAN[1], AND NOHA ALNAZZAWI[2]
[1]Department of Computer Science, University of Peshawar, Peshawar 25120, Pakistan
[2]Computer Science and Engineering Department, Yanbu Industrial College, Yanbu 46452, Saudi Arabia

Corresponding author: Muhammad Sadeequllah (sadeequllah@uop.edu.pk)

ABSTRACT Frequent itemset mining (FIM) is a highly resource-demanding data-mining task fundamental to numerous data-mining applications. Support calculation is a frequently performed computation-intensive operation of FIM algorithms, whereas storing transactional data is memory-intensive. The FIM is even more resource-hungry for dense data than for sparse data. The rapidly growing size of datasets further exacerbates this situation and necessitates the design of out-of-the-box highly efficient solutions. This paper proposes a novel approach to frequent itemset mining for dense datasets. This approach, after the initial stage, does not use transactional data, which makes it memory efficient. It also replaces processing-intensive support calculations with efficient support predictions, which are probabilistic and need no transactional data. To predict the support of an itemset, it only needs the support of its subsets. However, this technique works only for itemsets of size three or higher. We also propose an FIM algorithm ProbBF, which incorporates this technique. The ProbBF discards transactional data after it uses it to calculate frequent one and two-size itemsets. For the itemsets of size k, where $k \geq 3$, ProbBF uses the proposed probabilistic technique to predict their support. It is considered frequent if the predicted support is greater than a given threshold. Our experiments show that ProbBF is efficient in both time and space against state-of-the-art FIM algorithms that use transactional data. The experiments also show that ProbBF can successfully generate the majority of the frequent itemsets on real-world datasets. Since ProbBF is probabilistic, some loss in quality is inevitable.

INDEX TERMS Association rule, data mining, frequent itemsets mining, support-count approximation, approximate algorithms, transaction databases.

I. INTRODUCTION

Association rule mining [1] is a rule-based machine learning approach used to discover meaningful associations among variables representing market basket items, consecutive clicks by respective users in click streams, user actions in intrusion detection, stock analysis, bioinformatics, medical diagnosis, and business decisions. A major task in association rule mining is frequent itemset mining (FIM), first introduced in [2]. It is an unsupervised data-mining technique that aims to find items that appear together frequently in transactions of a transaction dataset. It is a highly computational and space-intensive task, as it finds all possible combinations of frequent items that are also frequent in the transaction dataset.

The associate editor coordinating the review of this manuscript and approving it for publication was Juan A. Lara.

The most resource-demanding part of the FIM algorithm is the support counting of itemsets from the transaction dataset. For example, in a vertical dataset format (e.g., ECLAT [3]), the transaction IDs (TIDs) of all transactions are recorded against each item in which that item appears. If there are two frequent items A and B, and we want to check if itemset {A, B} is also frequent, we find the cardinality of the intersection of the two sets of TIDs of A and B. The result of this intersection is also stored because to check if itemset {A, B, C} is also frequent, we need only one intersection, i.e., intersecting the TIDs of itemset {A, B} and item C. If we do not store this intermediate result, we would do two intersections, i.e., the outcome of the intersection of A and B will be intersected with C. Though this saves computations, it increases memory demand. These transaction IDs and the intersection operations on them represent

the major share of the space and time an FIM algorithm requires.

Horizontal formats (e.g., Apriori [2]) have similar computational and memory requirements. Pattern-growth-based algorithms [4], [5], [6] use a compressed data structure, the FP-tree, which saves memory. However, the recursive buildup of conditional FP-trees places it in the same class of time and space complexity. The recent state-of-the-art algorithms are no exception [7], [8], [9], [10], [11], [12], [13].

A. MOTIVATION

The performance bottleneck of FIM algorithms can be attributed to two main factors: 1) the requirement for extensive memory to store transaction data; and 2) the computation of itemset support, a frequently executed and highly processing-intensive operation. For large-scale datasets, the time required for support calculations can easily exceed reasonable limits [14]. This problem becomes even more severe in dense datasets than in sparse datasets [15]. This is because the total frequent itemsets, even on high support thresholds and average-size datasets, could easily reach billions [10]. The intensity of the problem is rising because decades of developments in information systems have resulted in the fast growth of transaction data, which is posing tremendous challenges to the exact FIM algorithms [16]. More efficient parallel algorithms were developed [17], [18], but their scalability is limited by the size of the shared memory. The distributed algorithms, based on the MapReduce framework [19], are more scalable, but they also suffer from frequent I/O operations and communication overheads. In recent years, renewed interest has been noted in the approximate FIM algorithms [14], [16], [20], [21], [22], [23]. These algorithms discover frequent itemsets with limited computations by minimizing reliance on transaction data, enhancing the efficiency and scalability of the approximate FIM algorithms. The flip side is that these algorithms could miss some of the frequent itemsets.

Although Approximate FIM algorithms have achieved high efficiency, but their reliance on transaction data, even in reduced size, poses scalability limitations due to associated memory and computational costs. Furthermore, the inclusion of additional parametric settings in these algorithms also burdens users, requiring an understanding beyond frequent itemset mining. To address these challenges, approximate FIM algorithms should minimize dependence on transaction data and avoid introducing unnecessary parameters. This approach ensures high scalability and ease of use while still being capable of generating the majority of frequent itemsets (FI).

B. CONTRIBUTION

The main focus of this paper is to propose an approximate FIM algorithm that is not only more efficient but also easy to use. The main contributions of this paper are outlined below.

1. This paper introduces a statistical technique, the Probabilistic Support Prediction Model (PSPM), which is distinct from other support approximation methods that rely on partial transaction data, as it utilizes zero transaction data.
2. It proposes the ProbBF FIM algorithm, which recursively applies the PSPM to successfully identify the majority of frequent itemsets.
3. It is easier to use as it does not require any additional parameters other than the support threshold, unlike other approximate FIM algorithms that heavily rely on additional parameters.
4. In contrast to other approximate FIM algorithms facing challenges with large dense datasets, ProbBF is tailored for efficient performance on such datasets.
5. Unlike some approximate algorithms, ProbBF is suitable for generating frequent itemsets (FIs), maximal frequent itemsets (MFIs), and closed frequent itemsets (CFIs).

The rest of the paper is organized as follows. In Section II, we review the most relevant work in the FIM literature. Section III introduces some preliminaries related to frequent itemset mining. In Section IV, the PSPM technique mentioned earlier is presented. Section V introduces the proposed algorithm, ProbBF. In Section VI, we analyze the performance of the ProbBF algorithm against state-of-the-art FIM algorithms on frequently used benchmark datasets. Finally, we conclude the paper in Section VII.

II. RELATED WORK

Since the first FIM algorithm, Apriori [2], many such algorithms have been presented in the last two decades. These algorithms are classified into different types, such as sequential patterns mining algorithms [24], [25], data stream mining algorithms [26], [27], graph mining algorithms [28], [29], approximate frequent itemset mining in uncertain data [30], [31], and high utility frequent itemset mining algorithms [32], [33]. In this section, we present the FIM algorithms relevant to the research presented in this paper.

The early FIM algorithms, such as Apriori [2], were used to generate a batch of candidate itemsets and visited the database to find support for these candidates. These algorithms utilized the horizontal data format. However, they were inefficient due to the repeated visits made to the database. Another class of algorithms, such as Eclat [3], [12], [34], uses the vertical data format, and transaction data is read once and stored in memory-resident data structures. Though these algorithms are more efficient than Apriori-like algorithms, they are very slow and require a large memory on dense datasets due to the large size of the transaction data.

The pattern-growth-based algorithms, starting with FP-growth [4], [35], represent another class of algorithms that use a compressed data structure called the FP-tree. These algorithms are very memory-efficient because they compress repeating transaction information. However, the

performance of these algorithms is also slow on dense datasets due to the large number of conditional FP trees they create. To address these issues, various authors have suggested different improvements over the years. Recently, the CFSP-growth algorithm was proposed by Jamsheela [6], introducing a structural improvement in the FP-tree to enhance the efficiency of the mining process. In another study, the Dynamic Prefix Tree (DFT) [36] algorithm was proposed to dynamically modify the FP tree and not create additional trees. Yet, these algorithms are less scalable on dense datasets due to the high processing they require.

There is another class of algorithms [37], [38], [39] that uses node-set-based data structures and derives these nodes from the FP tree but does not create conditional FP trees. These algorithms are nearly as memory-efficient as the FP-growth-derived algorithms but more time-efficient than the FP-growth-based algorithms, though their scalability is not better than their predecessors.

Recently, there has been a growing interest in applying heuristic-based techniques to frequent itemset mining [40], [41]. In these algorithms, approaches such as Genetic Algorithm [42], [43], [44] or Particle Swarm Optimization [45], [46], [47] are employed to generate a population of candidate itemsets, utilizing the support of these itemsets as the fitness value. During each iteration of the algorithm, the population is updated by introducing new items into the itemsets. In each update, fit members (itemsets with high support) exert a strong influence on the population, for instance, having more offspring. Although these algorithms aim to be more scalable and efficient in the frequent itemset mining domain, they also suffer from some drawbacks. For example, determining the optimal population size, maintaining significant population diversity, and, notably, dealing with false negatives pose challenges.

FIM is a highly processing-intensive task in dense datasets, primarily due to the frequent execution of support-finding operations. The aforementioned algorithms utilize the entire spectrum of transaction data for support finding and face challenges in achieving high efficiency, and scalability [18]. To enhance the efficiency and scalability of frequent itemset mining (FIM), some researchers have attempted to approximate the support of itemsets. In the earliest of such efforts, sampling techniques were introduced in FIM [18]. In these sample-based FIM algorithms, a sample from the large dataset is selected, and frequent itemsets (FIs) are discovered only in this sample data. In this technique, false negatives are reduced by lowering the support threshold, and false positives are eliminated by an extra pass over the data. However, this step reduces its computational efficiency [48]. Furthermore, the sample size parameter plays an important role in balancing accuracy versus efficiency [49], [50], [51]. This is also true for a very recently introduced parallel sampling-based FIM algorithm [16]. Haifeng et al. [20] used a dynamic sample size but introduced two other parameters. Ordonez [52] explored clustering in FIM to approximate the support of itemsets. Recently, Fatemi et al. [23] used clustering to approximate Maximal Frequent Itemsets (MFI). However, the problem with clustering is that it requires tuning two additional parameters, i.e., the number of clusters and the centroid threshold.

To approximate the size of the intersection of two or more sets, hash-function-based methods have also been studied. Ordonez [53] applied MinHash to frequent itemsets mining. They used the Jaccard similarity index to identify whether a given candidate itemset could be frequent. Cohen et al. [54] developed an estimator based on k-permutation Minwise hashing. Dasu et al. [55] proposed a method based on one permutation MinHash to estimate the size of the intersection of an arbitrary number of sets. Pagh et al. [56] proposed three explicit variations of Apriori that use LSH for computing frequent itemsets. In a recent study in [14] and [21], Zhang utilized the Minwise Hashing technique that uses reduced transaction sets to estimate the support count of itemsets. In another study, Abbasi and Moieni [22] applied a hashing-based Bloom Filter to frequent itemset mining to speed up the support-finding process. However, the problem with the hashing techniques is that they also require additional parameters (namely, the values of K and E). They also store complete transaction data to calculate, instead of approximate, support of itemsets whenever E values are high. They use transaction lists' signatures to approximate support counts of the itemsets. These signatures are again transaction lists, albeit reduced in length (depending on the parameter K).

These hash-based algorithms have the problem that they use complex computations on transaction data. Though they use limited transaction data, it restricts the efficiency and scalability of these algorithms. Similarly, additional parametric settings required in these algorithms make their use more complex.

III. PROBLEM SETTING AND PRELIMINARIES

In this section, we first present a formal definition of the FIM problem. In the next section, we present the data formats commonly used in FIM literature. In the last section, we present a peculiar property of the dense datasets.

A. PROBLEM STATEMENT AND PRELIMINARIES

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n transaction items and $T = \{t_1, t_2, \ldots, t_m\}$ be a set of m transactions, also called transaction dataset, such that each $t_i \subseteq I$. Let $X \subseteq I$ be a $k-$ *size* itemset, called k-itemset. For example, $\{A, B, C\}$ is a 3-itemset which, sometimes for brevity, we will write as ABC. If $Y \subseteq T$ and $\forall y \in Y, X \subseteq$ y then $\sigma(X) = \frac{|Y|}{|T|}$, where $\sigma(X)$ is called the support of X, and $|Y|$ is called the support count of X. It is clear from this definition that $\sigma(X)$ is the probability of the occurrence of itemset X in a randomly picked transaction t from T. The $k-itemset\ X$ is frequent if $\sigma(X) \geq min\_sup$, where *min_sup* is a user-supplied probability that is known as support threshold. Association rule of the form $A \Rightarrow B$ is the probability that a random transaction $t \in T$ contains both of the itemsets A and B simultaneously, given that $A \in I, B \in I$, $A \neq \emptyset, B \neq \emptyset$, and $A \cap B = \emptyset$. The support of rule $A \Rightarrow B$ is

TABLE 1. Horizontal vs. vertical data format.

Transaction ID	Items	Items	Transaction IDs
1	A B D	A	1, 2, 3, 4, 5, 6, 7, 9, 10
2	A B C F	B	1, 2, 3, 5, 7, 8, 9, 10
3	A B C E	C	2, 3, 4, 5, 6, 7, 9
4	A C D F	D	1, 4, 6, 7, 8, 10
5	A B C E F	E	3, 5, 6, 8, 9
6	A C D E	F	2, 4, 5, 10
7	A B C D		
8	B D E		
9	A B C E		
10	A B D F		

the joint probability A and B, that is $P(A \cap B)$, given by $\sigma(AB)$. The confidence of the rule $A \Rightarrow B$ is $P(B|A)$, the conditional probability of B given A has occurred, given by $\sigma(AB)/\sigma(A)$.

A typical association rule mining involves generating all possible rules of the sort mentioned above from a given dataset that has the support and confidence above user-supplied thresholds. The computationally intensive part of this task is frequent itemset mining. This is because, to determine if an itemset X is frequent, it requires enumerating all transactions that contain X. As mentioned, this value is called the support count of X, and it incurs significant computational costs. In this paper, we approximate this value with a technique proposed in the next section.

