Hindawi
Complexity
Volume 2022, Article ID 1942517, 20 pages
https://doi.org/10.1155/2022/1942517

Research Article

An Incremental Interesting Maximal Frequent Itemset Mining Based on FP-Growth Algorithm

Hussein A. Alsaeedi[1] **and Ahmed S. Alhegami**[2]

[1]*Department of Computer Science, University of Science and Technology, Sana'a, Yemen*
[2]*Faculty of Computers and Information Technology, University of Sana'a, Sana'a, Yemen*

Correspondence should be addressed to Hussein A. Alsaeedi; alhussein1977@gmail.com

Received 8 April 2022; Revised 3 June 2022; Accepted 23 June 2022; Published 26 July 2022

Academic Editor: Atila Bueno

Frequent itemset mining is the most important step of association rule mining. It plays a very important role in incremental data environments. The massive volume of data creates an imminent need to design incremental algorithms for the maximal frequent itemset mining in order to handle incremental data over time. In this study, we propose an incremental maximal frequent itemset mining algorithms that integrate subjective interestingness criterion during the process of mining. The proposed framework is designed to deal with incremental data, which usually come at different times. It extends FP-Max algorithm, which is based on FP-Growth method by pushing interesting measures during maximal frequent itemset mining, and performs dynamic and early pruning to leave uninteresting frequent itemsets in order to avoid uninteresting rule generation. The framework was implemented and tested on public databases, and the results found are promising.

1. Introduction

Association rule mining (ARM) [1] has been widely used as a leading technique in data mining. It is usually utilized in analyzing marketing baskets. ARM commonly employs two main subtasks, namely mining of frequent itemsets to ensure a minimum support threshold and generation of association rules to satisfy a minimum confidence threshold. Most of the studies have addressed the efficiency criterion of frequent itemset mining as it normally entails more resource capacity and computing time [2].

Frequent itemsets (FIs) can be mined from transaction databases through one of the traditional algorithms that can be generally grouped into two methods [3]: Apriori-based method, which is used for generating and filtering candidate itemsets such as Apriori algorithm [4], and tree-based method that is normally used for building FP-tree and then mining FIs from the FP-tree such as FP-Growth [5], TRR [6], PrePost+ [7], FIN [8], dFIN [9], and negFIN [10] algorithms. Since Apriori-based methods depend on continuous scanning of the database to generate multiple candidate itemsets, they require high I/O. On the contrary, tree-based methods scan the database only twice, but they need higher memory for constructing multiple sub-trees [2, 11].

Generally, the representation of FIs can be either closed frequent itemsets (CFIs) or maximal frequent itemsets (MFIs). An FI is considered a CFI if it has no superset having the same support [12]. CFI algorithms include AprioriClose [13], FP-Close [14], Closet+ [15], CHARM [12], and NEclatClosed [16] algorithms. On the other hand, a given FI is described as an MFI only if any superset of the itemset is not frequent [17, 18]. MFI set is considerably smaller than that of FIs [19]. There are many competent algorithms used for mining MFIs such as MAFIA [20], GenMax [21], FP-Max [22], FP-Max* [23], Charm-MFI [24], PADS [25], SelPMiner [19], and CL-Max [26]. The FP-Max [22] and FP-Max* [23] algorithms use FP-Growth to build a tree as FP-tree and mine MFIs from the tree based on a bottom-up and divide-and-rule strategy.

In dynamic data, when adding an incremental database to the previous databases, some previous FIs become invalid and new FIs appear due to the changes in the support value of some FIs. Generally, the traditional FI mining algorithms are ineffective for incremental data as the entire database is re-mined afresh [2]. Since transaction databases are unceasingly and dramatically growing, incremental FI mining algorithms are needed as useful for making decisions and getting real-time information sought by users.

Many attempts have been made for processing incremental data without re-mining the entire updated databases such as FUP [27], FUFP-tree [28], FUFP-tree maintenance [29], FCFPIM [30], FPISC-tree [31], FIUFP-Growth [2], pre-large [32], pre-FUFP [33], and FPMSIM [11] algorithms. It is noteworthy that these algorithms were used for mining incremental FIs. IM_WMFI [17] and IMU2P-Miner [18] have been proposed to deal with incremental MFIs. The drawbacks of these algorithms are that they do not regard the factors of size and time of data entry and, therefore, require re-mining of the updated database. Since new data arrive over time, researchers have been motivated to propose techniques that update the entire model of previously discovered knowledge (PDK), instead of running the algorithms from a scratch, thus presenting a new incremental model such as [34–36], and our proposed framework [37].

For association rules, two measures have been used: objective and subjective measures. Objective measures are statistical values, such as support, confidence, all confidence, and left [38]. These measures were used during mining, such as support in the first task and confidence in the second task of ARM. On the other hand, subjective measures, such as unexpectedness [39], actionability [40], and novelty [34, 36, 37], were used to capture the user's belief about the domain. However, these subjective measures were used at post-mining.

Our proposed approach is motivated by the increasing need for an efficient MFI algorithm that deals with larger data entry over time. As an extended form of FP-Growth and FP-Max algorithms for mining incremental interesting MFIs, our method uses novelty metrics (NM) as a subjective measure during the process of the mining stage. A major contribution of this proposed framework is handling the time-changing data and user-domain knowledge. This is useful when many databases arrive at various times or from a distributed environment. Certainly, it is desirable to update the discovered frequent itemsets each time new data arrives. Moreover, the incremental nature of the proposed framework makes it valuable to mine interesting frequent items at the current time concerning the previously discovered frequent items more willingly than wholly mining all frequent items. So, dynamic pruning for these frequently discovered itemsets is performed in real time. The objective of dynamic pruning is to save time and reduce search complexity.

This study introduces an algorithm, based on a tree structure, for mining interesting MFIs. The major contributions of our work are as follows: (1) extending the FP-Max algorithm for incremental MFI mining; (2) integrating subjective interestingness criterion (novelty measure) during the process of mining for reducing the count of discovered interesting MFIs and subsequently reducing search complexity; and (3) introducing a structure that handles all the items (frequent or infrequent) with related information along with previous discovered IMFIs for use next time to speed up the construction and size of the tree.

This study is structured as follows. Section 2 reviews the related work. Section 3 introduces the design issues of our approach. Section 4 discusses the experimental settings and results. Section 5 concludes the study.

2. Related Work

The concept of association rule mining was introduced by Agrawal et al. in 1993 [1], and the Apriori algorithm was proposed a year later [4] for mining FIs and generating association rules. The algorithm was used to generate and filter candidate itemsets in a level-by-level manner. However, a disadvantage of this method is that it generates several candidate itemsets, which need multiple database scans, thus consuming much time and high I/O. FP-Growth algorithm [5] was presented, using compact data structure as FP-tree to compact all transactions of the database inside the tree. This algorithm scans a database twice only, firstly for finding support for each item and secondly for building the FP-tree. Then, the algorithm recursively builds sub-trees to mine all FIs. A limitation of this algorithm, however, is its need to create multiple sub-trees to mine FIs, which, in turn, require considerable resources and processing time.

MFI mining concept was later introduced as in MAFIA [20], FP-Max [22], FP-Max* [23], PADS [25], SelPMiner [19], and CL-Max [26]. MAFIA is an MFI method, which uses a bitmap representation to check itemsets' support information without any database scan. A major disadvantage of this algorithm is that it is inefficient with respect to sparse databases [20]. FP-Max was then proposed to mine all MFIs, and it used FP-tree structure and MFI-tree structure [22]. FP-Max* is an FP-tree-based algorithm used for mining MFIs utilizing its own two-dimensional array structure, called FP array to improve mining performance by reducing the number of tree scans [23]. A drawback of these algorithms, however, is that in a dense database, the FP-trees become more compact and thus more memory usage and slower execution time. PADS is another FP-tree-based method that defines and uses a pattern-aware dynamic search order to pre-prune unnecessary operations [25]. It is praised for guaranteeing higher speed compared with the previous approaches. However, it is memory-consuming due to its storing conditional databases and time-consuming due to longer search space [25]. CL-Max is an algorithm, which uses k-means concept for MFI mining [26]. SelPMiner was introduced to utilize the optimizations of the search space pruning through itemset-count tree format [19].

Remarkably, all the abovementioned algorithms have to do with static data. Consequently, new methods have been introduced to work on incremental data without re-mining the entire updated databases such as FUP [27], FUFP-tree [28], FUFP-tree maintenance [29], pre-large [32], and pre-FUFP [33] algorithms.

Based on the concept of Apriori method, Cheung et al. introduced the FUP algorithm [27] to handle the new database and update FIs efficiently. This algorithm reduces the scans of the database. Based on previously found frequent or infrequent itemsets of the previous databases, the algorithm partitions discovered itemsets from the incremental database into four cases: Case 1: frequent in previous and incremental databases, so frequent in the updated database; Case 2: frequent in the previous database but infrequent in the incremental database, so frequent or infrequent in the updated database; Case 3: infrequent in the previous database but frequent in the incremental database, so frequent or infrequent in the updated database. Only if infrequent, rescanning of previous databases is needed; and Case 4: infrequent in previous and incremental databases, so infrequent in the updated database.