B. TRANSACTIONS DATA FORMAT

The transaction dataset includes transactions, each with a unique transaction ID (TID) and associated items. For support count calculations of itemsets, FIM algorithms use one of two formats of this data: horizontal or vertical. For example, Table 1 illustrates a transaction dataset with 10 transactions in horizontal format on the left and vertical format on the right. Calculating the support count of an itemset in horizontal format requires counting all transactions that subsume the itemset. In the vertical format, the intersection operation is applied to the TID sets of the items comprising the itemset, and the support count is the size of the final TID set produced.

Since ProbBF is designed for dense datasets, it employs the more efficient diffset approach for the vertical format in dense data introduced in [34]. Lastly, it is important to note that this study deals with traditional certain data only, i.e., the dataset where the presence or absence of an item is certain in a transaction. In uncertain or noisy data, the presence or absence of an item in a transaction is uncertain or probabilistic.

C. PARENT EQUIVALENCE IN DENSE DATA

Parent equivalence (PE) could be stated as follows:

If a frequent itemset X is extended with a frequent item i in the set enumeration tree, then the $supp(X) = supp(X \cup i)$, i.e., the support of the child node is equal to the support of the parent node. The PE property was first utilized in Max-miner [58]. Later on, it was used in the design of MAFIA [59], Genmax [60], and more recently by [10], [37], [38], and [39].

The results in Table 2 show that PE is a condition that occurs more frequently in dense data. For example, when the Chess dataset is mined at 40 percent support, 6,439,702 frequent itemsets are generated. In these frequent itemsets, PE is observed 4,317,130 times, which is 67.04% of 6,439,702. Similarly, an algorithm that uses PE will test only 1,436,968 frequent nodes, which is 77.69% less than 6,439,702. Other datasets in Table 2 show even more promising results. It is also clear from Table 2 that the lower the support threshold, the more the gains from PE. Similar results have been reported about runtime speedup in [59] when the PE property is used. It mined the Connect dataset at 40 percent support twice, with and without PE. It was noted that the runtime dropped from 8423.85 sec to just 20.56 sec when PE was used. PE could also be stated as:

If X and Y are k-itemsets such that $x_i = y_i$, $x_i \in X$, $y_i \in Y$ for $i = 1, 2, \ldots, k-1$, $x_k \neq y_k$, and X and Y represent the set of transactions that subsume these itemsets, then $|X| = |X \cap Y|$, i.e., the $supp(X) = supp(X \cup Y)$. This also means that $P(X) = P(X \cap Y)$. Similarly, the correlation factor of the joint probability of X and Y could be written as:

$$C_f(X, Y) = \frac{P(X \cap Y)}{P(X)P(Y)} = \frac{1}{P(Y)} > 1 \because P(X) = P(X \cap Y)$$

From this discussion, we can draw two conclusions: 1) if S is the immediate proper subset of T, i.e., $S \subset T$, then, more often, $S \rightarrow T$ (in other words, the difference between S and T is very small) in dense datasets, 2) dense data is more frequently positively correlated.

IV. PROPOSED MODEL FOR PREDICTION

In this section, we first present our proposed PSPM prediction model. The next section focuses on evaluating the effectiveness of this model on real-world datasets. In the final section, we introduce upper and lower bounds on the PSPM predictions to further enhance its performance.

A. PROBABILISTIC SUPPORT PREDICTION MODEL

In this subsection, we first derive an equation that we will use to predict the support count of any itemset of size 3. We call this equation the probabilistic support prediction model, or simply PSPM. Later on, we apply this model recursively to find the support of higher order sets of size four and above.

Let A, B, and C be any three finite subsets of a universal set U. According to the probability theory, we know that:

$$P(A \cap B) = P(A) \times P(B \mid A) \tag{1}$$

If A and B are independent events, then $P(B|A) = P(B)$, and (2) becomes:

$$P(A \cap B) = P(A) \times P(B) \tag{2}$$

Another way to write (1) is:

$$P(A \cap B) = P(A) \times P(B) \times C_f(A, B) \tag{3}$$

TABLE 2. Advantage of using PE in FIM algorithms on dense data.

Dataset	Support(%)	Total freq itemsets	Child = Parent (PE)	Total support calculations
Chess	40	6439702	4317130 (67.04%)	1436968 (77.69%)
	35	15108722	10771635 (71.29%)	2855372 (81.10%)
Mushroom	20	53583	50669 (94.56%)	3191 (94.04%)
	10	574431	559183 (97.35%)	15957 (97.22%)
Connect	70	4129839	3900516 (94.45%)	41072 (99%)
	60	21250671	20483260 (96.39%)	80115 (99.62%)
PUMSB	70	2698264	1586521 (58.8%)	683914 (74.65%)
	65	8094688	5479298 (67.69%)	1429116 (82.34%)
PUMSB_star	40	27354	19182 (70.12%)	4866 (82.21%)
	30	432698	350979 (81.11%)	39893 (90.78%)

Here, C_f shows the correlation of two probabilistic events A and B, and it is written as follows:

$$C_f = \frac{P(A \cap B)}{P(A) \times P(B)} \tag{4}$$

Let's designate C_f as the correlation factor. If A and B are independent events, then $C_f(A, B) = 1$, and (3) equals (2). Although (1) and (3) are equal, the advantage of (3) over (1) is that the data dependence has been separated. If $P(A \cap B)$, $P(A \cap C)$, and $C_f(B, C)$ are known, and $X = (A \cap B)$, $Y = (A \cap C)$, then we can prove that:

$$\mathrm{P((X \cap Y)|A) = P(X|A) \times P(Y|A) \times C_f((X, Y)|A)} \tag{5}$$

Lemma#1

$$P((X \cap Y)|A) = P(X|A) \times P(Y|A) \times C_f((X \cap Y)|A)$$

Proof:

$$\begin{aligned}
P((X \cap Y)|A) &= \frac{P((X \cap Y) \cap A)}{P(A)} \\
&= \frac{|(X \cap Y) \cap A| \,/|U|}{|A| \,/|U|} \\
&= \frac{|X \cap Y \cap A|}{|A|} \left[\frac{|A|}{|A|} \times \frac{|X|}{|X|} \times \frac{|Y|}{|Y|}\right] \\
&= \frac{|X|}{|A|} \times \frac{|Y|}{|A|} \times \frac{|X \cap Y \cap A| \times |A|}{|X| \times |Y|} \\
&= \frac{|X \cap A|}{|A|} \times \frac{|Y \cap A|}{|A|} \\
&\quad \times \frac{|X \cap Y \cap A| \times |A|}{|X| \times |Y|} \because X \subset A \\
&= P(X|A) \times P(Y|A) \times \frac{|X \cap Y \cap A| \times |A|}{|X| \times |Y|} \\
&\quad \times \left[\frac{|A|}{|A|}\right]
\end{aligned}$$

$$\begin{aligned}
\frac{|X \cap A|}{|A|} &= \frac{|X \cap A|/|U|}{|A|/|U|} = P(X|A) \\
&= P(X|A) \times P(Y|A) \times \frac{|X \cap Y \cap A|/|A|}{|X|/|A| \times |Y|/|A|} \\
&= P(X|A) \times P(Y|A) \times \frac{P((X \cap Y)|A)}{P(X|A) \times P(Y|A)} \\
&= P\,(X\,|\,A) \times P\,(Y\,|\,A) \times C_f\,((X, Y)\,|\,A)\ \blacksquare
\end{aligned}$$

It is evident from (5) that $(X \cap Y) = (A \cap B \cap C)$, but $(A \cap B \cap C)$ is the quantity we want to determine. In Section III-A, we established for dense data that $S \rightarrow T$ more frequently if S is the immediate proper subset of T. Given that X, Y, and A are the immediate proper subsets of B, C, and U respectively, we can infer that $X \rightarrow B$, $Y \rightarrow C$, and $A \rightarrow U$. Consequently, we assume that the correlation of B and C in U approximates the correlation of X and Y in A. This assumption is based on the logic that if $X \rightarrow B$, $Y \rightarrow C$, and $A \rightarrow U$, then $C_f((X, Y)|A) \cong C_f(B, C)$. Therefore, we can approximate the probability of $P((X \cap Y)|A)$ by substituting $C_f(B, C)$ for $C_f((X, Y)|A)$ in (5).

$$P_a((X \cap Y) \mid A) = P(X \mid A) \times P(Y \mid A) \times C_f(B, C) \tag{6}$$

where P_a is the approximate probability. This model will work well if either our assumption holds or sets B and C are very weakly dependent, i.e., $C_f(B, C) \cong 1$. If we have the probability of a set, we can easily determine its cardinality, i.e., support count in FIM, with the following equation.

$$|V| = P(V) \times |U| \because P(V) = \frac{|V|}{|U|} \tag{7}$$

where $|V|$ is called the cardinality, representing the number of elements in the set V. U is the total sample space. It is worth noting that, in the context of FIM, U represents all transactions in the dataset. Similarly, A, B, C, X, and Y are frequent items or itemsets representing the set of transactions that are the subsets of U. The cardinality $|V|$ is known as support count in the context of itemset mining. Moreover, the basic idea is to apply (6) to calculate the support count probabilistically, i.e., without visiting the transaction dataset. Our algorithm, ProbBF, uses the following two equations, which are reduced forms of (6), for predicting the support count of $k - size$ itemsets for $k \geq 3$.

$$\begin{aligned}
&|A \cap B \cap C| \\
&= \frac{|A \cap B| \times |A \cap C| \times |B \cap C| \times |U|}{|A| \times |B| \times |C|}
\end{aligned} \tag{8}$$

$$\begin{aligned}
&|Pref \cap A \cap B \cap C| \\
&= \frac{|Pref \cap A \cap B| \times |Pref \cap A \cap C| \times |Pref \cap B \cap C| \times |Pref|}{|Pref \cap A| \times |Pref \cap B| \times |Pref \cap C|}
\end{aligned} \tag{9}$$

When $k = 3$, (8) is used. In this equation, $|U|$ represents the number of transactions in the transaction dataset. When $k \geq 4$, (9) is used. In this equation, the last 3 items of a $k-$ itemsetare represented with A, B, and C, respectively. *Pref* (for prefix) is an itemset of size $k-3$. For example, if $HIJKL$ is a $5-itemset$, then J, K, and L are the A, B, and C, respectively, and HI is the *Pref* itemset. The significance of the recursive application of (6) in the form of (9) is that the sample space $|pref|$ continuously reduces. It is also important to note that support counts of all the sets used in (9) would already be known. The flowchart of the PSPM-based ProbBF algorithm is depicted in Fig. 1.

B. EVALUATING THE PSPM ON REAL-WORLD DATASETS

To assess the predictive efficiency of the PSPM on dense data, we have chosen six real-world dense datasets frequently encountered in the FIM literature, namely Chess, Accidents, Connect, Pumsb, Kddcup99, and Pumsb_star. They can be easily downloaded from the FIMI repository (http://fimi.ua.ac.be) or the datasets of the Java open-source library SPMF at (https://www.philippe-fournier-viger.com/spmf/).

In this section, we evaluate the validity of our basic assumption in the PSPM on these real-world datasets. Additionally, we employ statistical measures to quantitatively assess the quality of predictions made by the PSPM for 3-itemsets on these datasets.

1) EVALUATING OUR KEY ASSUMPTION IN PSPM

As we mentioned earlier, if $A \rightarrow U$, $X \rightarrow B$, and $Y \rightarrow C$ (i.e., A is close in size to U, X is close in size to B and Y is close in size to C), then $C_f((X,Y)|A) \cong C_f(B,C)$. The differences among these sets (namely, $U-A$, $B-X$, and $C-Y$) and their effects on error signal are shown in Fig. 2 To avoid clutter, $C-Y$ is not displayed in the figure. In this figure, the x-axis represents the frequent itemsets of their corresponding dataset of size $k = 3$. These frequent itemsets are obtained by combining frequent items in their decreasing order of support. It is indicated for all six datasets that the more these differences grow, the more the error signal increases. It is important to note that due to the large values of $U-A$, the proposed technique fails to produce good predictions for the Pumsb_star dataset. This is because the Pumsb_star dataset is obtained by deleting items from the Pumsb dataset that appear in 80% or above transactions. This surely weakens our assumption $A \rightarrow U$. A similar situation could be observed for all these datasets when frequent items of low support are joined with other frequent items of low support. This is the reason for the gradual increase in error values when we move from left to right in the error graphs shown in Fig. 2 These error values are calculated using the following equation.

$$Error = \frac{\text{Predicted support} - \text{Real support}}{\text{Real support}} \times 100 \qquad (10)$$

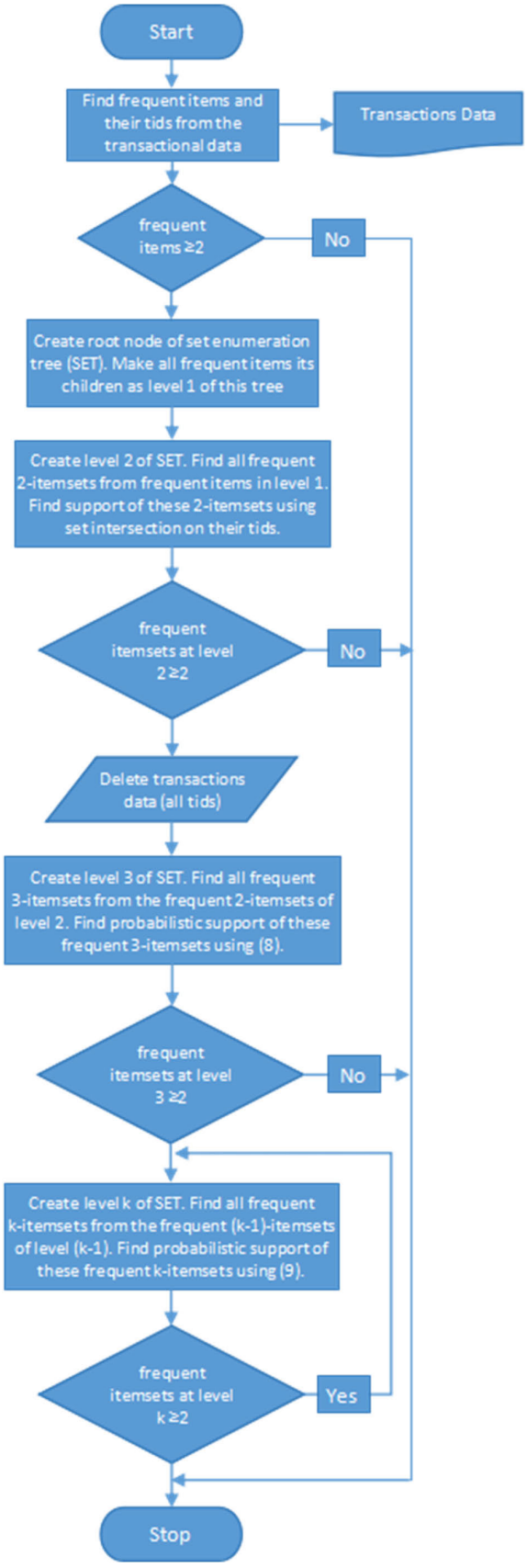

FIGURE 1. The flow chart of the PSPM-based ProbBF Algorithm.