However, many researchers preferred to use FP-Growth-based algorithms for better management of frequent itemset search in a dynamic database such as FUFP-tree algorithm, which was introduced as a developed algorithm to effectively deal with new transactions and improve the efficiency of the updated FP-tree structure, by reducing the number of rescans of the previous databases after adding an incremental database [28]. This FUFP-tree was characterized by double action of node insertion and deletion from the tree. The FUFP-tree algorithm [28] handles the itemsets based on the four principles of the FUP algorithm [27]: Case 1: frequent in previous and incremental databases, so support item is updated in header table and FUFP-tree; Case 2: frequent in the previous database but infrequent in incremental database, maybe frequent or infrequent. If frequent, support item is updated in header table and FUFP-tree, else the item from header table and all nodes from FUFP-tree are deleted; Case 3: infrequent in previous databases but frequent in incremental databases, maybe frequent or infrequent. If frequent, the item is placed at the end of the header table and its nodes to the leaf node of a path in the FUFP-tree are added, else nothing is done; and Case 4: infrequent in previous and incremental databases, so nothing is done. Like the FUP algorithm, the FUFP-tree maintenance algorithm improves the FUFP-tree structure after the addition of a new database [29]. An improvement was made to FUFP-tree structure so that when deleting transactions from the databases, the mining performance of incremental association rules was efficiently improved and the execution time was reduced. After updating tree, the algorithm continued to mine all FIs from the updated tree [29]. Pre-FUFP maintenance algorithm was proposed as a modified FUFP-tree algorithm. This algorithm was based on the "pre-large" concept that identified upper and lower support thresholds [33]. Accordingly, the previous databases need not be rescanned if the incremental transactions count is less than the safety number f of new transactions. The f number can be obtained according to the following equation:

$$f = \frac{(S_u - S_l)d}{1 - S_u}, \tag{1}$$

where S_u is the upper support threshold, S_l is the lower support threshold, and d is the count of transactions in previous databases [33]. The problem with these algorithms is that they were based on a modified FP-tree structure, working on rescanning the whole database and updating and deleting items in the tree for mining incremental FIs from the updated tree [2]. Other algorithms used upper bound as an improved method for the traditional Phi correlation for mining item pairs from static databases based on the Apriori method such as the one proposed by Li et al. [41].

Other algorithms used list structure on a dynamic database to mine erasable patterns and high-utility patterns such as IWEL [42], LINE [43], IMSEM [44], VME [45], PRE-HAUIMI [46], and HUI-list-INS [47] algorithms. Some other algorithms were based on the Apriori or tree method [48]. The point of convergence between these studies and this study is that they are incremental, working on frequent pattern mining, using some additional measures of interestingness. However, these studies were concerned with erasable and high-utility patterns using list structure with weighted or pre-large concepts. The main purpose of this study was to develop an FP-Max tree-based algorithm by utilizing subjective and objective measures for mining MFIs from a dynamic database. List structure can be more effective for mining, especially high-utility and erasable patterns, with regard to quantities or profit value conditions and other criteria, which are beyond our research.

There was a growing need for more comprehensive methods that can operate on both maximal and incremental frequent itemsets, and several incremental MFI algorithms were proposed [17, 18, 49]. IM_WMFI algorithm [17] used a tree structure to mine WMFIs from incremental databases using weighted criteria of representative patterns and item importance. It scans the entire incremental database only once and extracts fewer number of MFIs. IMU2P-Miner algorithm [18] was introduced for mining MFIs from univariate uncertain dada. This algorithm used a tree structure local array to keep the updates without the need for tree reconstruction as it allows only one path to be updated or added.

Interestingness is another feature that has received little attention in the field of incremental MFI mining. Briefly, two interestingness measures are considered: objective and subjective measures. The most considered objective measures that reflect the statistical strength of a pattern are support, confidence, all confidence, and left [38]. They are significantly used to discover only strong rules and, hence, filter out the number of uninteresting patterns. Since these measures failed to reflect user's knowledge, subjective measures were called to ensure the reduction in the number of the discovered interesting rules only [36]. Novelty measure (NM), as a subjective measure, was presented in [34–36] according to the following equation:

$$NM = \frac{\left(\left(\left|S_1\right| + \left|S_2\right|\right) - 2 * K\right) + \sum_{i-1}^{k} \delta\left(C_1^i, C_2^i\right)}{\left|S_1\right| + \left|S_2\right|}, \tag{2}$$

where S_1 and S_2 are two conjunct sets with cardinalities $|S_1|$ and $|S_2|$, respectively. K is the pairs of compatible conjuncts between S_1 and S_2. $\delta(C_1^i.C_2^i)$ is the ith pair of compatible conjuncts. Interested readers may refer to [34–37].

To the best of our knowledge, SelPMiner and CL-Max are the recent state-of-the art algorithms used for mining

Table 1: Characteristics of the major algorithms reviewed.

Algorithm	Year	Tree-based	Incremental	Type of FIs	Use subj. measure	No. ref.
Apriori	1994	No	No	FIs	No	[4]
FP-Growth	2000	Yes	No	FIs	No	[5]
FP-Max∗	2005	Yes	No	MFIs	No	[23]
PADS	2009	Yes	No	MFIs	No	[25]
SelPMiner	2019	Yes	No	MFIs	No	[19]
CL-Max	2021	No	No	MFIs	No	[26]
FUP	1996	No	Yes	FIs	No	[27]
FUFP-tree	2008	Yes	Yes	FIs	No	[28]
FUFP-tree maintenance	2009	Yes	Yes	FIs	No	[29]
Pre-large	2001	No	Yes	FIs	No	[32]
Pre-FUFP	2009	Yes	Yes	FIs	No	[33]
IM_WMFI	2016	Yes	Yes	MFIs	No	[17]
IMU2P-Miner	2018	Yes	Yes	MFIs	No	[18]
Our proposed	—	Yes	Yes	MFIs	Yes	—

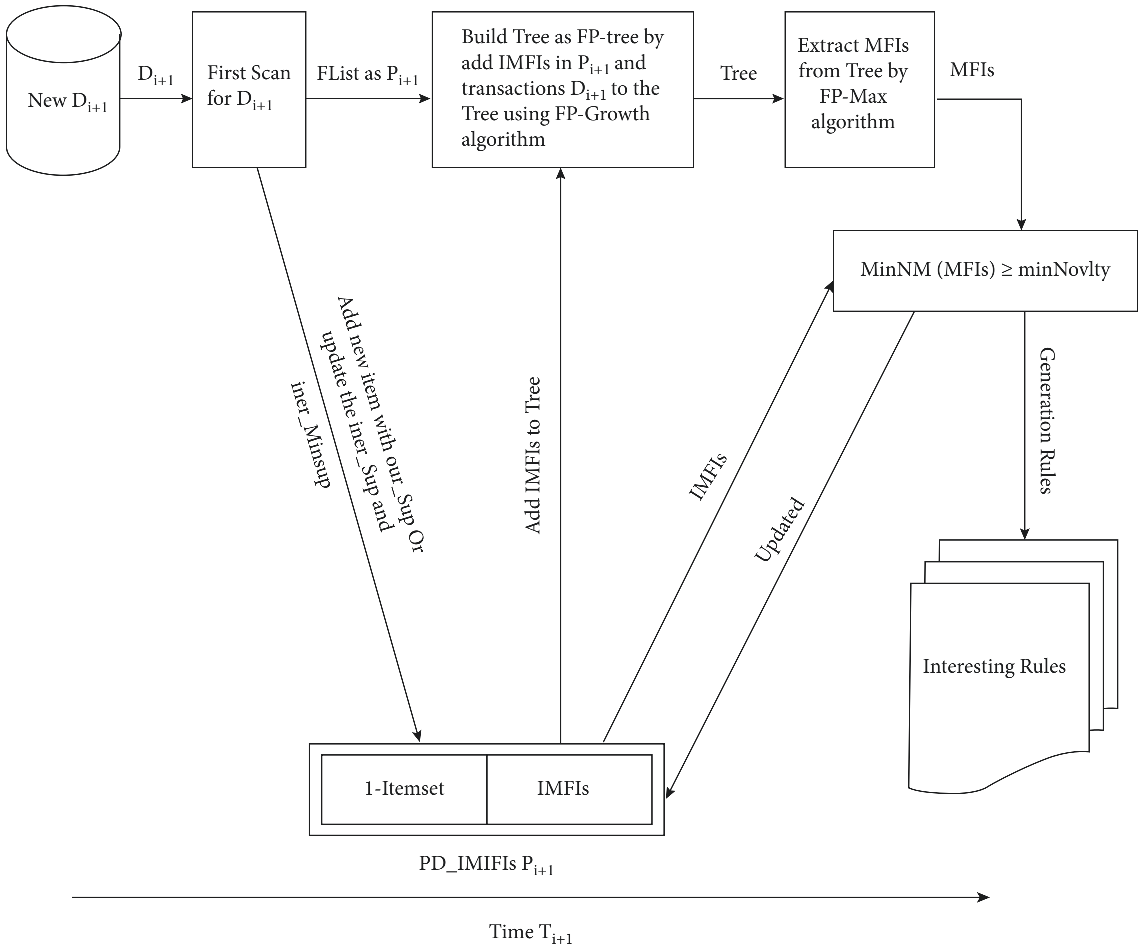

New D_{i+1}
D_{i+1}
First Scan for D_{i+1}
FList as P_{i+1}
Build Tree as FP-tree by add IMFIs in P_{i+1} and transactions D_{i+1} to the Tree using FP-Growth algorithm
Tree
Extract MFIs from Tree by FP-Max algorithm
MFIs
MinNM (MFIs) ≥ minNovlty
Add new item with our_Sup Or update the iner_Sup and iner_Minsup
Add IMFIs to Tree
IMFIs
Updated
Generation Rules
Interesting Rules
1-Itemset
IMFIs
PD_IMIFIs P_{i+1}
Time T_{i+1}

Figure 1: General architecture of the proposed framework.