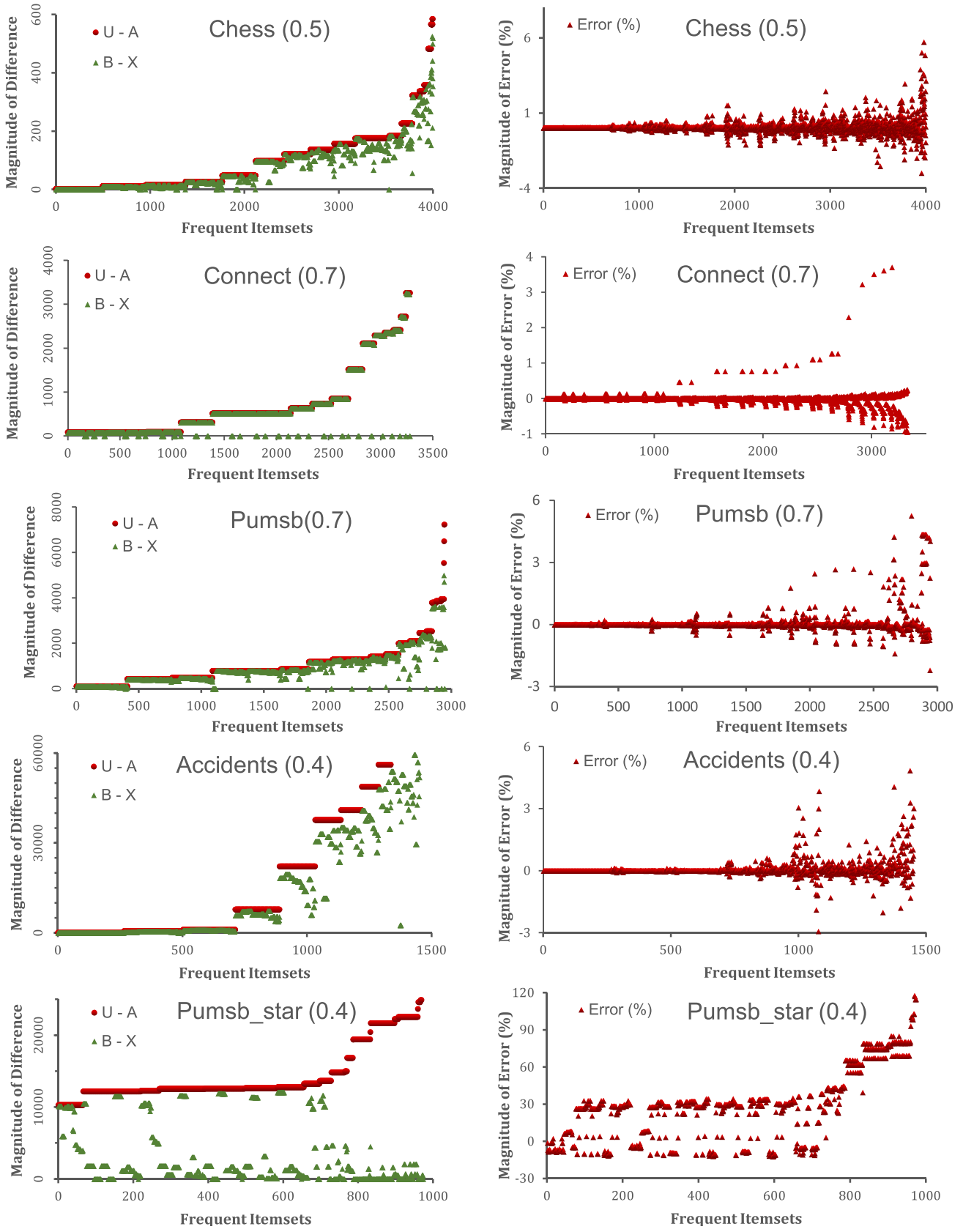

FIGURE 2. Investigation of our assumption (Left) and its effects on error (Right) in real-world datasets.

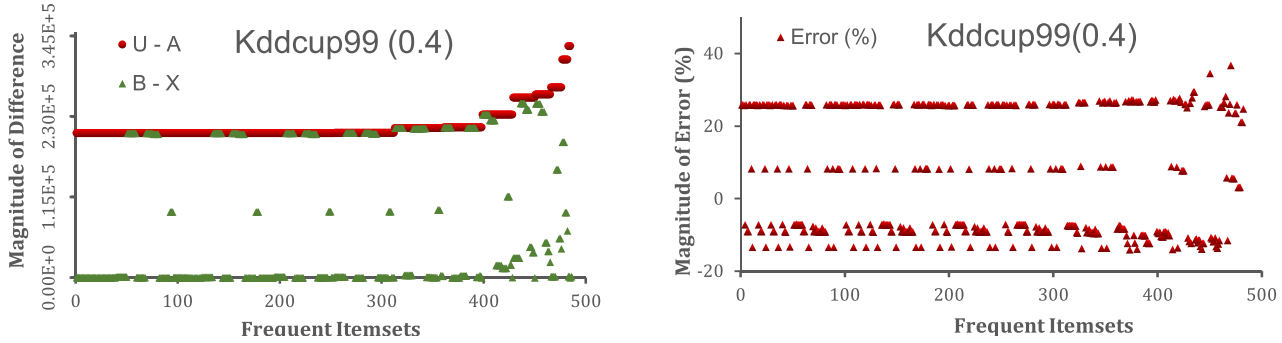

FIGURE 2. *(Continued.)* **Investigation of our assumption (Left) and its effects on error (Right) in real-world datasets.**

TABLE 3. R-squared and Mean values.

	|Chess(0.5)	|Connect(0.7)	|Mushroom(0.4)	|Pumsb(0.7)	|Pumsb_star(0.4)
A. Mean	|0.000792925	|-0.000280678	|-0.005391111	|-0.000748284	|28.0638809
R-Squared	|0.999594802	|0.999540763	|0.990718543	|0.981318892	|0.264674047

It is worth noting that many of these error values at the right end of the graph are still low. This is attributed to the second case mentioned in the previous subsection, i.e., $C_f(B, C) \cong 1$ (the dependence between sets B and C is very weak). The results in another figure, Fig. 3, also confirm the observations recorded in Fig. 2. In this figure, the same frequent items of Fig. 2 have been used. On the left side of the figure, graphs have been plotted for the real support and the predicted support. These graphs display frequent itemsets sorted in ascending order of their real support. On the right side of the figure, graphs depict the $C_f(B, C)$ and $C_f(X, Y)$. In these graphs, frequent itemsets are sorted in ascending order of $C_f(X, Y)$ values. Important observations from Fig. 3 are as follows: 1) predicted support values sharply mirror the graph of real support values, 2) the discrepancies, though very few, between the real support and predicted support values are more pronounced for low support frequent itemset, 3) $C_f(B, C)$ values closely track the graph of $C_f(X, Y)$, and 4) noticeable discrepancies between the values of $C_f(X, Y)$ and $C_f(B, C)$ are low in numbers but distributed uniformly across the frequent itemsets. Consequently, the findings in Fig. 2 and Fig. 3 indicate that the proposed technique achieves higher accuracy with denser datasets and a higher support threshold. In Fig. 3, our premiss that $C_f(B, C)$ could approximate $C_f(X, Y)$ is further validated because the signal $C_f(B, C)$ follows the signal $C_f(X, Y)$ to a greater extent, in at least, some of the datasets.

2) EVALUATING THE QUALITY OF PSPM PREDICTIONS

In this subsection, we introduce well-known statistical quantitative measures that indicate the effectiveness of a prediction model when it attains specified results. The quantification of this effectiveness is rooted in the evaluation of the error term known as the residue. The residue is the difference between the observed value and the predictive value. The foremost important assumption from the fitted model is that the residuals should be normally distributed around zero mean. The arithmetic mean for all six datasets has been shown in TABLE 3. Except for the Pumsb_star dataset, all the values are very close to zero. The normality of the distribution of residuals around the mean is shown in Fig. 4. The bar graphs on the left side of this figure are bell-shaped and peak at zero. The normal probability plot shown on the right side of Fig. 4 is also normally distributed around 0 mean, though the residuals are more heavily concentrated around the mean. It is worth noting that the values of support count predictions used in the calculations in Fig. 4 are not adjusted for bounds introduced in Section D.

The coefficient of determination, also known as R-squared (R2), is another widely used measure of goodness of fit to determine how well a predictive model fits the observed values. The R-squared value represents the extent to which the variance present in the dependent variable is predicted by the independent variables. It ranges between 0 and 1, with 0 indicating a poor fit and 1 indicating a perfect fit. R-squared values calculated for all six datasets are presented in Table 3. Since these values are very close to 1, it indicates that our proposed model is highly effective in its predictions. R-squared is graphically represented by plotting observed values against the fitted line to illustrate how well the variance in the observed values is captured by the fitted line. This is shown in Fig. 3, where 'Real Support' represents the observed line, and 'Predicted Support' represents the fitted line. The figure demonstrates that the fitted line closely follows the observed line, except for the Pumsb_star dataset (the reason for this has already been discussed earlier in this section).

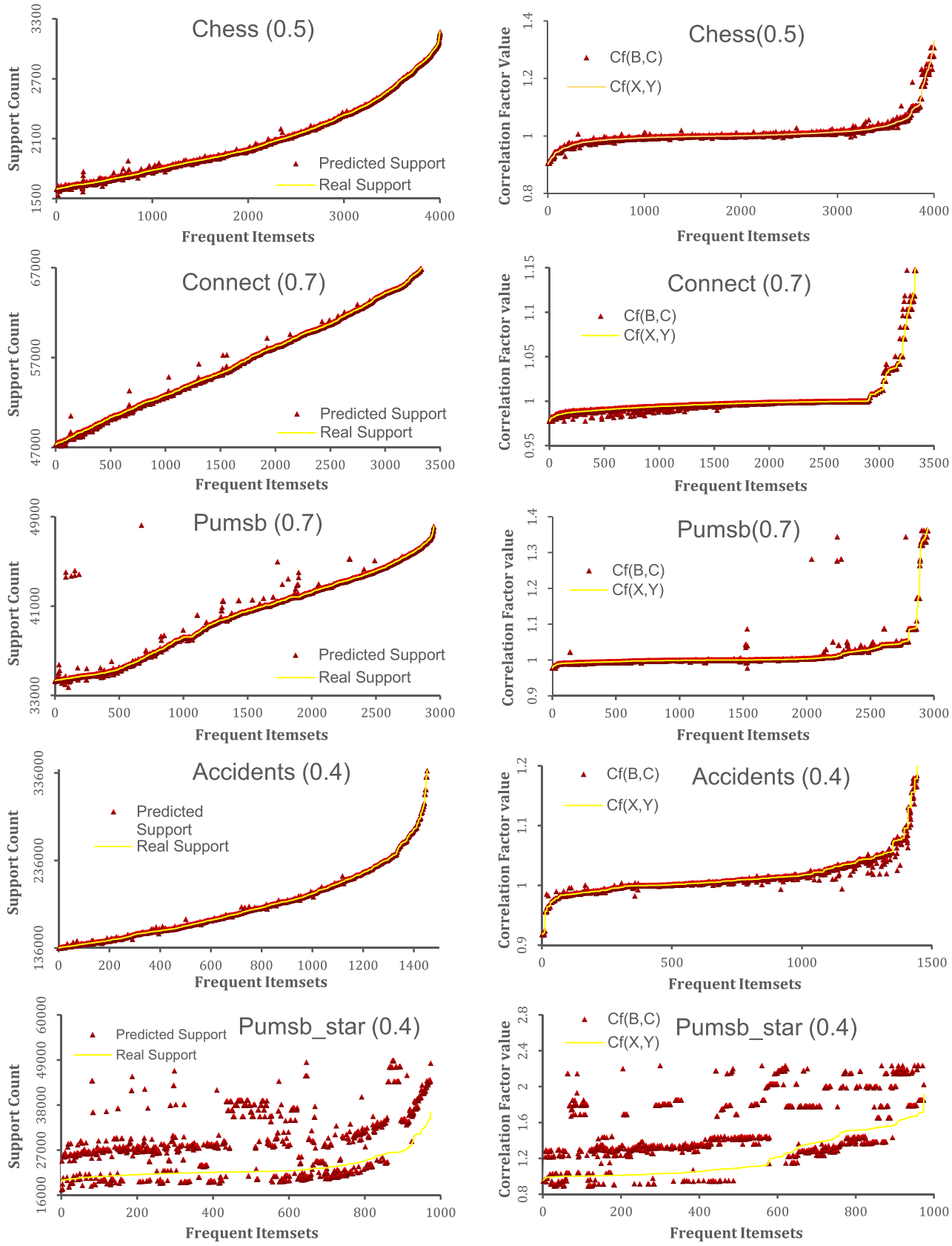

FIGURE 3. Comparison of how closely follows predicted support the real support (left) and substitute correlation the actual correlation (Right).