MFIs from static databases. IM_WMFI and IMU2P-Miner algorithms are the only recent algorithms that use the tree method for mining MFIs from incremental database. These algorithms used additional objective measures in mining process.

In our proposed approach, an integrated algorithm was used. The basic FP-Growth algorithm was adopted to mine FIs from the tree as FP-tree without adjusting the tree structure, and FP-Max algorithm was used for mining all MFIs based on MFI-tree structure. Therefore, there is no need to rescan the previous databases and reconstruct the FP-tree. Added to the incremental nature of our method, NM is used to ensure the reduction in the number of MFIs and, consequently, the number of the discovered interesting rules. Table 1 shows the characteristics of the major reviewed algorithms.

Table 2: Notation and meaning.

Notation	Meaning
T_{i+1}	The new time (current time)
D_{i+1}	Current database at T_{i+1}
n	The total count of transactions in D_{i+1}
D^u	Di + 1 and all previous databases
Min_Sup	Minimum support threshold, set by user where ≥0 and ≤1
cur_Sup	The count of transactions that contains item in D_{i+1}
cur_Minsup	The value integer of Min_Sup in D_{i+1}
incr_Sup	The count of transactions that contains item in D_U
incr_Minsup	The sum of value integer minSup in DU
1-Itemset	1-Itemset containing name item (key) with cur_Sup, incr_Sup, and incr_Minsup
MFIsnew	A new maximal frequent itemsets
NM	Value of novelty measure
minNM	The least value within the NM values
minNovlty	Minimum novelty threshold, set by user where ≥0 and ≤1
IMFIs	A new interesting maximal frequent itemsets
PD_IMFIs	Previous discovered interesting MFIs
P_{i+1}	PD_IMFIs at Ti + 1 containing 1-Items and IMFI list
S_1	Any IMFIs within IMFI list
S_2	A new MFIs
$\|S_1\|$	Length (size) of S_1
$\|S_2\|$	Length (size) of S_2
K	Count of similar itemsets of S_1 and S_2
Conf	The value of confidence for any rule
minConf	Minimum confidence threshold, set by user where ≥0 and ≤1

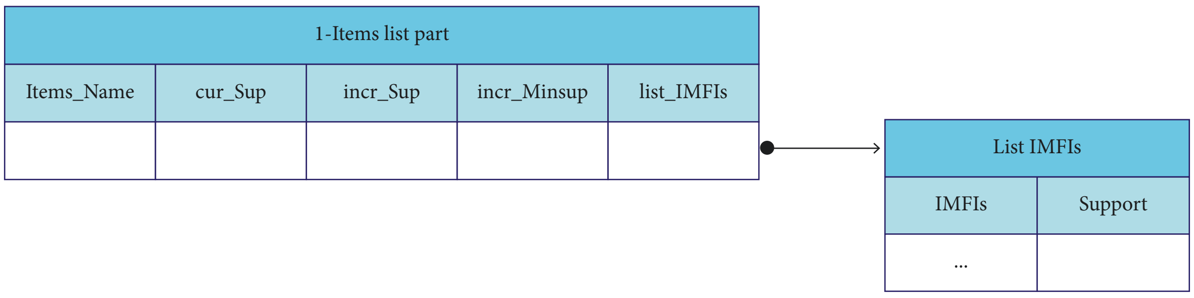


Figure 2: PD_IMFI structure.

3. The Proposed Approach

The proposed approach is designed to discover incremental IMFIs, named as IIMFIs, from dynamic database, as shown in Figure 1. The proposed approach contains several components to which three functions are added: (1) keeping all items, frequent or infrequent, with their related information; (2) constructing a tree from the discovered IMFIs and current database; and (3) dynamic pruning of uninteresting MFIs. Symbols and notation used in this study are shown in Table 2.

The approach acts incrementally firstly by adding any new item to the item list and updating the support and threshold support of each item and, secondly, by adding IMFIs to the list of IMFIs in PD_IMFIs. PD_IMFI structure will be explained in Section 3.1. The framework architecture contains five phases: the first phase scans all transactions in D_{i+1} to add any new item as 1-Itemset with its cur_Sup or update the incr_Sup and incr_Minsup for each item in P_{i+1}. The second phase builds up the tree from previous IMFIs and transactions in D_{i+1}, utilizing FP-Growth algorithm similar to FP-tree structure. As for IMFIs (with its corresponding support), only the associated item with incr_Sup ≥ incr_Minsup or items with cur_Sup ≥ cur_Minsup are added. For each transaction in D_{i+1}, only items in transaction are added to the tree where the item incr_Sup ≥ incr_Minsup, or cur_Sup ≥ cur_Minsup. In the third phase, FP-Max algorithm is used to extract MFIs from the tree using MFI-tree structure. In the fourth phase, dynamic pruning is performed using NM to compare each new MFI with all IMFIs of IMFI list in P_{i+1}. A new MFIs can be added to the list of IMFIs only if it is interesting; otherwise, it is discarded. In the fifth phase, association rules are generated from IMFIs. The output of this framework is incremental IMFIs and interesting rules. The detailed description of the proposed algorithm will be given in Section 3.5.

3.1. PD-IMFI Structure

3.1. PD-IMFI Structure. The objective of PD_IMFIs is to speed up the construction of the tree by adding the previous IMFIs to tree nodes with a counter equal to the support value of IMFIs instead of rescanning previous transactions, thus reducing the size of the constructed tree, through keeping all the items (frequent or infrequent) with related information such as updated support, incremental threshold support, and the list of IMFI, for use next time, and so on.

As shown in Figure 2, PD_IMFI structure consists of two main parts: 1-Items list and IMFI list. 1-Items list part contains four fields: item_Name, cur_Sup, incr_Sup, and incr_Minsup. item_Name is a key and identifier for each item in PD_IMFIs. cur_Sup is the count of transactions, which contains the item in D_{i+1}. incr_Sup is the sum support of each item in D^U. incr_Minsup is the sum minimum threshold support of each item in D^U. It is worth noting that cur_Minsup is a temporal condition in D_{i+1} used to update incr_Minsup. cur_Minsup is calculated according to the following equation [50]:

$$cur\_\text{Min sup} = \text{min\_Sup} * n. \tag{3}$$

As an example, let the count of transactions in $D_{i+1} = 10$, and Min_Sup = 0.5, so cur_Minsup = 0.5 ∗ 10 = 5.

Each item in 1-Items list part has an IMFI list, which may be null or have one/more IMFIs. IMFI list part contains two fields: IMFI, referring to an array of associated itemsets as IMFIs, and support, indicating the frequency value of IMFIs in D^U.

3.2. Incremental MFI Mining. The incremental nature of the proposed approach self-adjusts the minimum support (incr_Minsup) due to the change in the support value (incr_Sup) of each item in 1-Items list as a result of the Di + 1 scan in the first phase, algorithms 1-2. The construction of the tree is based on previously discovered IMFIs in Pi (IMFI list) and transactions in Di + 1, tree starting is null, and fetching is only of IMIFs in Pi that associated item in Pi + 1, if cur_Sup ≥ cur_Minsup, or incr_Sup ≥ incr_Minsup. In the second scan of Di + 1, for each transaction, any item is removed, if item in 1-Items list of Pi + 1 has the value of cur_Sup < cur_Minsup and incr_Sup < incr_Minsup, and items are re-sorted in descending order based on the value of incr_Sup of an item in 1-Items list of Pi + 1. Support in the header table and the counter node can be updated only if it is less than incr_Sup of each item in the 1-Items list part of Pi + 1, as in the second phase, Algorithm 3. As a result, the counter values of nodes are updated. In the third phase, incremental MFIs with updated support are extracted from this tree. Consequently, the Conf value may change as it is related to support.

3.3. Incremental Dynamic Pruning of MFIs. One of the main advantages of our approach is its ability to handle dynamic pruning based on NM. The goal is to reduce the count of the MFIs. The framework computes NM according to equation (2). In fact, we do not need to calculate $\delta(C_1^i, C_2^i)$ in case of association rules, because it is always equal to zero, and it is used in the case of classification. So, equation (2) can be modified as follows:

$$NM = \frac{\left(|S_1| + |S_2| - 2 * K\right)}{|S_1| + |S_2|}. \tag{4}$$

NM value of new MFIs determines whether it is novel or not. The NM value of these new MFIs is calculated against all the IMFIs in the IMFI list part of PD_IMFIs as follows:

$$NM_{MFIs\text{new}}^{IMFIs\text{list in}P_{i+1}} = \frac{\left(|S_1| + |S_2| - 2 * K\right)}{|S_1| + |S_2|}. \tag{5}$$

As a result of using equation (5) that represents the relationship between S_1 and S_2, four cases are observed. Case 1: S_1 is equal to S_2, $S_1 = S_2$ and $K = |S_1| = |S_2|$. Case 2: S_2 is a subset of S_1, $S_2 \in S_1$, $K > 0$, and $K = |S_2|$. Case 3: S_1 is a subset of S_2, $S_1 \in S_2$, $K > 0$, and $K = |S_1|$. Case 4: S_1 is not equal to S_2, $S_1 \neq S_2$, $K \geq 0$, $K < |S_1|$, and $K < |S_2|$. Cases 1 and 2 are not used in our approach since FP-Max algorithm cancels out any recurrence of FIs in the MFI-tree. Case 3 is not applicable to our approach since the tree-building process removes any IMFIs from IMFI list part when any cur_Sup ≥ cur_Minsup, or incr_Sup ≥ incr_Minsup of items associated with IMFIs in Pi + 1. Only Case 4 is utilized in our approach to calculate the value of NM depending on the value of K, as in the fourth phase, Algorithm 4. After the computation of the NM of each IMFI in Pi + 1, the dynamic pruning is performed by eliminating those new MFIs with NM less than minNovlty as shown in Algorithm 1. For example, let minNovlty = 0.5, a new MFIs = {a,s}, and the three IMFIs in IMFI list, i.e., {{r,m,s}, {r,b,m}, {m,t}}. The NM between a new MFIs is S_2 = {a, s} and $|S_2| = 2$ and each IMFI can be calculated and illustrated as in Table 3. Note here that minNM = 0.6. So, minNM > minNovlty, and then, the new MFIs {a, s} are interesting; subsequently, it is added to IMFI list. Suppose later comes a new MFIs = {u,s,m}. The NM between the new MFIs is S_2{u,s,m} and $|S_2| = 3$ and each IMFI in IMFI list can be calculated as shown in Table 4. Note here that minNM = 0.2. So, minNM < minNovlty, and then, the new MFIs {u,s,m} are uninteresting; subsequently, it is pruned.

3.4. Generation of Interesting Rules. NM is used in this approach as a determining measure during the pruning process to discover MFIs that are only interesting. As such, NM becomes a crucial constraint within the algorithm to reduce the number of MFIs and subsequently the count of discovered rules. Therefore, only those rules interesting to the user can be generated. Interestingly, the incremental nature of the proposed approach makes it significantly flexible to cope with time-changing data and user-changing beliefs. This enforces its computing functionality with two or more databases arriving at different courses of time with different volumes and from different locations. Hence, incremental updating of the discovered knowledge is a striking feature of the proposed algorithm. However, a keyword for generating the association rules is to identify the measure of Conf. This can be calculated according to the following equation:

Table 3: NM calculation between the new MFIs {a,s} and all IMFIs in IMFI list.

S_1	$\|S_1\|$	S_2	$\|S_2\|$	K	NM	minNM	≥minNovlty
{r,m,s}	3			1	$((3+2)-(2\times 1))/(3+2)=3/5=0.6$		
{r,b,m}	3	{a,s}	2	0	$((3+2)-(2\times 0))/(3+2)=5/5=1$	0.6	Yes
{m,t}	2			0	$((2+2)-(2\times 0))/(2+2)=4/4=1$		

Table 4: NM calculation between the new MFIs {u,s,m} and all IMFIs in IMFI list.

S_1	$\|S_1\|$	S_2	$\|S_2\|$	K	NM	minNM	≥ minNovlty
{r,m,s}	3			2	$((3+3)-(2\times 2))/(3+3)=2/6=0.33$		
{r,b,m}	3	{u,s,m}	3	1	$((3+3)-(2\times 1))/(3+3)=4/6=0.2$	0.2	No
{m,t}	2			1	$((2+3)-(2\times 1))/(2+3)=3/5=0.6$		
{a,s}	2			1	$((2+3)-(2\times 1))/(2+3)=4/4=0.6$		

$$\text{conf} = \frac{\text{support}(A \longrightarrow B)}{\text{suport}(A)}, \tag{6}$$

where support $(A \longrightarrow B)$ is the number of transactions containing itemsets A and B together, and support (A) is the count of transactions composing itemset A [50].

3.5. Proposed Framework Algorithms

3.5. Proposed Framework Algorithms. As stated earlier, five phases have been proposed by our approach framework representing algorithms 1–4. Algorithm 1 (IIMFI algorithm) includes these phases, which call other algorithms.

Phase 1. First scan of Di + 1. Lines 1–4 in Algorithm 1 explain Phase 1. This phase consists of two tasks.

First Task: a new PD_IMFIs are created as Pi + 1. This is the integration between the previous PD_IMFI Pi and Di + 1. Algorithm 2 (created PD_IMFI algorithm) describes these tasks.

Second Task: the integer value of cur_Minsup is calculated for Di + 1 as to equation (3), and incr_Minsup is updated by increasing its value with cur_Minsup for each item in Pi + 1.

Phase 2. Create a new tree as FP-tree structure. Lines 5–6 in Algorithm 1 explain Phase 2, which calls Algorithm 3 (created tree algorithm), describing Phase 2. This phase consists of three tasks.

First Task: the tree is built by adding the previously discovered IMFIs (along with their corresponding support) of Pi to the tree.

Second Task: the construction of the tree is completed by reading the records from the Di + 1.

Third Task: the support value for each item in the header table and the counter for the nodes in the tree are updated so that it becomes equal to incr_Sup for that item in Pi + 1.

Phase 3. Extract MFIs from the tree using FP-Max algorithm, conducting bottom-up looping of each item in Pi + 1 where cur_Sup ≥ cur_Minsup, or incr_Sup ≥ incr_Minsup. Lines 7-8.1. 2 in Algorithm 1 explain Phase 3. In line 8.1.3, if useNM is true, then Phase 4 (lines 8.1.3.1-line 8.1.3.2) and Phase 5 are proceeded; else, line 8.1.4 Phase 5 is proceeded (lines 8.1.4.1-8.1.4.2).

Phase 4. Perform dynamic pruning of MFIs using NM constraint. In this phase, one task is to calculate the NM between MFIs and all IMFIs in Pi + 1 and select the minNM value from NM values. Algorithm 4 (interesting measure algorithm) explains the first task. If the minNM ≥ minNovlty, then the MFIs are interesting (IMFIs), so Phase 5 is proceeded, else the MFIs are discarded; thus Phase 3 is processed until the first item in Pi + 1.

Phase 5. This phase runs two tasks.

First Task: the list of IMFIs in Pi + 1 is updated by adding new MFIs to the list, as represented in line 8.1.4, Algorithm 1.

Second Task: rules from MFIs are generated, which have Conf ≥ minConf, by calling association rule generation algorithm, Algorithm 1, line 8.1.5.

3.6. Time Complexity. Regarding the time complexity of the developed algorithm (IIMFIs), the complexity varied vis-à-vis the algorithmic conditioning of the mining process at three divided times and the total time spent on the incremental database. The first algorithmic step was to call Algorithm 2 to create PD_IMFIs, and the time complexity is $O(n\times i)$, where n is the total number of transactions in the databases and i is the number of items in n. The time complexity here is almost similar to all the other algorithms when performed on static databases. However, in the case of incremental databases, the time is less complex in Algorithm 2 of our proposed method compared with other algorithms due to the algorithmic condition that if any item has been already discovered, it needs just the update of P data and not the creation of a new item. As Algorithm 3 was called to create the updated tree, two recursive cycles were involved. The first cycle was to build the tree from the previously discovered MFIs associated only with the items having incremental support cur_Sup value ≥ cur_Minsup value, or incr_Sup ≥ incr_Minsup, and the time complexity here is $o(n)$. The second recursive cycle was used to conduct the second scan of the databases to complete the construction of the tree, and the time complexity for this step is the same as for Algorithm 2, $O(n\times i)$. It is to be noted that the tree

```
Input:
Di + 1
Pi // when $i = 0$ T0, the P0 is null.
useNM // its optional, either true or false. If true call Algorithm 4 to calculate minNM.
minNovlty, minSup, minConf are values $\geq 0$ and $\leq 1$, entered by user.
Output:
Pi + 1 // a new PD_IMFIs output at now time Ti + 1
Phase 1: Create PD_IMFIs as Pi + 1 and incremental support and minimum threshold
(1) Pi + 1 = Call Created PD_IMFIs Algorithm (Di + 1, Pi)
(2) cur_Minsup = Min_Sup $* n$
(3) For each item in Pi + 1 as $P$
    (3.1) P. incr_Minsup + = cur_Minsup
(4) End for
Phase 2: Create of tree, it includes previous IMFIs and transactions in Di + 1
(5) tree = null // as structure FP-tree
(6) tree = Call Created Tree Algorithm (Pi + 1, Di + 1, cur_Minsup)
Phase 3: Find MFIs from tree based on FP-Max algorithm
(7) MFI-tree = null
(8) For each item from bottom to up in Pi + 1 as P // where from bottom to up as header table with tree
    (8.1) If P. cur_Sup $\geq$ cur_Minsup or P. incr_Sup $\geq$ P. incr_Minsup Then
        (8.1.1) MFIsnew = null
        (8.1.2) MFIsnew = call FP-Max Algorithm (tree, MFI-tree)
        (8.1.3) If useNM = = true Then // Go to Phase 4:
Phase 4: Incremental Dynamic Pruning of MFIs
            (i) Min$NM^{Pi+1}_{MFIs\text{new}}$ = call Interesting Measure Algorithm (Pi + 1, MFIsnew) // to determine the minimum value of the deviation between MFIsnew and all IMFIs in Pi + 1.
            (ii) If Min$NM^{Pi+1}_{MFIs\text{new}}$ minNovlty Then
                (a) Add MFIsnew to P.list_IMFIs in Pi + 1
            (iii) End if
        (8.1.4) End if
Phase 5: Generation Association Rules
        (8.1.5) Call Association Rule Generation Algorithm (MFIsnew, minConf)
    (8.2) End if
    (9) End for
(10) Return Pi + 1
```