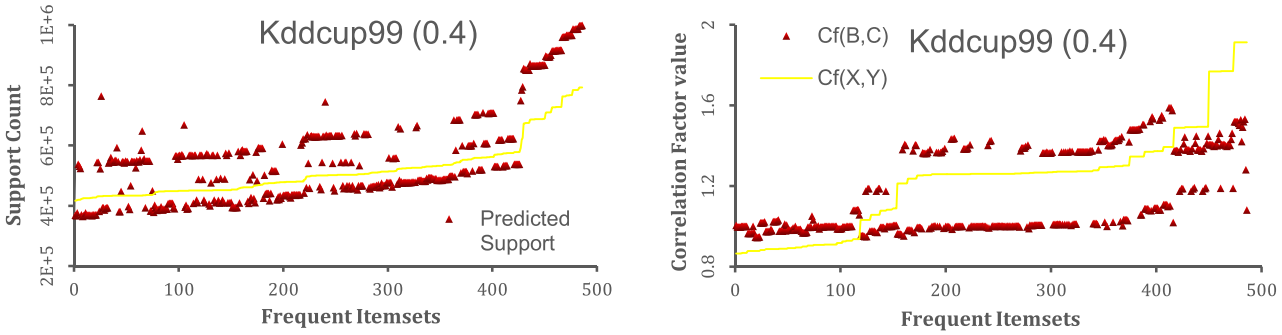

FIGURE 3. *(Continued.)* **Comparison of how closely follows predicted support the real support (left) and substitute correlation the actual correlation (Right).**

C. REINFORCING PSPM PREDICTIONS WITH ERROR BOUNDING

There is quite a possibility that a prediction falls in neither of the two cases (namely, $C_f((X,Y)|A) \cong C_f(B,C)$ and $C_f(B,C) \cong 1$) identified in the previous subsections. Hence, it is necessary to restrict the maximum error possible. According to the anti-monotone property of the support-finding process [49], if an itemset P is extended with an item i, then $\sigma(P \cup i) \leq \sigma(P)$. Based on this property, the following equation defines the upper bound (Ub) on support prediction.

$$\sigma(A \cup B \cup C) \leq min(X, Y, Z) = Ub \tag{11}$$

In the above equation, we know that $X = A \cup B$, $Y = A \cup C$ and $Z = B \cup C$ (It is worth noting that we wrote $X = A \cap B$ in the previous sections where A, B, and X were representing the set of transactions. In that context, the set X could be obtained by taking the intersection of sets A and B. However, when A, B, and X are treated as itemsets, $X = A \cup B$ signifies that X is obtained by combining itemsets A and B). Similarly, a lower bound on the support of a 3-itemset could be defined using the following equation.

$$\sigma\,(A \cup B \cup C) \geq \sigma\,(A \cup B) + \sigma\,(A \cup C) - \sigma(A) \tag{12}$$

This equation was first used in Max-miner [58]. Consequently, a lower bound (Lb) on the support prediction of a 3-itemset could be defined as the following equation.

$$\sigma(A \cup B \cup C) \geq max \begin{pmatrix} \sigma\,(X) + \sigma\,(Y) - \sigma\,(A)\,, \\ (\sigma\,(X) + \sigma\,(Z) - \sigma(B), \\ \sigma\,(Y) + \sigma\,(Z) - \sigma(C) \end{pmatrix} = Lb \tag{13}$$

The predicted support could now be clipped on either of these theoretical bounds if it crosses it. Let ϑ represent the maximum limit on the error (i.e., the difference between the real support and predicted support), defined as follows.

$$Error \leq \vartheta = Ub - Lb - 1 \tag{14}$$

In the above equation, ϑ defines the upper limit on error (i.e., the error is now constrained in $[0, \vartheta]$). The -1 in the above equation signifies a unique situation when $\sigma(A \cup B \cup C) = min(X, Y, Z)$. In this scenario $Lb = Ub$.

The effect of introducing bounds on support count predictions is illustrated in Fig. 5. The error graph shown in Fig. 2, and the support count graph shown in Fig. 3, both have been reproduced in Fig. 5. However, this time, the values of the support predictions are capped in the range $[0, \vartheta]$. The improvement in the accuracy of the predicted support values could be observed by comparing the results shown in Fig. 5 to the results shown in Fig. 2 and Fig. 3. These improvements are particularly noticeable in cases where neither $C_f((X,Y)|A) \cong C_f(B,C)$ nor $C_f(B,C) \cong 1$.

V. THE PROPOSED ALGORITHM, ProbBP

In this section, the proposed algorithm ProbBF is introduced. This algorithm models the search space as the set-enumeration tree [61]. Every node N in this tree has four items of information: the item, the support count, a reference to its parent, and a collection of references to its children. Each node represents a $k - size$ frequent itemset, where k is equal to the level of the tree at which it is situated. The root node is positioned at level 0. This $k - itemset$ is obtained by $N.item \cup N.Parent \cup N.Parent.Parent \cup \ldots$, and $N.Support \leq N.Parent.Support$. Item at the root node is set to empty set $\emptyset$, while the support is set to the total number of transactions. This is because ProbBF treats the root node as the universal set U. To construct this tree, ProbBF employs the following four algorithms.

Algorithm 1: This algorithm, presented in Fig. 6, takes the transaction dataset $\mathcal{D}$ as input. The dataset comprises a total of $|\mathcal{D}|$ transactions, where each transaction $t \subseteq I$. The algorithm processes each transaction $t \in \mathcal{D}$ and each item $i \in t$ to build $F1$. $F1$ is a collection of the form $< item, tids^* >^*$, where symbols $< >$ and $*$ denote composite structure and multiplicity (or collection), respectively. This implies that each item in $\mathcal{D}$ has a record in $F1$, including its associated transaction IDs. If the support of an item is below the user-supplied support threshold min_sup, it is deleted, leaving $F1$ with only frequent items. ProbBF then invokes Algorithm 2 and Algorithm 3, respectively. Algorithm 2 creates the root node of the frequent itemset tree, populates its first level with frequent items from $F1$ received as input from

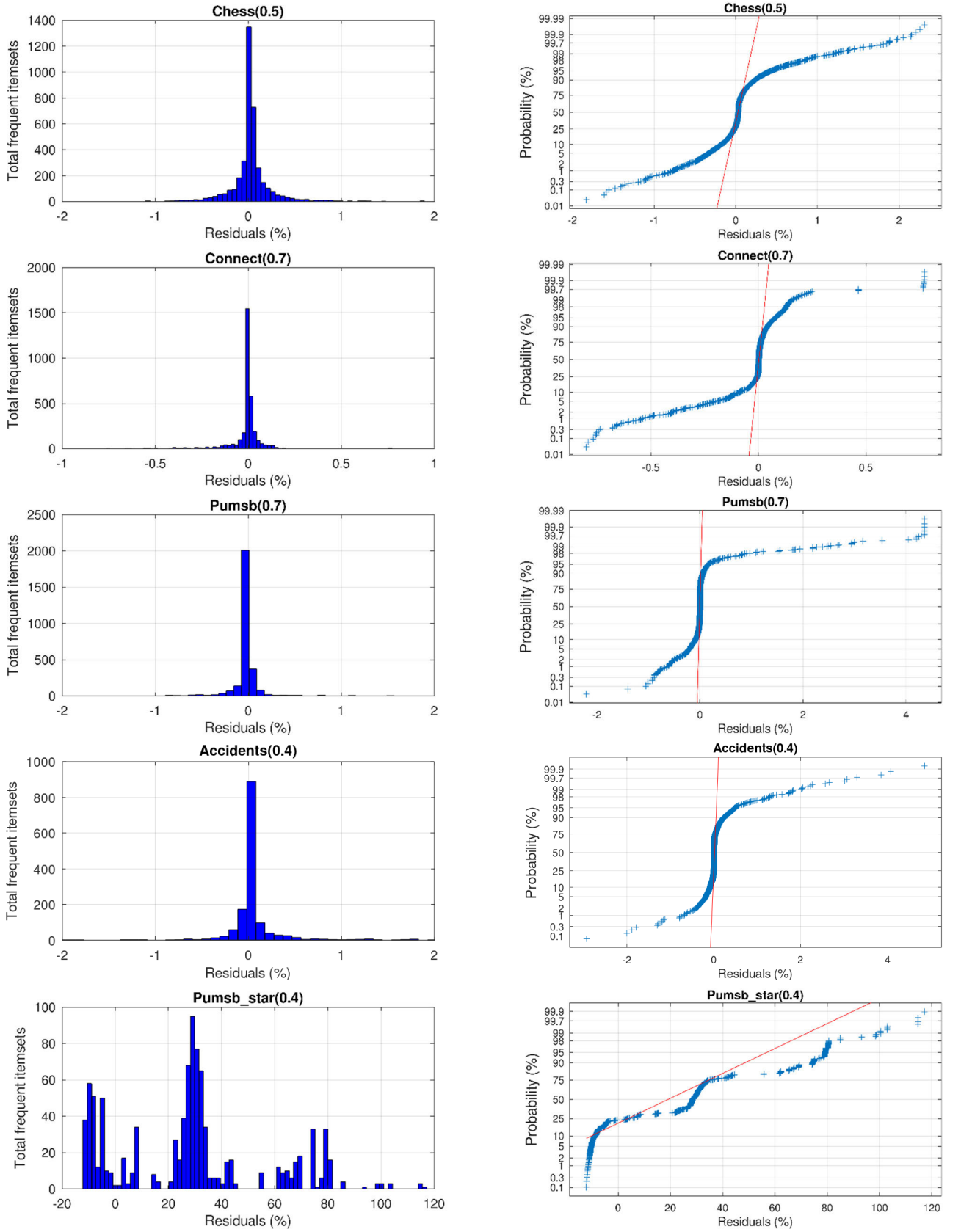

FIGURE 4. Quantitative assessment of quality of predictions. Residual distribution (left) and normal probability distribution (right).

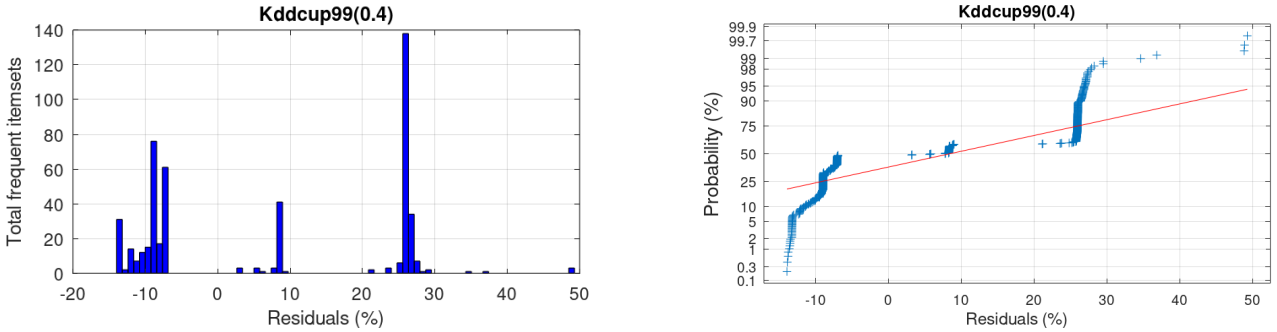

FIGURE 4. *(Continued.)* **Quantitative assessment of quality of predictions. Residual distribution (left) and normal probability distribution (right).**

ProbBF, and returns a reference to the root node. Algorithm 3 generates the rest of the levels of this tree.

Algorithm 2: In Fig. 7, Algorithm 2 creates the root node and a child of the root for each frequent item in $F1$.

Algorithm 3: This algorithm, presented in Fig. 8, plays a major role in the ProbBF algorithm as it creates levels beyond 0 and 1 in the frequent itemset tree. It utilizes a queue data structure to store nodes with children, starting with the root node. The algorithm retrieves a node from the front of the queue and sequentially visits each of its children. The visited child is extended with all its subsequent siblings that are frequent. If a child has one or more frequent extensions, it is also inserted into the queue. This process continues until the queue becomes empty, indicating that no node in the tree is left with possible frequent extensions.

This algorithm calculates the support count of the new nodes in two different ways. If the new node belongs to level 2 of the tree, i.e. it makes frequent $2-itemset$, then it uses transactional data to find support. Otherwise, it calls Algorithm 4 to probabilistically predict the support count of the new $k-itemset$ node for $k \geq 3$.

Algorithm 4: This algorithm, depicted in Fig. 9, calculates the probabilistic support of a node as suggested in (8) and (9).

Example: The ProbBF algorithm has been applied to the dataset mentioned in Table 1 for the support threshold of 2 transactions. The set enumeration (SE) tree the ProbBF builds is shown in Fig. 10. The ProbDF routine (i.e., the algorithm 1) first finds all the frequent items from the dataset. For the support threshold of 2, all items present in this dataset are already frequent. The ProbBF routine sends the list of these frequent items to the initializeRoot routine (i.e., the algorithm 2). This routine creates the root node of the SE tree and populates its level 1 with the frequent items it receives as arguments. The root node and its frequent children can be observed in Fig. 10. The initializeRoot routine eventually returns a reference to the root node of this SE tree to the ProbBF. The ProbBF then calls the genFreqTree routine (i.e., the algorithm 3), and passes it the reference to the root node. Starting from level 2, it is the genFreqRoutine that creates the rest of the levels of the SE tree. For the support calculations of itemsets at level 2, the genFreqRoutine uses the traditional tids-based method. These tids are deleted after all the frequent 2-itemsets are found. Starting from level 3, genFreqRoutine uses PSPM to probabilistically calculate the support of the itemsets. For this purpose, it calls the genProbSupp routine (i.e., the algorithm 4). The complete SE tree for the dataset in Table 1 is shown in Fig. 10.

All the frequent itemsets generated by the ProbBF are also shown level-wise in more detail in Table 4. Instead of just for level 1 and level 2, the tids intersection-based support [3] is shown for the frequent itemsets of all the levels. For the itemsets at level 3, it uses (8) to calculate PSPM-based probabilistic support of these itemsets. For example, the probabilistic support of the itemset {F, D, B}, abbreviated as FDB, is calculated using (8) as follows.

$$support(FDB) = \frac{|F \cap D| \times |F \cap B| \times |D \cap B| \times |U|}{|F| \times |D| \times |B|}$$

$$support(FDB) = \frac{2 \times 3 \times 4 \times 10}{4 \times 6 \times 8} = 1.25$$

It uses (9) to calculate PSPM-based support of the itemsets at level 4. For example, the probabilistic support of itemset FCBA is calculated as follows.

$$support(FCBA) = \frac{|F \cap C \cap B| \times |F \cap C \cap A| \times |F \cap B \cap A| \times |F|}{|F \cap C| \times |F \cap B| \times |F \cap A|}$$

$$support(FCBA) = \frac{2.01 \times 3.33 \times 2.91 \times 4}{3 \times 3 \times 4} = 2.164 \approx 2.17$$

Evaluating the quality of ProbFB's output in terms of false positives and false negatives, it can be observed that no false positives are produced in the tree. A rectangle with a dashed line border, labeled FDA, indicates the single false negative node in this tree. This suggests that false outputs are always in the vicinity of the frequent itemsets boundary (i.e., their support is close to the minimum support threshold). Therefore, ProbBF has the slightest chance of missing high-support frequent itemsets. Since the worth of an itemset is derived from its support (i.e., the higher the support value, the higher its importance), ProbBF proves to be an effective approximate FIM algorithm. These observations are further reinforced

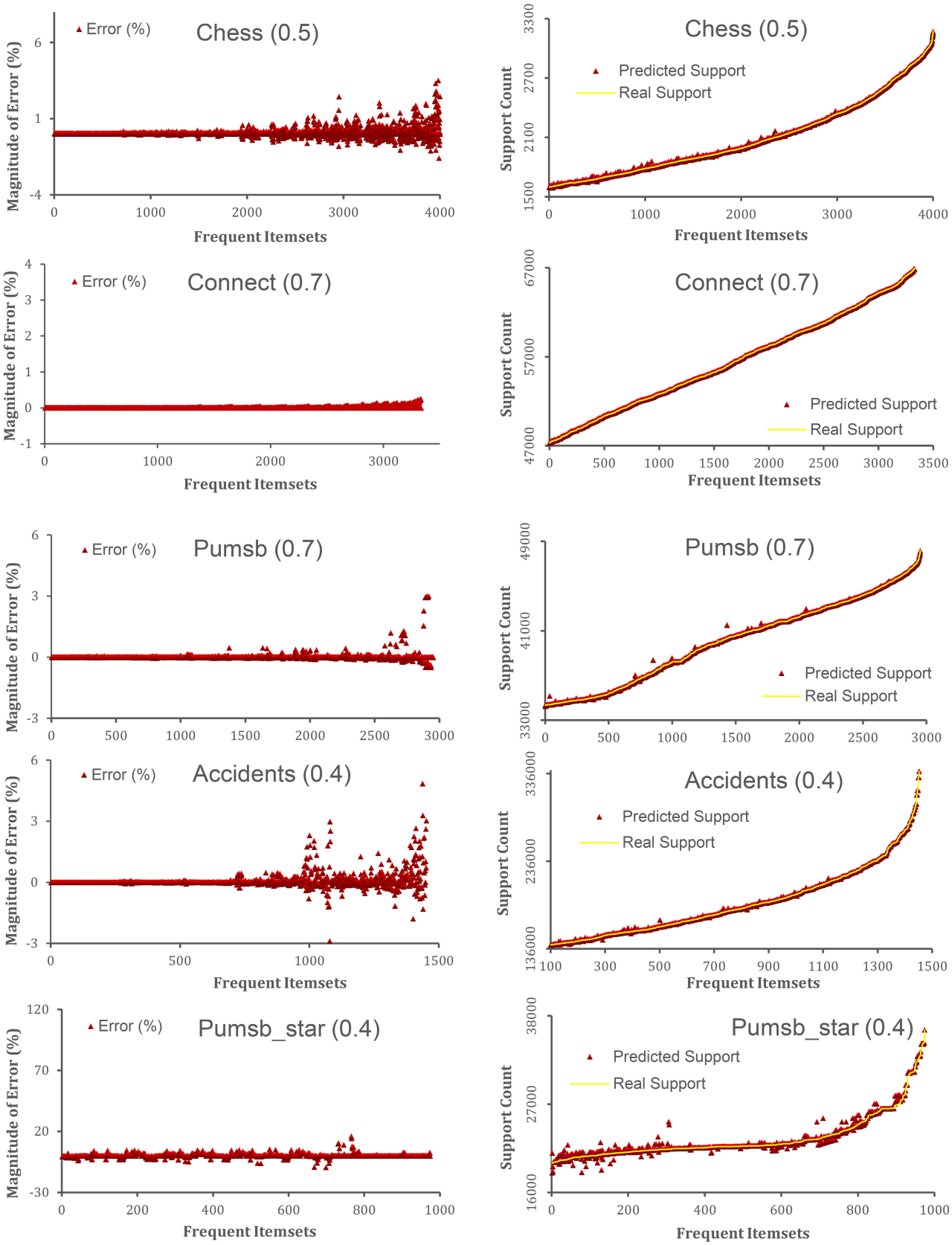

FIGURE 5. Improved accuracy of support count predictions after introducing bounds.

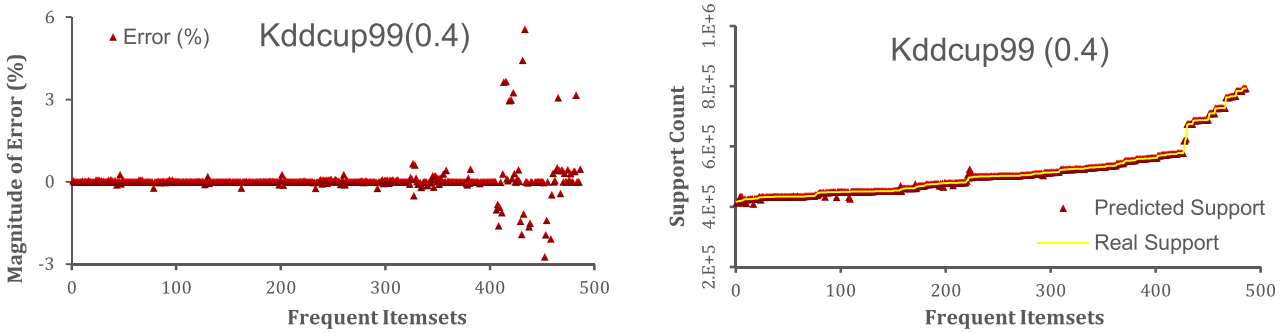

FIGURE 5. *(Continued.)* **Improved accuracy of support count predictions after introducing bounds.**

TABLE 4. **The ProbBF algorithm applied to data in Table 1. In this table AB is short for {A,B}, |A∩B| is the size of tids(A)∩tids(B), the universal set U=10 (total transactions), and this dataset has zero false positive and FDA (written in bold font) the only false negative.**

Level 1		Level 2		Level 3			Level 4		
1-Itemsets	**Support**	**2-Itemsets**	**Support**	**3-itemsets**	**PSPM Support**	**Support**	**4-itemsets**	**PSPM Support**	**Support**
X	|X|	XY	|X∩Y|	XYZ	$\frac{(X\cap Y)(X\cap Z)(Y\cap Z)U}{XXYXZ}$	|(XY)∩(XZ)|	WXYZ	$\frac{(W\cap X\cap Y)(W\cap X\cap Z)(W\cap Y\cap Z)W}{(W\cap X)(W\cap Y)(W\cap Z)}$	|(WXY)∩(WXZ)|
F	4	FD	2	FDB	1.25	1	FCBA	2.17	2
E	5	FB	3	FDC	1.71	1	ECBA	2.46	3
D	6	FC	3	**FDA**	**1.85**	**2**			
C	7	FA	4	FCB	2.01	2			
B	8	ED	2	FCA	3.33	3			
A	9	EC	4	FBA	2.91	3			
		EB	4	EDC	1.14	1			
		EA	4	EDB	1.33	1			
		DC	3	EDA	1.48	1			
		DB	4	ECB	2.85	3			
		DA	5	ECA	3.55	4			
		CB	5	EBA	3.11	3			
		CA	7	DCB	1.77	1			
		BA	7	DCA	2.77	3			
				DBA	3.24	3			
				CBA	4.86	4			

Input: $\mathcal{D}$: Transactions Dataset, min_sup: Support Threshold
Output: root: Root of complete frequent itemset tree

1. F1 = ∅ //As <item, tids*>*
2. $foreach\ tran_i \in \mathcal{D}, \forall i = 1,2,\ldots,|\mathcal{D}|\ do$:
3. | $for\ item \in t\ do$:
4. | | $if\ NOT\ F1.isItemPresent(item)$
5. | | | $F1.addItem(item)$
6. | | $F1.addTransaction(item, i)$
7. $min\_count = \lceil min\_sup \times |\mathcal{D}| \rceil$ //min support count
8. $foreach\ f_i \in F1, \forall i = 1,2,\ldots,|F1|\ do$:
9. | $if\ |f_i.tids| < min\_count$
10. | | $F1.deleteItem(f_i)$ //delete infrequent item
11. $|x_i.tids| \geq |x_{i+1}.tids|, \forall(x_i, x_{i+1}) \in F1$ //sorting F1 in ascending
12. $root = initializeRoot(F1, |\mathcal{D}|)$
13. $root = genFreqTree(root, F1, min\_count)$
14. $Return\ root$

FIGURE 6. **ProbBF algorithm (Algorithm 1).**

Input: $F1$: $As\ < item, tids^* >^*, t\_trans$: $As\ |\mathcal{D}|$
Output: $root$: $level\ 0\ and\ level\ 1\ of\ frequent\ itemsets\ tree$

1. $CREATE\ root\ //As < ItemID, Support, Parent, Children^* >$
2. $root.Parent = \emptyset$
3. $root.Support = t\_trans$
4. $foreach\ t \in F1$
5. | $CREATE\ child\ //As < itemID, Support, Parent, Children *>$
6. | $child.itemID = t.itemID$
7. | $child.Support = |t.tids|$
8. | $child.Parent = root$
9. | $root.addChild(child)$
10. $Return\ root$

FIGURE 7. **Initialize root Routine initializeRoot (Algorithm 2).**

when ProbBF is applied to real-world datasets in the next section.

VI. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our proposed algorithm, ProbBF, against seven state-of-the-art frequent itemset mining algorithms, namely HashECLAT [21], CL-MAX [23], CFSP growth [6], PrePost+ [37], LCM [62], negFIN [39], and RSP-PFI [16]. HashEclat, CL-MAX, and RSP-PFI are state-of-the-art approximate FIM algorithms. CFSP-growth is a state-of-the-art FP-tree-based FIM algorithm, and negFIN is the nodeset-based state-of-the-art FIM algorithm. The remaining two, PrePost+ and LCM, are

Input: $root < ItemID, Support, Parent, Child >, F1, min\_sup\_count$
Output: $root$: *Root node of Complete Frequent Itemset Tree*

1. Q = ∅ //create an empty queue data structure
2. Q. Enqueue(root)
3. *While* $Q \neq \emptyset$
4. | temp = Q.Dequeue(Q)
5. | if Q ≠ ∅ and F1 ≠ ∅ //Delete F1 before creating level 3
6. | | Delete F1
7. | *foreach* $child\_a \in temp.children$ *do*:
8. | | *foreach* $child\_b \in temp.Children, child\_b < child\_a$
9. | | | *if* $child\_a.Parent = NULL$
10. | | | | d_sup = |F1[child_a.item].tids − F1[child_b.item].tids|
11. | | | | sup = child_a.Support − d_sup
12. | | | | *if* $sup \geq min\_sup\_count$
13. | | | | | CREATE child //<itemID,Support, Parent, Children*>
14. | | | | | $child.itemID = child\_b.itemID$
15. | | | | | $child.support = sup$
16. | | | | | $child.Parent = child\_a$
17. | | | | | $child\_a.addChild(child)$
18. | | | *else*
19. | | | | $ExpSupp = genExpSupp(child\_a, child\_b.itemID)$
20. | | | | *if* $ExpSupp \geq min\_supp\_count$
21. | | | | | CREATE child //<itemID, Support, Parent, Children*>
22. | | | | | $child.itemID = child\_b.itemID$
23. | | | | | $child.support = ExpSupp$
24. | | | | | $child.Parent = child\_a$
25. | | | | | $child\_a.addChild(child)$
26. | | *if* $child\_a.Children \neq NULL$
27. | | | $Q.Enqueue(child\_a)$
28. *Return root*

FIGURE 8. Build frequent tree routine genFreqTree (Algorithm 3).

Input: X, Y: Both of type < itemID, Support, Parent, Children >
Output: Probabilistic support of (X ∪ Y)

1. $B\_id = X.itemID$
2. $C\_id = Y.itemID$
3. $U = X.Parent.Parent$
4. $A = X.Parent$
5. $B = U.getChild(B\_id)$
6. $C = U.getChild(C\_id)$
7. $Z = B.getChild(C\_id)$
8. $Supp = \frac{X.Support * Y.Support * Z.Su \quad * U.Support}{A.Support * B.Support * C.Support}$
9. *If* $Supp > min(X, Y, Z)$
10. | $Supp = min(X, Y, Z)$
11. *if* $Supp < max(X + Y - A, X + Z - B, Y + Z - C)$
12. | $Supp = max(X + Y - A, X + Z - B, Y + Z - C)$
13. *Return Supp*

FIGURE 9. Probabilistic support routine genProbSupp (Algorithm 4).

also efficient FIM algorithms. For instance, LCM was the top-performing algorithm in FIMI'04.