Algorithm 1: Main IIMFI algorithm.

```
Input:
Di + 1, Pi
Output:
Pi + 1
(1) Pi + 1 = Pi
(2) For each transaction $T$ in Di + 1 // first scan current database
    (2.1) For each item I in $T$
        (2.1.1) If I is equal item_Name for each item in Pi + 1 as P Then
            (i) increase value of P. cur_Sup, and P. incr_Sup by 1
        (2.1.2) Else
            (i) Create a new item as Pnew with default value fields item_Name = I, cur_Sup = 1, incr_Sup = 1, incr_Minsup = 0, and list_IMFIs = null
            (ii) add a Pnew to 1-Items list in Pi + 1
        (2.1.3) End if
    (2.2) End for
(3) End for
(4) Re-sort Pi + 1 in descending order by the value of incr_Sup filed.
(5) Return Pi + 1
```

Algorithm 2: Created PD_IMFI algorithm.

```
Input:
Pi + 1, Di + 1, cur_Minsup
Output:
tree // tree is update includes all items in IMFIs and Di + 1
(1) For each item in Pi + 1 as P
  (1.1) If P.list_IMFIs! = null and (P. cur_Sup ≥ cur_Minsup or P.incr_Sup ≥ P.incr_Minsup) Then
    (a) For each IMFIs in P. list_IMFIs
      (i) Add all itemsets in IMFIs to new array as arrayTrns
      (ii) Re-sort items of arrayTrns in descending order as items in Pi + 1
      (iii) tree = Call FP-Growth algorithm to build tree (arrayTrns, IMFIs.support).
    (b) End for
    (c) P.list_IMFIs = null // reset list_IMFIs as null or remove all IMFIs from P.list_IMFIs
  (1.2) End if
(2) End for
(3) For each transaction $T$ in Di + 1 // second scan new database Di + 1.
  (3.1) add all items in $T$ to new array as arrayTrns //where item is equal to item_Name; and cur_Sup ≥ cur_Minsup or incr_Sup ≥ incr_Minsup in Pi + 1.
  (3.2) Re-sort items of arrayTrns in descending order as Pi + 1.
  (3.3) tree = Call FP-Growth algorithm to build tree (arrayTrns).
(4) End for
(5) Update support in header table, and counter nodes on main path to equal incr_Sup for each item in Pi + 1 where the item in header table and node equal item_Name and the same level on main path and Pi.
(6) Return tree
```

Algorithm 3: Created tree algorithm.

```
Input:
Pi + 1, MFIsnew
Output:
minNM
(1) minNM = 1
(2) For each Item in Pi + 1 as P
  (2.1) For each IMFIs in P.list_IMFIs // where P.list_IMFIs not null
    (2.1.1) NMtemp = $(S_1 + S_2) - (2 * k)/(S_1 + S_2)$ //where $S_1$ is size of IMFIs, $S_2$ is size of MFIsnew, $k$ is count items of the similar between IMFI and MFIsnew.
    (2.1.2) If NMtemp < minNM then
      (a) minNM = NMtemp
    (2.1.3) End if
  (2.2) End for
(3) End for
(4) Return minNM
```

Algorithm 4: Interesting measure algorithm.

construction in IIMFIs is much faster and more efficient because the assumed pathways of the tree have already been constructed in the first recursive cycle. The second cycle involved only completion of the tree construction through just updating the counter for each node. The worst time complexity is O(n3) where Algorithm 4 was called within the recursive cycle of Algorithm 1 in case useNM = true, but if useNM = false, Algorithm 4 would not be called and, the time complexity, in this case, is O(n2).

3.7. A Detailed Example. Assume D1, D2, and D3 are three databases coming at three different times T1, T2, and T3, as shown in Tables 5–7, respectively. The last column (ordered

Table 5: Database D1 at T1.

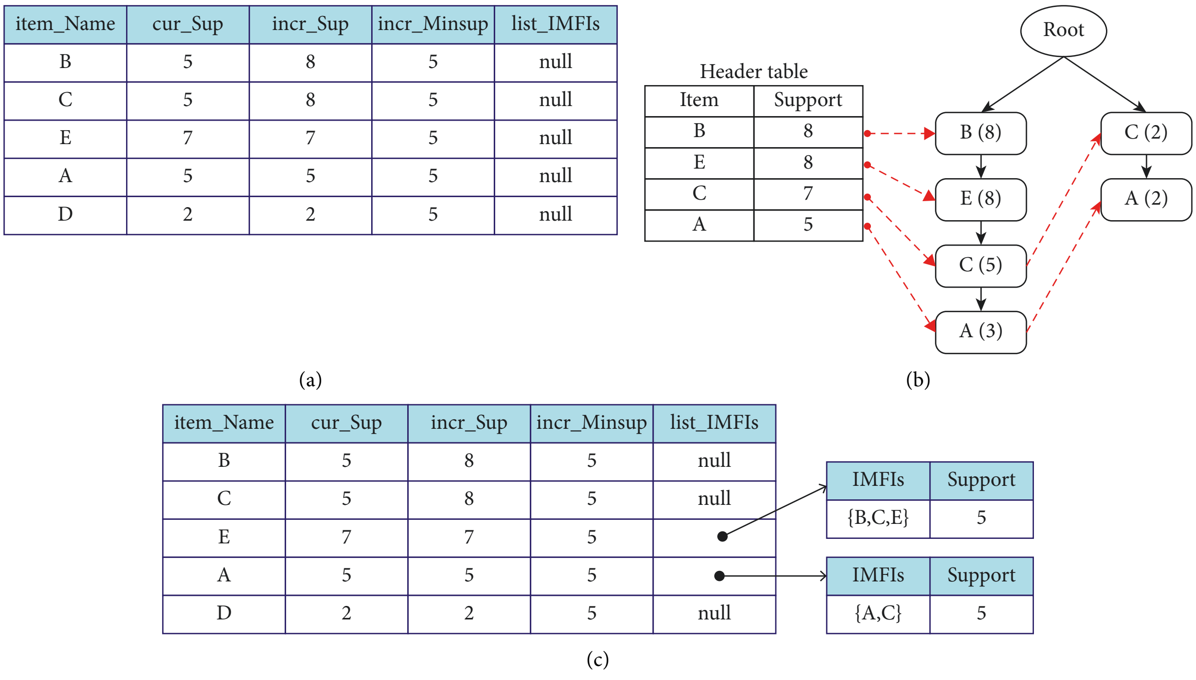
Tid	Itemsets	Ordered itemsets
1	A C D	C A
2	B C E	B E C
3	A B C E	B E C A
4	B E	B E
5	A C D	C A
6	B C E	B E C
7	A B C E	B E C A
8	B E	B E
9	A B C E	B E C A
10	B E	B E

Table 6: Database D2 at T2.

Tid	Itemsets	Ordered itemsets
1	D C B	B C D
2	B C E G	B C E G
3	D B G	B D G
4	A C D G	C D G
5	B C E G	B C E G
6	A B C E D	B C E D

Table 7: Database D3 at T3.

Tid	Itemset	Ordered itemsets
1	E B C	B C E
2	C E B R	B C E
3	G B D A M	B M
4	B A C	B C
5	M R B	B M
6	C B E M	B C E M



(a)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	5	8	5	null
C	5	8	5	null
E	7	7	5	null
A	5	5	5	null
D	2	2	5	null

(b)

Header table

Item	Support
B	8
E	8
C	7
A	5

(c)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	5	8	5	null
C	5	8	5	null
E	7	7	5	●
A	5	5	5	●
D	2	2	5	null

IMFIs	Support
{B,C,E}	5

IMFIs	Support
{A,C}	5

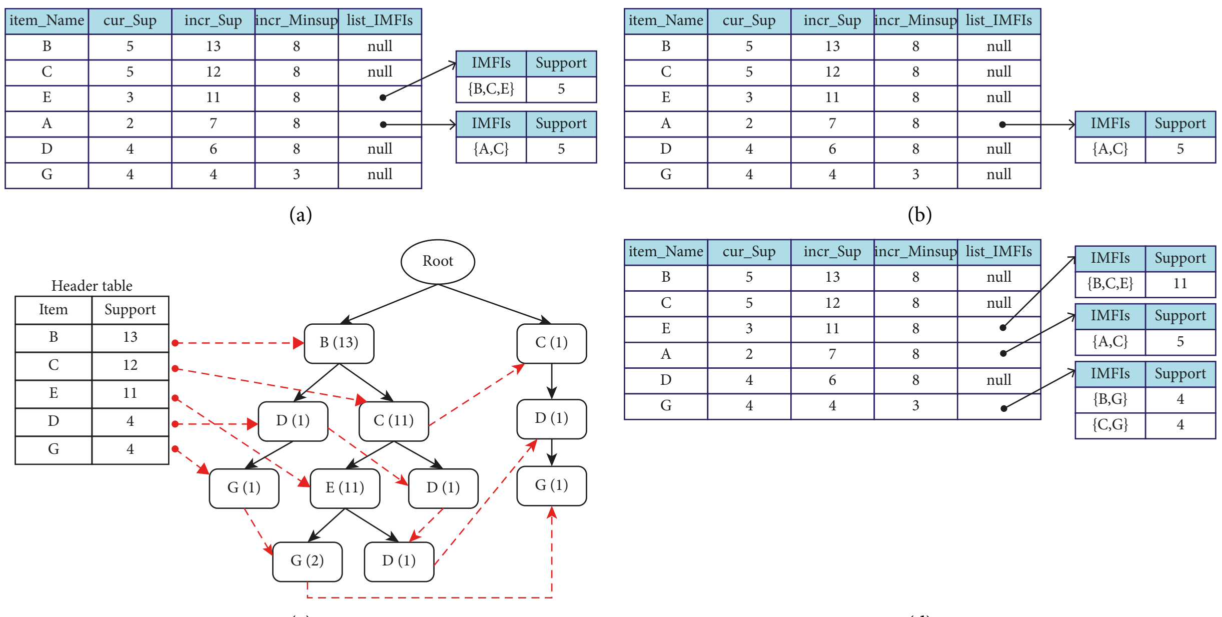
Figure 3: Example at T1. (a) P1 after phase 1, (b) tree as FP-tree after phase 2, and (c) P1 after phases 3-4.

Table 8: Rules that have Conf ≥0.7 after phase 5 at T1.

No.	Rule	Conf	No.	Rule	Conf	No.	Rule	Conf
1	$A \longrightarrow C$	1	4	$B \longrightarrow E$	1	7	$CE \longrightarrow B$	1
2	$C \longrightarrow A$	0.71	5	$E \longrightarrow B$	1	8	$BC \longrightarrow E$	1
3	$C \longrightarrow B$	0.71	6	$C \longrightarrow E$	0.71	9	$C \longrightarrow BE$	0.71

items) in these tables shows the representation of items in descending order according to incr_Sup value for each item in 1-Items list as used in the construction of the tree. Let Min_Sup = 0.5, minNovlty = 0.5, and minConf = 0.7.

At T1 Phase 1, P_1 is created, all items in D_1 to1-Items list with cur_Sup and incr_Sup = support of item in D_1 are added, and incr_Minsup is set; here, = cur_Minsup, and the value of cur_Minsup = 0.5 $*10 = 5$. Figure 3(a) shows P1 after Phase 1. Phase 2: the tree is created as shown in Figure 3(b). At T1, the tree is built only from transactions in D1. Phase 3: bottom-up looping of each item in 1-Items list from P1 is started; if cur_Sup ≥ cur_Minsup or incr_Sup ≥ incr_Minsup, it is started



(a)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	5	13	8	null
C	5	12	8	null
E	3	11	8	
A	2	7	8	
D	4	6	8	null
G	4	4	3	null

IMFIs	Support
{B,C,E}	5

IMFIs	Support
{A,C}	5

(b)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	5	13	8	null
C	5	12	8	null
E	3	11	8	null
A	2	7	8	
D	4	6	8	null
G	4	4	3	null

IMFIs	Support
{A,C}	5

(c)

Header table

Item	Support
B	13
C	12
E	11
D	4
G	4

(d)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	5	13	8	null
C	5	12	8	null
E	3	11	8	
A	2	7	8	
D	4	6	8	null
G	4	4	3	

IMFIs	Support
{B,C,E}	11

IMFIs	Support
{A,C}	5

IMFIs	Support
{B,G}	4
{C,G}	4

FIGURE 4: Example at T2. (a) P2 after phase 1, (b) P2 after phase 2, (c) tree as FP-tree after phase 2, and (d) P2 after phases 3-4.

TABLE 9: Rules that have Conf ≥0.7 after phase 5 at T2.

No.	Rule	Conf	No.	Rule	Conf	No.	Rule	Conf
1	$A \longrightarrow C$	1	7	$C \longrightarrow E$	0.91	13	$CE \longrightarrow B$	1
2	$C \longrightarrow A$	0.71	8	$E \longrightarrow C$	1	14	$B \longrightarrow CE$	0.84
3	$B \longrightarrow C$	0.85	9	$BC \longrightarrow E$	1	15	$G \longrightarrow B$	1
4	$C \longrightarrow B$	0.91	10	$E \longrightarrow BC$	1	16	$G \longrightarrow C$	1
5	$B \longrightarrow E$	0.85	11	$BE \longrightarrow C$	1			
6	$E \longrightarrow B$	1	12	$C \longrightarrow BE$	0.91			

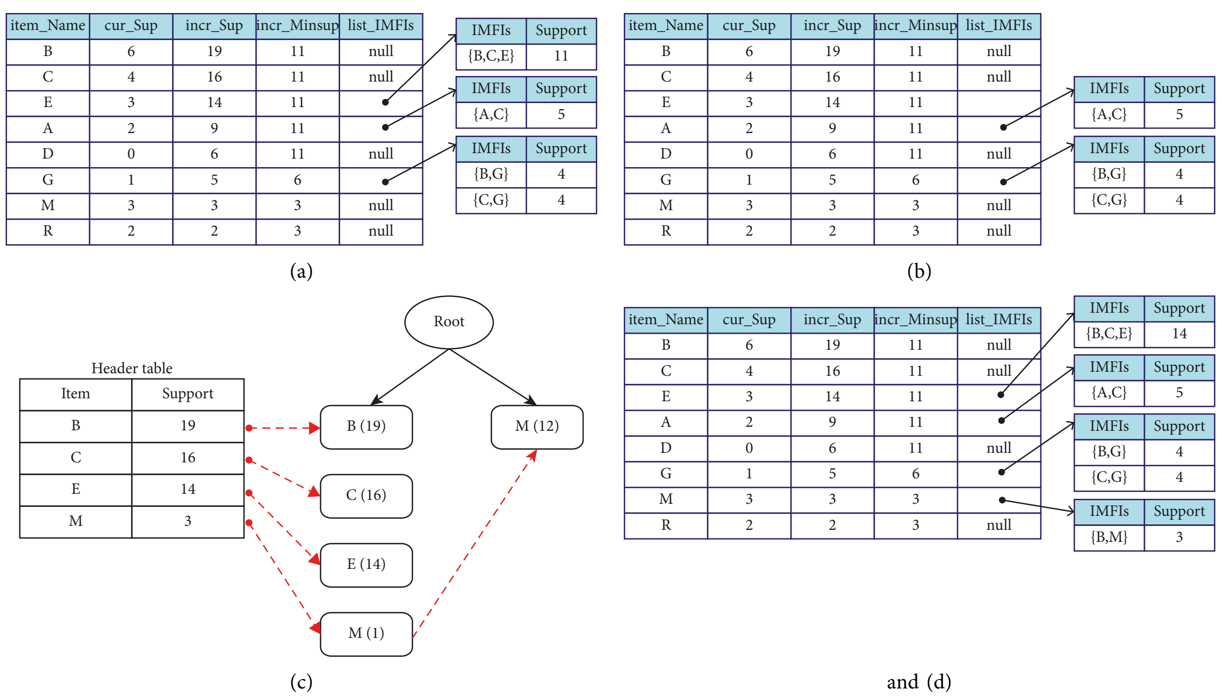
from item {A} and then FP-Max is called for mining MFIs, and its output is MFIsnew = {A, C:5}. Phase 4: the value of Min$NM^{IMFIslist}_{\{A,C\}}$ is calculated, and its value = 1, because IMFI list for each item in 1-Items list from P1 is null. Then, the value of Min$NM^{IMFIslist}_{\{A,C\}}$ ≥ minNovlty (1 ≥ 0.5) is compared; i.e., MFIs {A, C:5} are interesting, so it is added to IMFI list of the item {A}. Phase 5: rules from it are generated where each rule has Conf ≥0.7. Then, phase 3 is proceeded for taking item{C}. The output of this phase is MFIs = {B, C, E:5}. Phase 4: the value of $NM^{AC}_{BCE} = ((2+3) - (2*1)/2+3) = 0.6$ is calculated, which is the minNM. Then, the value of Min$NM^{IMFIslist}_{BCE}$ ≥ minNovlty (.6 ≥ .5) is compared; thus, MFIs {B, C, E:5} are interesting, so it is added to IMFI list with item {C}. Phase 5: rules from it are generated where each rule has Conf ≥0.7. Then, Phase 3 is proceeded, and item{E} is taken; this phase is not the output MFIs for item{E}, so Phase 4 is not processed; also, item {B} has no MFIs. Figure 3(c) shows P1 after phases 3-4 at T1. Table 8 shows generated rules that have Conf ≥0.7 after phase 5 at T1. (Note) At T1, 9 rules are interesting (1-9) (green color).

At T2, Phase 1, P2 = P1 is created, and then, cur_Sup = support of item in D2, incr_Sup+ = cur_Sup, and incr_Minsup + = cur_Minsup are set. At T2, the value of cur_Minsup = 0.5 ∗ 6 = 3. Figure 4(a) shows P2 after phase 1. Phase 2: the tree is created. Figure 4(b) shows P2, and Figure 4(c) shows the tree after phase 2. The outcome of phases 3-4 is P2 as Figure 4(d) shows. Phase 5: rules that have Conf ≥0.7 are generated as shown in Table 9. (Note) At T2, only 2 rules are interesting (rule 15 and rule 16) (green color), 14 rules are not interesting, 12 rules undergo change in value Conf (3-14) (red color), and 2 rules experience no change in value Conf (1-2).

At T3, Phase 1: P3 = P2 is created, and then, cur_Sup = support of item in D3, incr_Sup+ = cur_Sup, and incr_Minsup + = cur_Minsup are set. At T3, the value of cur_Minsup = 0.5 ∗ 6 = 3. Figure 5(a) shows P3 after Phase 1. Phase 2: the tree is created. Figure 5(b) shows P3, and Figure 5(c) shows the tree after Phase 2. Figure 4(d) shows P3 after phases 3-4. Phase 5: rules that have Conf ≥0.7 are generated as shown in Table 10. (Note) At T3, only 1 rule is interesting (rule 17) (green color), 16 rules are not interesting, 12 rules undergo change in value Conf (3-14) (red color), and 4 rules undergo no change in value Conf (1,2,15,16). The results of the previous example at three times are summarized in Table 11.