To evaluate ProbBF's performance on dense data, we chose the same six datasets—Chess, Accidents, Connect, Pumsb, Kddcup99, and Pumsb_star—that were used in Section IV-B. These datasets are commonly employed as benchmarks in the frequent itemset mining literature.

In Section VI-A and Section VI-B, we compare the time efficiency and memory efficiency, respectively, of the ProbBF algorithm against the seven benchmark algorithms selected for comparison. In Section VI-C, we analyze the capacity of our algorithm to generate all possible frequent itemsets.

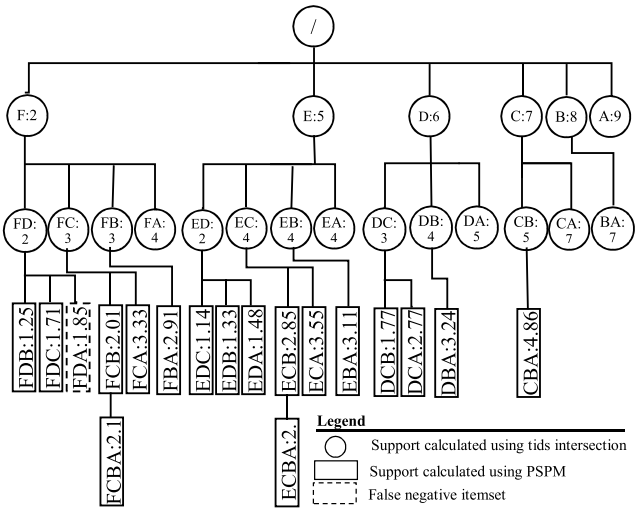

FIGURE 10. ProbBF applied to data in Table 1.

A. TIME EFFICIENCY OF ProbBF

In Fig. 11, the total runtime of seven algorithms—HashECLAT, CL-MAX, CFSP growth, PrePost+, LCM, NegFIN, and RSP-PFI—has been compared against the runtime of the ProbBF algorithm. Results for all six dense datasets—Chess, Connect, Accidents, Pumsb, Kddcup99, and Pumsb_star—have been calculated. As stated earlier, the set intersection operation is the most processing-intensive operation in frequent itemset mining. Since ProbBF doesn't require any set intersection operation after the second pass, this fact makes it the most efficient algorithm on all selected benchmark datasets compared to all seven state-of-the-art algorithms.

LCM performs better at high support threshold values, but it loses to the latest algorithms, such as NegFIN, at low support values. The reason for this characteristic of LCM is the database reduction technique that loses its effectiveness at low support threshold values. Moreover, no algorithm outperforms all others on every dataset. For example, LCM outperforms all other algorithms on the Pumsb and Kddcup99 datasets, NegFIN on the Connect dataset, and CFSP-growth on the Accidents dataset. Although Hash-Eclat and RSP-PFI are approximate algorithms, they are not efficient on dense datasets. HashEclat internally uses the Eclat algorithm, and RSP-PFI internally uses the Apriori algorithm. They were originally designed for sparse datasets. Though the performance of CL-MAX is good on dense datasets, it internally uses the MAFIA algorithm and generates maximal frequent itemsets (MFIs) only. Even though CL-MAX is an MFI algorithm, it performs slower than exact algorithms in some situations.

B. SPACE EFFICIENCY OF ProbBF

Since ProbBF does not require any set-intersection operation beyond the second pass, it frees all memory occupied by transactional data. This means that at the end of the sec-

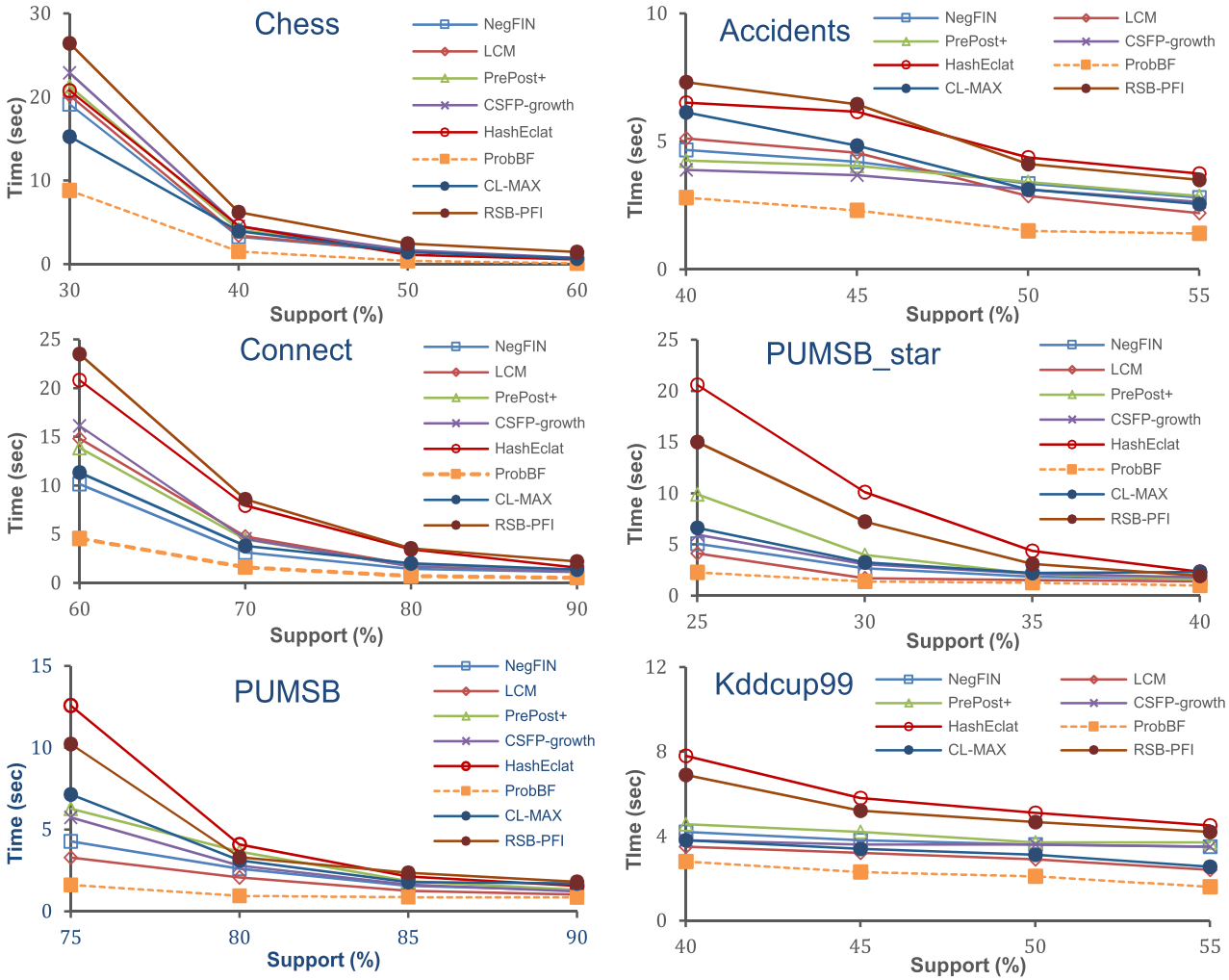

FIGURE 11. Runtime efficiency of ProbBF vs. state-of-the-art algorithms.

ond pass, the memory of ProbBF comprises only frequent itemsets. This fact renders ProbBF more memory-efficient than all other state-of-the-art algorithms, as shown in Fig. 12. In this figure, we can observe that CFSP-growth consumes more memory than FP-growth, even though the CFSP tree takes less time than the FP-tree to mine frequent itemsets. This figure also shows that CL-MAX is a memory-efficient algorithm, consuming even less memory than ProbBF on Connect and Pumsb datasets. However, it's important to note that CL-MAX is specifically a maximal frequent itemset mining algorithm. It is also worth noting that, for a better comparison, all algorithms in this study have been compelled to store all frequent itemsets in memory. In other words, no frequent itemset is written to disk. Additionally, ProbBF employs a space-efficient set enumeration tree for storing frequent itemsets, in contrast to the list used by other algorithms.

C. EVALUATING THE EFFECTIVENESS OF ProbBF

In this subsection, we assess the effectiveness of ProbBF in successfully generating frequent itemsets. Since ProbBF is probabilistic, it can make two types of mistakes: it may designate an itemset as frequent when it is indeed not, leading to a false positive (FP). Conversely, it could fail to generate a frequent itemset, resulting in a false negative (FN). The frequent itemsets that ProbBF correctly identifies as frequent are referred to as true positives (TP). We demonstrate that ProbBF does not miss the high-support TP frequent itemsets. Subsequently, we show that ProbBF generates high-quality results compared to other approximate algorithms.

1) EVALUATING HIGH SUPPORT FREQUENT ITEMSETS

In the support-confidence framework, it is the support of an itemset that lends it more importance than the other itemsets. In this subsection, we analyze the strength of the ProbBF algorithm to generate high-support frequent itemsets. Let the frequent itemset the ProbBF generates using PSPM be a probabilistic frequent itemset. This analysis, depicted in Fig. 13, has been conducted on all six selected benchmark datasets. Each dataset has been assessed by varying support threshold values along the x-axis. For instance, ProbBF was

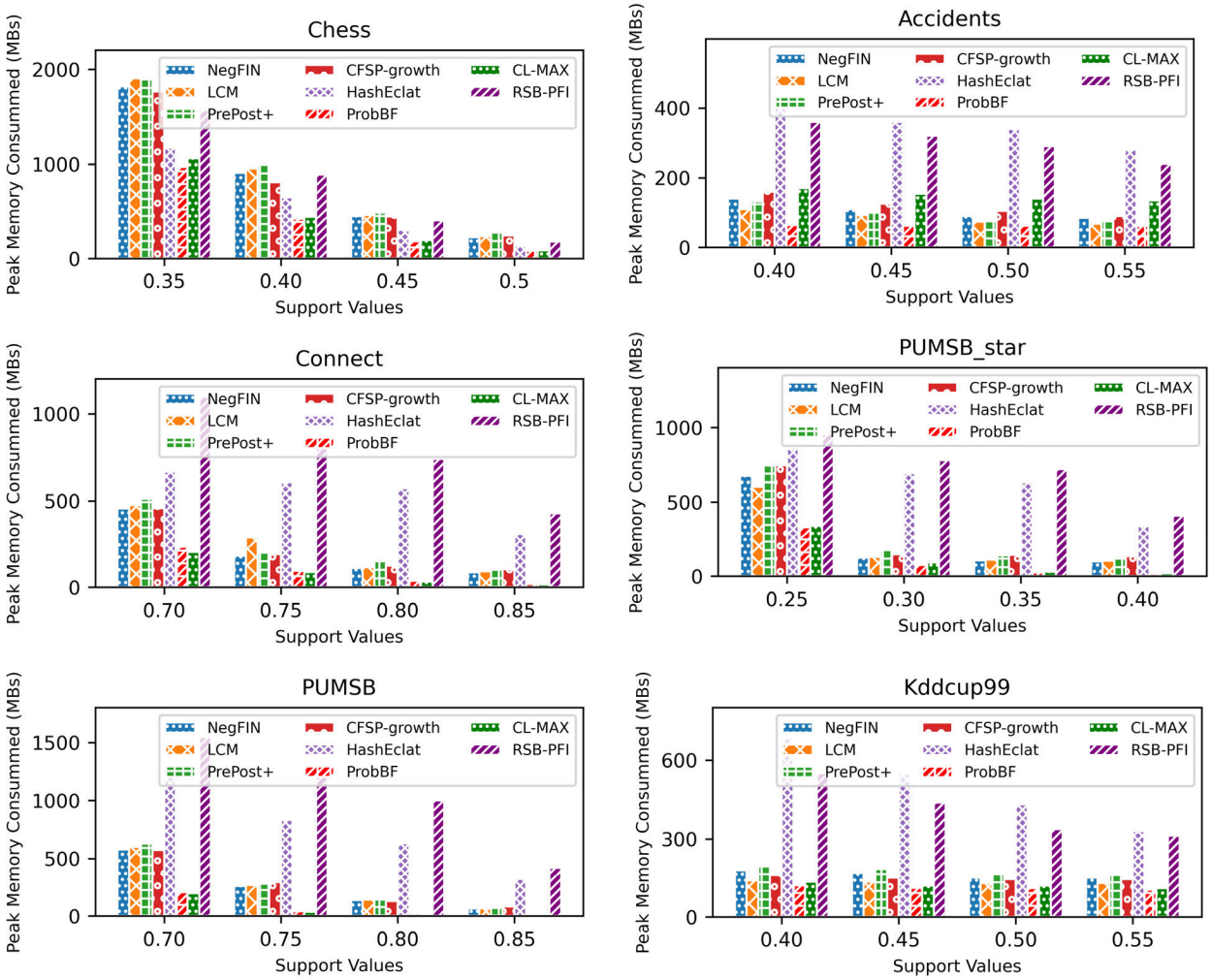

FIGURE 12. Memory efficiency of ProbBF vs. State-of-the-art Algorithms.

executed on the Connect dataset with a support threshold of 0.6. The x-axis enumerates four support threshold values, ranging from 0.9 to 0.6 with a difference of 0.1. The red bar at 0.9 represents the frequent itemsets with support ranging from 0.9 to 1, totaling 26,929. Within these frequent itemsets, the blue bar signifies the ones successfully generated by ProbBF. Notably, the blue bar leveled perfectly with the red bar, indicating that every frequent itemset within the 0.9 support threshold range has been successfully generated by ProbBF. Similarly, 506,699, 3,595,760, and 17,120,708 frequent itemsets have supports in the ranges [0.8, 0.9), [0.7, 0.8), and [0.6, 0.7), respectively. The use of semi-closed intervals, such as [0.8, 0.9), indicates that their support is less than 0.9 and equal to or greater than 0.8. All these frequent itemsets within the specified support ranges have been successfully generated by ProbBF. Therefore, the blue bars corresponding to each range are perfectly leveled with their respective red bars. It is noteworthy that not a single frequent itemset is missing from the probabilistic frequent itemsets in these ranges. Similarly, when ProbBF was run on the PUMSB dataset for a 0.6 support threshold, all frequent itemsets in the ranges [0.9, 1], [0.8, 0.9), and [0.7, 0.8) were successfully generated. This is shown by all the blue bars at par with their corresponding red bars in these mentioned ranges. Only, the frequent itemsets in the range [0.6, 0.7), which are 16,831,553, are not fully generated. Out of these 16,831,553 frequent itemsets, 16,490,248 frequent itemsets have been successfully generated by ProbBF. These 16,831,553 and 16,490,248 frequent itemsets are shown with the red bar and blue bar respectively. In the remaining 341,305 frequent itemsets, 341,291 frequent itemsets are such that they have also been generated by ProbBF, but with only one such item which is present in frequent itemsets but absent from the probabilistic frequent itemsets. These itemsets are shown with a thin green bar on top of the blue. Only 14 frequent itemsets are such that they lack two or more items. The results of the Chess dataset are very similar to that of the PUMSB. The Kddcup99 has been mined on a relatively low support threshold value of 0.4. The thick green bar on top of the blue bar, for the support threshold range of [0.4, 5), shows the