4. Experiments and Results

To evaluate the performance of the proposed algorithm (IIMFIs), three experiments were conducted. The first experiment was performed to compare the running time of


(a)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	6	19	11	null
C	4	16	11	null
E	3	14	11	●
A	2	9	11	●
D	0	6	11	null
G	1	5	6	●
M	3	3	3	null
R	2	2	3	null

IMFIs	Support
{B,C,E}	11

IMFIs	Support
{A,C}	5

IMFIs	Support
{B,G}	4
{C,G}	4

(b)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	6	19	11	null
C	4	16	11	null
E	3	14	11	
A	2	9	11	●
D	0	6	11	null
G	1	5	6	●
M	3	3	3	null
R	2	2	3	null

IMFIs	Support
{A,C}	5

IMFIs	Support
{B,G}	4
{C,G}	4

(c)

Header table

Item	Support
B	19
C	16
E	14
M	3

and (d)

item_Name	cur_Sup	incr_Sup	incr_Minsup	list_IMFIs
B	6	19	11	null
C	4	16	11	null
E	3	14	11	●
A	2	9	11	●
D	0	6	11	null
G	1	5	6	●
M	3	3	3	●
R	2	2	3	null

IMFIs	Support
{B,C,E}	14

IMFIs	Support
{A,C}	5

IMFIs	Support
{B,G}	4
{C,G}	4

IMFIs	Support
{B,M}	3

Figure 5: Example at T3. (a) P3 after Phase 1, (b) P3 after phase 2, (c) tree as FP-tree after phase 2, and (d) P3 after phases 3-4.

Table 10: Generated rules that have Conf ≥0.7 after phase 5 at T3.

No.	Rule	Conf	No.	Rule	Conf	No.	Rule	Conf
1	$A \longrightarrow C$	1	7	$C \longrightarrow E$	0.88	13	$CE \longrightarrow B$	1
2	$C \longrightarrow A$	0.71	8	$E \longrightarrow C$	1	14	$B \longrightarrow CE$	0.74
3	$B \longrightarrow C$	0.74	9	$BC \longrightarrow E$	0.88	15	$G \longrightarrow B$	1
4	$C \longrightarrow B$	0.88	10	$E \longrightarrow BC$	1	16	$G \longrightarrow C$	1
5	$B \longrightarrow E$	0.74	11	$BE \longrightarrow C$	1	17	$M \longrightarrow B$	1
6	$E \longrightarrow B$	1	12	$C \longrightarrow BE$	0.88			

Table 11: Results of the example at times T1, T2, and T3.

Time	MFIs	Uninteresting rules	IMFIs	Interesting rules
T1	2	9	2	9
T2	4	16	2	2
T3	5	17	1	1

Table 12: Characteristics of experimental datasets.

Dataset	Avr. length	Count items	Count trans.
T10I4D100K	10	870	100000
Mushroom	23	119	8124
T25I10D10K	24	929	9976
Accidents	33	468	340183
Kosarak	8	41270	990002

Table 13: Characteristics of datasets parts at three times.

Dataset	Time	Avr. length	Count items	Count trans.
T10I4D100K	T1	10	869	40000
	T2	10	869	30000
	T3	10	868	30000
Mushroom	T1	23	78	2708
	T2	23	63	2708
	T3	23	106	2708
T25I10D10K	T1	22	865	3400
	T2	26	901	3400
	T3	27	917	3176
Accidents	T1	34	412	150000
	T2	34	372	100566
	T3	34	368	89617
Kosarak	T1	8	34062	400000
	T2	8	31407	300000
	T3	8	30703	290002

IIMFI algorithm with two incremental algorithms, namely IM_WMFI and IMU2P-Miner on datasets incrementally, as explained in Section 4.1. The second experiment was performed to compare the running time of IIMFI algorithm on static datasets against two state-of-the art static algorithms, namely CL-Max and SelPMiner, as explained in Section 4.2. (Note that in these two experiments, all algorithms discovered the same MFIs, confirming that the results produced by the algorithms in our experiments are complete and correct.) The third experiment was performed to test the effect of NM on reducing the count of the discovered MFIs as described in Section 4.3.

All algorithms were coded in Java programming language. The three experiments were run on a PC with an Intel Core i5 2.60 GHz CPU and 4 GB of RAM. The operating system was Windows 10 Pro (64-bit).

The experiments were conducted on five datasets from the FIMI repository: URL: http://fimi.ua.ac.be. The datasets are commonly used by researchers in the field of ARM. These

Table 14: Runtime at T1, T2, T3, and total runtime (Sec.) for IIMFIs, IM_WMFI, and IMU2P-Miner algorithms on five datasets with different Min_Sup values.

Dataset	Min_Sup (%)	T1 Runtime (Sec.)			T2 runtime (Sec.)			T3 runtime (Sec.)			Total runtime (Sec.)		
		Algorithms			Algorithms			Algorithms			Algorithms		
		IM_WMFI	IMU2P-Miner	IIMFIs	IM_WMFI	IMU2P-Miner	IIMFIs	IM_WMFI	IMU2P-Miner	IIMFIs	IM_WMFI	IMU2P-Miner	IIMFIs
T10I4D100K	0.2	1.93	1.61	0.82	2.28	1.90	0.62	2.79	2.32	0.59	6.99	5.83	2.03
	0.4	0.89	0.63	0.55	0.72	0.51	0.42	0.76	0.54	0.43	2.36	1.69	1.40
	0.6	0.71	0.44	0.35	0.54	0.34	0.27	0.58	0.36	0.29	1.83	1.14	0.91
	0.8	0.65	0.36	0.29	0.48	0.27	0.21	0.47	0.26	0.21	1.60	0.89	0.71
	1	0.57	0.29	0.23	0.43	0.21	0.17	0.42	0.21	0.17	1.42	0.71	0.57
Mushroom	5	0.22	0.21	0.20	0.25	0.24	0.22	0.33	0.31	0.30	0.80	0.76	0.72
	10	0.04	0.04	0.04	0.06	0.05	0.05	0.07	0.07	0.06	0.17	0.16	0.14
	15	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.07	0.06	0.05
	20	0.02	0.01	0.01	0.02	0.02	0.01	0.02	0.02	0.01	0.06	0.05	0.04
	25	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.04	0.03	0.03
T25I10D10K	0.6	0.57	1.19	0.36	0.89	3.10	0.56	1.13	4.57	0.71	2.59	8.86	1.62
	0.8	0.36	0.45	0.20	0.52	1.24	0.29	0.48	1.86	0.27	1.35	3.56	0.75
	1	0.33	0.42	0.19	0.51	0.98	0.26	0.47	1.34	0.24	1.32	2.74	0.68
	1.2	0.32	0.15	0.15	0.46	0.60	0.21	0.46	0.69	0.21	1.23	1.45	0.56
	1.4	0.32	0.08	0.13	0.44	0.30	0.19	0.46	0.53	0.19	1.22	0.90	0.52
Accidents	40	9.81	7.01	2.16	9.07	6.48	1.64	10.40	7.43	1.28	29.28	20.92	5.07
	45	4.04	2.78	1.70	3.23	2.23	1.31	3.64	2.51	1.05	10.90	7.52	4.06
	50	2.20	1.46	1.26	1.54	1.03	0.95	1.55	1.03	0.80	5.29	3.52	3.01
	55	1.49	0.96	1.13	1.01	0.65	0.83	0.94	0.61	0.74	3.44	2.22	2.70
	60	1.32	0.83	1.07	0.89	0.56	0.76	0.80	0.50	0.64	3.01	1.88	2.47
Kosarak	0.2	8.87	7.39	6.93	10.17	8.48	4.51	9.76	8.14	4.35	28.80	24.00	15.79
	0.3	3.62	2.78	1.86	2.59	2.00	1.38	2.54	1.95	1.39	8.75	6.73	4.63
	0.4	2.37	1.69	1.60	1.76	1.26	1.19	1.69	1.21	1.14	5.82	4.15	3.92
	0.5	2.01	1.34	1.36	1.48	0.98	1.02	1.41	0.94	0.99	4.89	3.26	3.37
	0.6	1.72	1.08	1.24	1.28	0.80	0.94	1.27	0.80	0.93	4.28	2.68	3.11

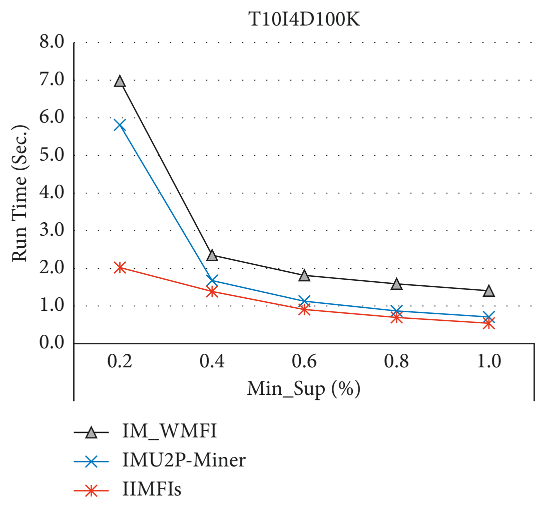
datasets are varied between real or synthetic datasets in terms of the number of transactions and the density of the transaction (average length of a transaction in datasets). Table 12 shows the characteristics of these datasets. For the experiment on incremental mining, each dataset was divided into three parts representing T1, T2, and T3, respectively, as shown in Table 13.