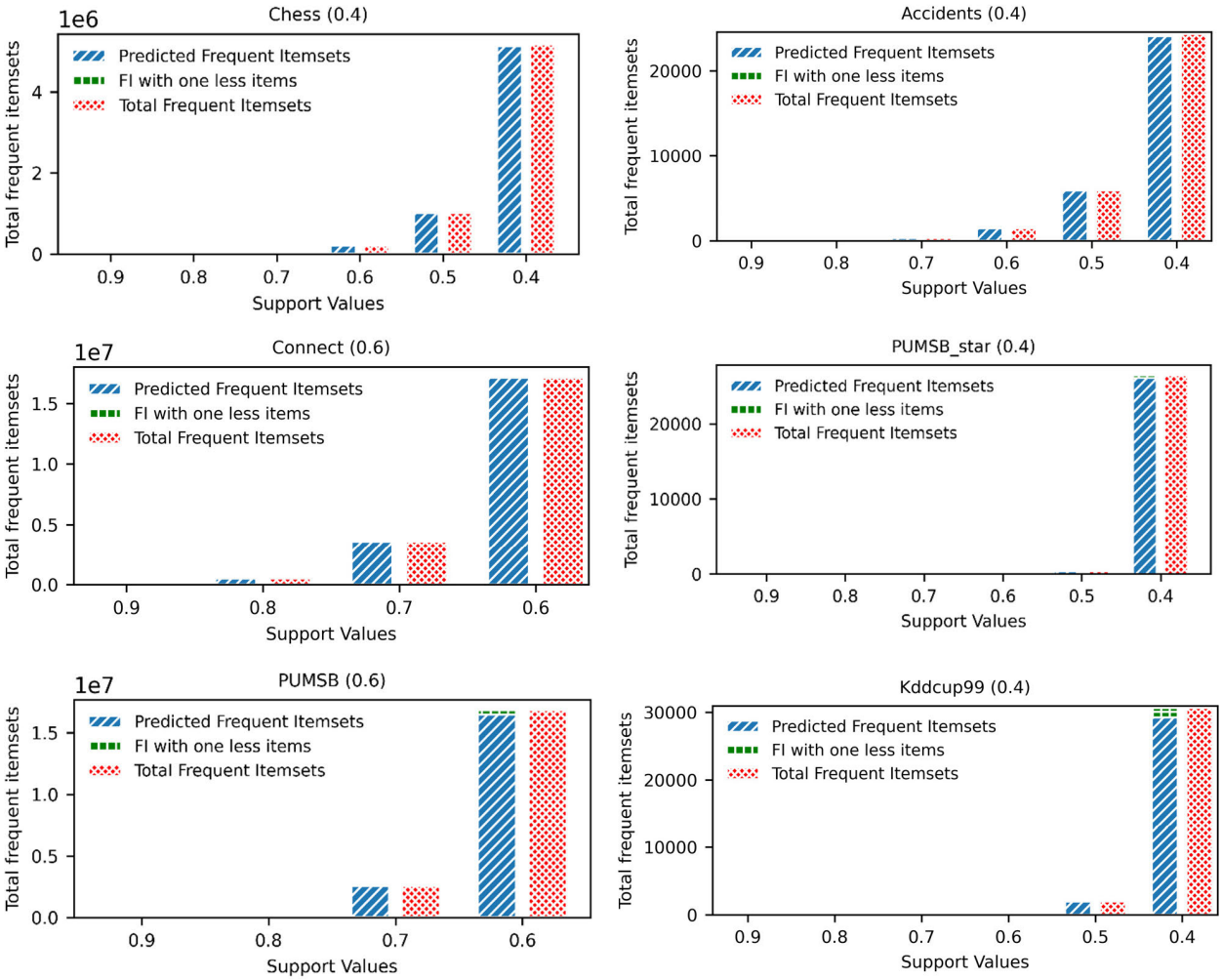

FIGURE 13. True frequent itemsets generation capability of ProbBF.

frequent itemsets that are deficient in one frequent item are relatively large in number. These itemsets are on the frequent itemset boundary, and usually deep down the set enumeration tree.

These results reinforce our observations made in Section VI-B and Section V (from the example). We observed that the more the dataset is dense and the support threshold is high, the proposed technique produces good predictions. This observation is confirmed by the results of the Connect and Kddcup99 datasets. We also observed that the frequent itemsets of high importance (i.e., frequent itemsets of high support) are successfully generated. This is confirmed by the results of all six datasets (i.e., all of the frequent itemsets in high support ranges are successfully generated on all six datasets). Moreover, false negative itemsets have probabilistic support a little short of the minimum support threshold. Similar observations could be noted for the false positive itemsets as well. These itemsets fall just below the frequent itemset boundary, and their support does not vary greatly from the support of frequent itemsets right on the boundary. With these findings, we could safely conclude that ProbBF attains high efficiency at the cost of marginal loss in quality.

2) MEASURING THE QUALITY OF THE OUTPUT OF ProbBF

To measure the effectiveness of an approximate algorithm, two widely used measures are recall and precision. Recall, also referred to as completeness, indicates the percentage of total frequent itemsets successfully generated by the algorithm. Precision measures exactness, representing the percentage of frequent itemsets generated by the approximate algorithm that are genuinely frequent.

To evaluate the effectiveness of the ProbBF, we selected three approximate algorithms from those employed in the previous sections: HashEclat, CL-MAX, and RSP-PFI. The result of this comparison has been shown in Fig. 14. These results show that, in the case of recall, ProbBF performs much better than the other three approximate algorithms. As discussed earlier in this section, the ProbBF misses very few frequent itemsets (i.e., the false negatives are very few).

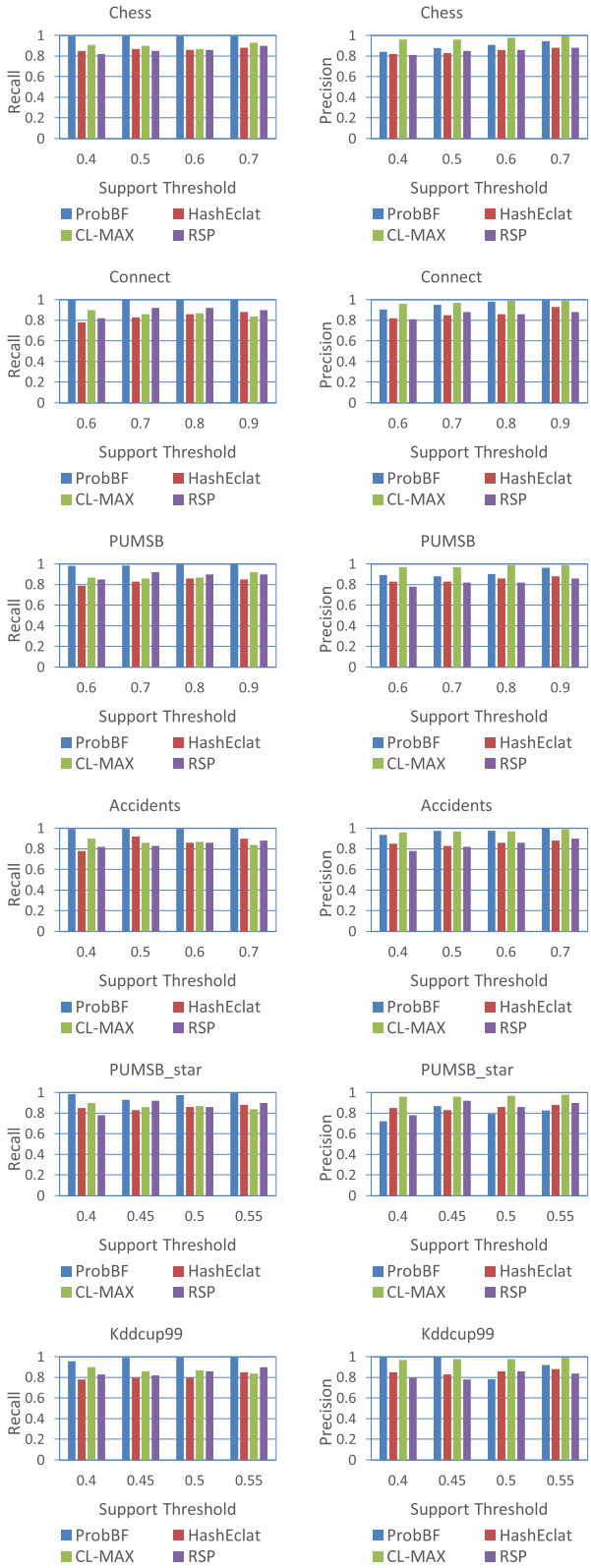

FIGURE 14. Precision and recall for the ProbBF Algorithm.

However, when it comes to precision, ProbBF is outperformed by CL-MAX when the number-of-clusters value is optimally selected – certainly a positive aspect of CL-MAX. However, the cost of this gain is a significant loss in the runtime performance of CL-MAX, as most of the work is delegated to the underlying MAFIA algorithm. Nevertheless, the performance of ProbBF is comparable to that of the other algorithms. In almost all cases, ProbBF performs better than the other two algorithms except on the Pumsb_star dataset. It is mainly because of the poor performance of PSPM on this dataset, as discussed in Section IV-B. It is worth noting that the recall value for this dataset is quite satisfactory, if not impressive. The overall performance of the ProbBF is highly encouraging, even though it uses zero data for these support approximations.

VII. CONCLUSION

In this paper, we present a novel probabilistic technique, PSPM, to predict an itemset's support based on its subsets' support. This technique is specifically designed for dense data. Authentic yardsticks were used to quantify the predictive accuracy of the PSPM, and highly encouraging findings were observed. We also propose a frequent itemset mining algorithm, ProbBF, that employs PSPM recursively to predict the support of all k − itemsets for $k \geq 3$. This algorithm stores transaction IDs for only frequent 1-itemsets and even these IDs are deleted after all the frequent 2-itemsets are determined. For itemsets of size 3 and onward, it uses PSPM to predict the support of these itemsets. The PSPM uses lightweight calculations, making the ProbBF algorithm highly efficient. The performance of ProbBF is verified by applying it to frequently used six real-world benchmark dense datasets. Results are compared against six state-of-the-art algorithms on the same datasets. These experiments confirm that ProbBF is highly efficient against all these algorithms on all six datasets.

Since ProbBF is a probabilistic algorithm, the trade-off between accuracy and efficiency is also analyzed. Although poor accuracy is observed on one of the datasets, considering the highly efficient performance over all other datasets, a slight loss in accuracy is reasonable. Our experiments suggest the use of the proposed algorithm in domains where some compromise on quality is tolerable in favor of high efficiency.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, New York, NY, USA, Jun. 1993, pp. 207–216.
[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules," *Adv. Knowl. Discovery Data Mining*, vol. 12, pp. 307–328, Feb. 1996.
[3] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May/Jun. 2000, doi: 10.1109/69.846291.
[4] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, Jan. 2004, doi: 10.1023/B:DAMI.0000005258.31418.83.
[5] K.-C. Lin, I.-E. Liao, and Z.-S. Chen, "An improved frequent pattern growth method for mining association rules," *Exp. Syst. Appl.*, vol. 38, pp. 5154–5161, May 2011, doi: 10.1016/j.eswa.2010.10.047.
[6] O. Jamsheela and G. Raju, "An improved frequent pattern tree: The child structured frequent pattern tree CSFP-tree," *Pattern Anal. Appl.*, vol. 26, no. 2, pp. 437–454, May 2023, doi: 10.1007/s10044-022-01111-1.

[7] Z. Halim, O. Ali, and M. G. Khan, "On the efficient representation of datasets as graphs to mine maximal frequent itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1674–1691, Apr. 2021, doi: 10.1109/TKDE.2019.2945573.

[8] Y. Xun, J. Zhang, H. Yang, and X. Qin, "HBPFP-DC: A parallel frequent itemset mining using spark," *Parallel Comput.*, vol. 101, Apr. 2021, Art. no. 102738, doi: 10.1016/j.parco.2020.102738.

[9] S. Lessanibahri, L. Gastaldi, and C. G. Fernández, "A novel pruning algorithm for mining long and maximum length frequent itemsets," *Exp. Syst. Appl.*, vol. 142, Mar. 2020, Art. no. 113004, doi: 10.1016/j.eswa.2019.113004.

[10] A. Bai, M. Dhabu, V. Jagtap, and P. S. Deshpande, "An efficient approach based on selective partitioning for maximal frequent itemsets mining," *Sadhana*, vol. 44, p. 183, Jul. 2019, doi: 10.1007/s12046-019-1158-1.

[11] M. Ghosh, A. Roy, P. Sil, and K. C. Mondal, "Frequent itemset mining using FP-tree: A CLA-based approach and its extended application in biodiversity data," *Innov. Syst. Softw. Eng.*, vol. 19, no. 3, pp. 283–301, Sep. 2023, doi: 10.1007/s11334-022-00500-3.

[12] J. Lu, W. Xu, K. Zhou, and Z. Guo, "Frequent itemset mining algorithm based on linear table," *J. Database Manag.*, vol. 34, no. 1, pp. 1–21, Jan. 2023, doi: 10.4018/jdm.318450.

[13] M. Ledmi, S. Zidat, and A. Hamdi-Cherif, "GrAFCI+ a fast generator-based algorithm for mining frequent closed itemsets," *Knowl. Inf. Syst.*, vol. 63, no. 7, pp. 1873–1908, Jul. 2021.

[14] C. Zhang, P. Tian, X. Zhang, Z. L. Jiang, L. Yao, and X. Wang, "Fast eclat algorithms based on minwise hashing for large scale transactions," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3948–3961, Apr. 2019, doi: 10.1109/JIOT.2018.2885851.

[15] J. M. Luna, P. Fournier-Viger, and S. Ventura, "Frequent itemset mining: A 25 years review," *WIREs Data Mining Knowl. Discovery*, vol. 9, no. 6, p. e1329, Nov. 2019, doi: 10.1002/widm.1329.

[16] T. Valiullin, Z. Huang, C. Wei, J. Yin, D. Wu, and I. Egorova, "A new approximate method for mining frequent itemsets from big data," *Comput. Sci. Inf. Syst.*, vol. 18, no. 3, pp. 641–656, 2021, doi: 10.2298/csis200124015v.

[17] S. Kumar and K. K. Mohbey, "A review on big data based parallel and distributed approaches of pattern mining," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 5, pp. 1639–1662, May 2022, doi: 10.1016/j.jksuci.2019.09.006.

[18] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, "A survey of parallel sequential pattern mining," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 3, pp. 1–34, Jun. 2019, doi: 10.1145/3314107.

[19] M. Shaikh, I. A. Tunio, S. M. S. Shah, F. K. Sohu, A. Aziz, and A. Ali, "Distributed incremental approximate frequent itemset mining using MapReduce," *Int. J. Comput. Sci. Netw. Secur.*, vol. 23, pp. 207–211, Jun. 2023, doi: 10.22937/IJCSNS.2023.23.5.22.

[20] H. Li, Y. Zhang, N. Zhang, and H. Jia, "A heuristic rule based approximate frequent itemset mining algorithm," *Proc. Comput. Sci.*, vol. 91, pp. 324–333, Jan. 2016.

[21] C. Zhang, P. Tian, X. Zhang, Q. Liao, Z. L. Jiang, and X. Wang, "HashEclat: An efficient frequent itemset algorithm," *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 11, pp. 3003–3016, Nov. 2019, doi: 10.1007/s13042-018-00918-x.

[22] S. Abbasi and A. Moieni, "BloomEclat: Efficient Eclat algorithm based on Bloom filter," *J. Algorithms Comput.*, vol. 53, pp. 197–208, Jun. 2021, doi: 10.22059/jac.2021.81890.

[23] S. M. Fatemi, S. M. Hosseini, A. Kamandi, and M. Shabankhah, "CL-MAX: A clustering-based approximation algorithm for mining maximal frequent itemsets," *Int. J. Mach. Learn. Cybern.*, vol. 12, no. 2, pp. 365–383, Feb. 2021, doi: 10.1007/s13042-020-01177-5.

[24] W. Song, W. Ye, and P. Fournier-Viger, "Mining sequential patterns with flexible constraints from MOOC data," *Int. J. Speech Technol.*, vol. 52, no. 14, pp. 16458–16474, Nov. 2022, doi: 10.1007/s10489-021-03122-7.

[25] H. Tang, L. Wang, Y. Liu, and J. Qian, "Discovering significant sequential patterns in data stream by an efficient two-phase procedure," *Math. Problems Eng.*, vol. 2022, pp. 1–23, Dec. 2022, doi: 10.1155/2022/5379086.

[26] W. Xiao and J. Hu, "SWEclat: A frequent itemset mining algorithm over streaming data using spark streaming," *J. Supercomput.*, vol. 76, no. 10, pp. 7619–7634, Oct. 2020, doi: 10.1007/s11227-020-03190-5.

[27] X. Lu, S. Jin, X. Wang, J. Yuan, K. Fu, and K. Yang, "A mining frequent itemsets algorithm in stream data based on sliding time decay window," in *Proc. 3rd Int. Conf. Artif. Intell. Pattern Recognit.*, 2020, pp. 18–24.

[28] L. B. Q. Nguyen, I. Zelinka, V. Snasel, L. T. T. Nguyen, and B. Vo, "Subgraph mining in a large graph: A review," *WIREs Data Mining Knowl. Discovery*, vol. 12, no. 4, p. e1454, Jul. 2022, doi: 10.1002/widm.1454.

[29] G. Preti, G. D. F. Morales, and M. Riondato, "MaNIACS: Approximate mining of frequent subgraph patterns through sampling," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 3, pp. 1–29, Apr. 2023, doi: 10.1145/3587254.

[30] M.-Y. Lin, C.-T. Fu, and S.-C. Hsueh, "Interactive mining of probabilistic frequent patterns in uncertain databases," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 30, no. 2, pp. 263–283, Apr. 2022, doi: 10.1142/s0218488522500118.

[31] S. Bashir and D. T. C. Lai, "Mining approximate frequent itemsets using pattern growth approach," *Inf. Technol. Control*, vol. 50, no. 4, pp. 627–644, Dec. 2021, doi: 10.5755/j01.itc.50.4.29060.

[32] R. Kumar and K. Singh, "High utility itemsets mining from transactional databases: A survey," *Int. J. Speech Technol.*, vol. 53, no. 22, pp. 27655–27703, Nov. 2023, doi: 10.1007/s10489-023-04853-5.

[33] N. M. Hung, T. Nt, and B. Vo, "A general method for mining high-utility itemsets with correlated measures," *J. Inf. Telecommun.*, vol. 5, no. 4, pp. 536–549, Jun. 2021, doi: 10.1080/24751839.2021.1937465.

[34] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 326–335.

[35] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 10, pp. 1347–1362, Oct. 2005, doi: 10.1109/TKDE.2005.166.

[36] J.-F. Qu, B. Hang, Z. Wu, Z. Wu, Q. Gu, and B. Tang, "Efficient mining of frequent itemsets using only one dynamic prefix tree," *IEEE Access*, vol. 8, pp. 183722–183735, 2020, doi: 10.1109/ACCESS.2020.3029302.

[37] Z.-H. Deng and S.-L. Lv, "PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning," *Exp. Syst. Appl.*, vol. 42, no. 13, pp. 5424–5432, Aug. 2015, doi: 10.1016/j.eswa.2015.03.004.

[38] Z.-H. Deng, "DiffNodesets: An efficient structure for fast mining frequent itemsets," *Appl. Soft Comput.*, vol. 41, pp. 214–223, Apr. 2016, doi: 10.1016/j.asoc.2016.01.010.

[39] N. Aryabarzan, B. Minaei-Bidgoli, and M. Teshnehlab, "negFIN: An efficient algorithm for fast mining frequent itemsets," *Exp. Syst. Appl.*, vol. 105, pp. 129–143, Sep. 2018, doi: 10.1016/j.eswa.2018.03.041.

[40] Y. Djenouri, P. Fournier-Viger, A. Belhadi, and J. C.-W. Lin, "Metaheuristics for frequent and high-utility itemset mining," in *High-Utility Pattern Mining*. Springer, Jan. 2019, pp. 261–278, doi: 10.1007/978-3-030-04921-8_10.

[41] W. Fang, C. Li, Q. Zhang, X. Zhang, and J. C.-W. Lin, "An efficient biobjective evolutionary algorithm for mining frequent and high utility itemsets," *Appl. Soft Comput.*, vol. 140, Jun. 2023, Art. no. 110233, doi: 10.1016/j.asoc.2023.110233.

[42] N. S. Sukanya and P. R. J. Thangaiah, "An integrated cuckoo search-genetic algorithm for mining frequent itemsets," *J. Discrete Math. Sci. Cryptogr.*, vol. 25, no. 3, pp. 671–690, Jun. 2022, doi: 10.1080/09720529.2021.2014131.

[43] S. Bagui and P. Stanley, "Mining frequent itemsets from streaming transaction data using genetic algorithms," *J. Big Data*, vol. 7, no. 1, pp. 1–20, Dec. 2020, doi: 10.1186/s40537-020-00330-9.

[44] J. S. P. Poovan, D. A. Udupi, and N. V. S. Reddy, "A genetic algorithm coupled with tree-based pruning for mining closed association rules," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 13, no. 3, p. 2876, Jun. 2023, doi: 10.11591/ijece.v13i3.pp2876-2890.

[45] G. Li, T. Wang, Q. Chen, P. Shao, N. Xiong, and A. Vasilakos, "A survey on particle swarm optimization for association rule mining," *Electronics*, vol. 11, no. 19, p. 3044, Sep. 2022, doi: 10.3390/electronics11193044.

[46] I. Tahyudin and H. Nambo, "Improved optimization of numerical association rule mining using hybrid particle swarm optimization and Cauchy distribution," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 9, no. 2, p. 1359, Apr. 2019, doi: 10.11591/ijece.v9i2.pp1359-1373.

[47] T. Su, H. Xu, and X. Zhou, "Particle swarm optimization-based association rule mining in big data environment," *IEEE Access*, vol. 7, pp. 161008–161016, 2019, doi: 10.1109/ACCESS.2019.2951195.

[48] H. Toivonen, "Sampling large databases for association rules," in *Proc. Vldb*, 1996, pp. 134–145.

[49] C. C. Aggarwal and J. Han, *Frequent Pattern Mining*. San Francisco, CA, USA: Morgan Kaufmann, 2014.

[50] M. Riondato and E. Upfal, "Mining frequent itemsets through progressive sampling with Rademacher averages," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 1005–1014.
[51] X. Wu, W. Fan, J. Peng, K. Zhang, and Y. Yu, "Iterative sampling based frequent itemset mining for big data," *Int. J. Mach. Learn. Cybern.*, vol. 6, no. 6, pp. 875–882, Dec. 2015, doi: 10.1007/s13042-015-0345-6.
[52] Z. Zhang, W. Pedrycz, and J. Huang, "Efficient frequent itemsets mining through sampling and information granulation," *Eng. Appl. Artif. Intell.*, vol. 65, pp. 119–136, Oct. 2017, doi: 10.1016/j.engappai.2017.07.016.
[53] C. Ordonez, "Models for association rules based on clustering and correlation," *Intell. Data Anal.*, vol. 13, no. 2, pp. 337–358, Apr. 2009, doi: 10.3233/ida-2009-0369.
[54] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, "Finding interesting associations without support pruning," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 1, pp. 64–78, Feb. 2001, doi: 10.1109/69.908981.
[55] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk, "Mining database structure; Or, how to build a data quality browser," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2002, pp. 240–251.
[56] R. Pagh, M. Stöckel, and D. P. Woodruff, "Is min-wise hashing optimal for summarizing set intersection?" in *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, Jun. 2014, pp. 109–120.
[57] D. Bera and R. Pratap, "Frequent-itemset mining using locality-sensitive hashing," in *Proc. Int. Comput. Combinatorics Conf.*, 2016, pp. 143–155.
[58] R. J. Bayardo, "Efficiently mining long patterns from databases," *ACM SIGMOD Rec.*, vol. 27, no. 2, pp. 85–93, Jun. 1998.
[59] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A maximal frequent itemset algorithm," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1490–1504, Nov. 2005, doi: 10.1109/TKDE.2005.183.
[60] K. Gouda and M. J. Zaki, "GenMax: An efficient algorithm for mining maximal frequent itemsets," *Data Mining Knowl. Discovery*, vol. 11, no. 3, pp. 223–242, Nov. 2005, doi: 10.1007/s10618-005-0002-x.
[61] R. Rymon, "Search through systematic set enumeration," in *Proc. KR*, 1992, pp. 539–550.
[62] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets," in *Proc. Fimi*, 2004, pp. 1–11.

MUHAMMAD SADEEQULLAH received the B.S. degree from the City University of Science & Technology, Peshawar, Pakistan, in 2004, and the M.S. degree in computer science from the University of Peshawar, Peshawar, Pakistan, in 2009, where he is currently pursuing the Ph.D. degree with the Department of Computer Science.

He has been teaching computer science for the last more than 15 years. His research interests include data mining, machine learning, swarm intelligence, and data science.

AZHAR RAUF received the M.Sc. degree in computer science from the Department of Computer Science, University of Peshawar, Pakistan, in 1994, and the M.S. and Ph.D. degrees in computer science from Colorado Technical University, Colorado Springs, CO, USA, in 2002 and 2006, respectively. He has more than 24 years of experience which includes 17 years of teaching and research and seven years of IT industry experience. He has worked for different companies, including Bearing Point, Greatwest Healthcare, Oracle Corporation, Actiontech Electronics, Colorado, CO, USA, and Wilfrid Laurier University, Waterloo, ON, Canada. He is currently a Professor with the Department of Computer Science, University of Peshawar. He is the author or coauthor of more than 50 research publications. He has supervised several Ph.D. and M.S. students. His research interests include data mining, data warehousing, databases, data privacy, and big data.

SAIF UR REHMAN received the M.S. degree in computer science with a specialization in data mining from International Islamic University Islamabad, Pakistan, the M.Sc. degree in computer science from the Computer Science Department, University of Peshawar, Peshawar, Pakistan, and the Ph.D. degree in computer science from Kohat University of Science and Technology, Kohat, Pakistan, with a specialization in data mining.

He is currently a Lecturer in computer science with the Department of Computer Science, University of Peshawar. His research interests include data mining, frequent itemset mining, and data science.

NOHA ALNAZZAWI received the B.Sc. degree from Taibah University, Yanbu, Saudi Arabia, and the M.Sc. degree in computer science and the Ph.D. degree in computer science with a specialization in clinical text mining from The University of Manchester, U.K.

She is currently an Associate Professor with the Computer Science and Engineering Department, Yanbu Industrial College. She is also holding the position of the Head of the Research and Consultation Department, Research and Industrial Development Directorate in the Education Sector in the Royal Commission—Yanbu. Her research interests include data mining, data analysis, data science, text mining, and machine learning.