4.1. Experiment 1 (Runtime Dynamic State). In this experiment, the computation of runtime of IIMFIs is performed against the well-known incremental IM_WMFI and IMU2P-Miner algorithms as shown in Table 14. Note that running time here means the execution time (Sec.), which is the period between the input of dataset and the finish mining for each time: time 1 (T1), time 2 (T2), and time 3 (T3). Total runtime refers to the sum of runtime (Sec.) at T1, T2, and T3. This experiment was used in the first phase to discover MFIs without stepping to the rule generation phase. The experiment calculates the runtime at T1, T2, and T3 on three classified times of given datasets as shown in Table 13 and then compares the total executed runtime of IIMFIs, IM_WMFI, and IMU2P-Miner algorithms as shown in Table 14. (Note) In IIMFI algorithm, useNM = false; subsequently, Algorithm 4 used to calculate NM is not called.

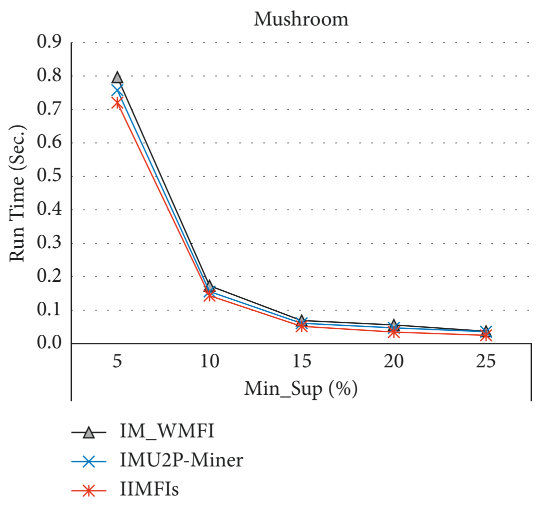
Figures 6(a)–6(e) reflect the results of the total runtime (Sec.) of the three algorithms on the five datasets with different Min_Sup values. As illustrated in Figure 6(a) that represents the results of the total runtime on T10I4D100K, IIMFI algorithm records the least runtime, especially at Min_Sup = 0.2%, followed by IMU2P-Miner algorithm that takes a reasonable runtime and appears closer to IIMFI algorithm, particularly at Min_Sup = 1.0%, whereas IM_WMFI shows a slowing performance. However, the runtime reduction gap between IIMFIs and IMU2P-Miner and IM_WMFI is not consistent as it declines at higher Min_Sup values. Similarly, on T25I10D10K, as Figure 6(c) shows, IIMFIs outperform the other algorithms at all Min_Sup values. The other algorithms show an oscillated runtime where IM_WMFI almost outpaces IMU2P-Miner at all Min_Sup except at 1.2%, where they almost exploit the same runtime, and at 1.4% where IMU2P-Miner shows better runtime performance. Figure 6(b) reveals that on Mushroom dataset, almost all the three algorithms take the same total runtime. A slight difference occurs at Min_-Sup = 5% where IIMFIs appears a little bit faster. At Min_Sup = 25%, however, the total runtime is almost the same for the three algorithms. When operated on Accidents as in Figure 6(d), IIMFIs perform generally well at lower Min_Sup = 40% to 50%, while its performance reverts at the higher Min_Sup = 55% to 60%. On the contrary, IMU2P-Miner shows better performance at higher Min_Sup values, while it slows down at lower Min_Sup values; IM_WMFI stays behind the two, especially at lower Min_Sup values. Figure 6(e) shows the results of the total runtime on Kosarak where IIMFIs score a greater reduction runtime rate at Min_Sup = 0.2% and 0.3%, the same runtime of IMU2P-Miner at 0.4% and 0.5%, and slower than IMU2P-Miner at 0.6. IM_WMFI is generally the slowest at all Min_Sup.

The performance of the three algorithms, IIMFIs, IMU2P-Miner, and IM_WMFI, varies from one dataset to another depending on the characteristics of datasets (Figures 6(a)–6(e)). Generally, IIMFI algorithm is faster than IM_WMFI and IMU2P-Miner on all datasets. This may be due to the nature of its incremental tree structure that reduces counting loop and time through maintaining the discovered MIFs of the first time and updating incr_Sup only without rescanning the whole data. Based on the experimental results, IIMFIs generally achieve a higher runtime efficiency rate against other algorithms, particularly on T10I4D100K, T25I10D10K, and Accidents. However, on Mushroom and Kosarak datasets the IIMFI runtime oscillates. We can notice the efficiency of IIMFIs in dealing with high-weight and dense dynamic data as with T10I4D100K and T25I10D10K, where it scores superiority, especially at lower Min_Sup. As shown in Figures 6(c) and 6(d), IIMFIs score higher time efficiency at smaller Min_Sup values due to the count of items and MFIs since T25I10D10K and Accidents are dense datasets. However, at higher Min_Sup, and subsequently, with the high MFI count discovered, it slightly slows. This explains that the count of MFIs is higher at T1, which consequently affects the sum of time needed at T2 and consequently affects the sum of runtime at T3. As shown in Figure 6(a), the higher the Min_Sup value is, the greater the increase in the number of MFIs and, consequently, the less the reduction runtime rate. Since Mushroom is a sparse dataset but with high count items, the difference in runtime consumption is almost the same for the three algorithms (Figure 6(b)). Therefore, at T1 and T2, item number is greater, MFI count is higher, and Min_Sup value is lower. In this case, IIMFI performance is faster because the tree can hold and update the support of MFIs without the need to re-mine MFIs from the tree. It is observed that at Min_Sup = 1.4% on Mushroom, IMU2P-Miner algorithm works well at higher Min_Sup values, which indicates that this algorithm best operates on univariate uncertain dada as it uses a tree structure local array to keep the updates.

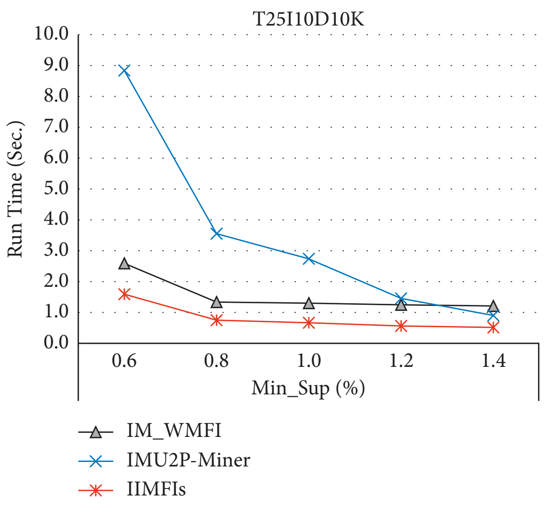
The evaluation of the execution time of the tested algorithms on dynamic data experimentally revealed that our algorithm was faster than all the other algorithms due to its ability to handle the MFI rule mining problem by updating the tree structure and generating fewer trees and, subsequently, shorter executed time. IM_WMFI took the longest time, especially at lower minimum supports or heavy datasets. This was most likely due to the larger number of the sub-generated trees and the schema of updating structure rules of the algorithm compared with faster mining of our algorithm due to its tree structure that does not allow subtrees and subsequently admits only new items or updates the cur_Sup, thus reducing the total runtime. It has been experimentally observed that when operating on large and high-weight datasets, IMU2P-Miner and our algorithm ran similarly well as they used similar FP-tree structure rules. This indicated that when the dataset's weight was too high, our algorithm was faster as it did not rescan the whole database, and hence, the runtime required to generate and scan the updated trees in our algorithm was also less than the



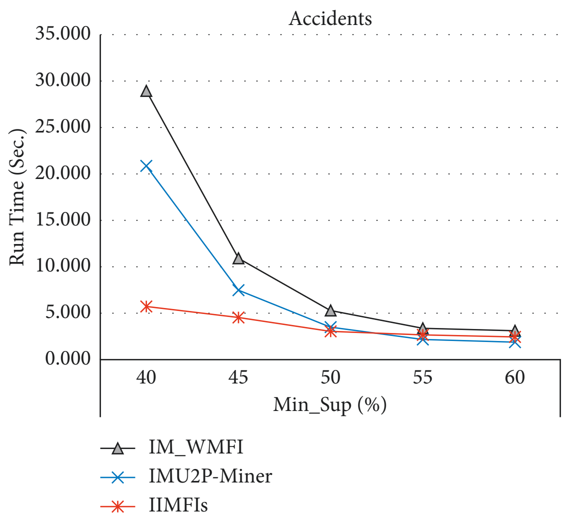
(a)



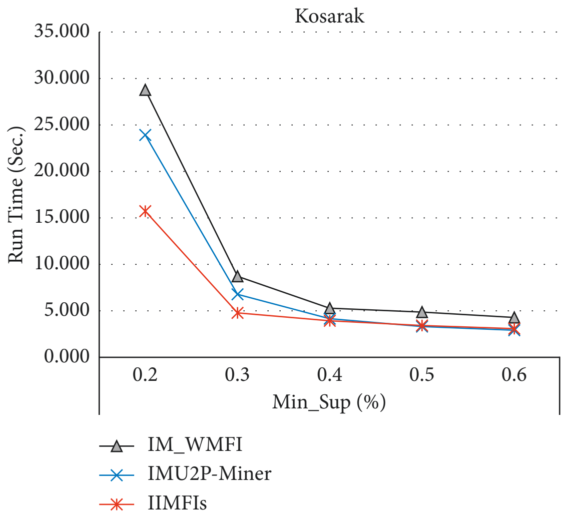
(b)



(c)



(d)



and (e)

FIGURE 6: Comparing the total runtime of T1, T2, and T3 (Sec.) between IM_WMFI, IMU2P-Miner, and IIMFI algorithms on the datasets. (a) T10I4D100K, (b) Mushroom, (c) T25I100D10K, (d) Accidents, and (e) Kosarak.

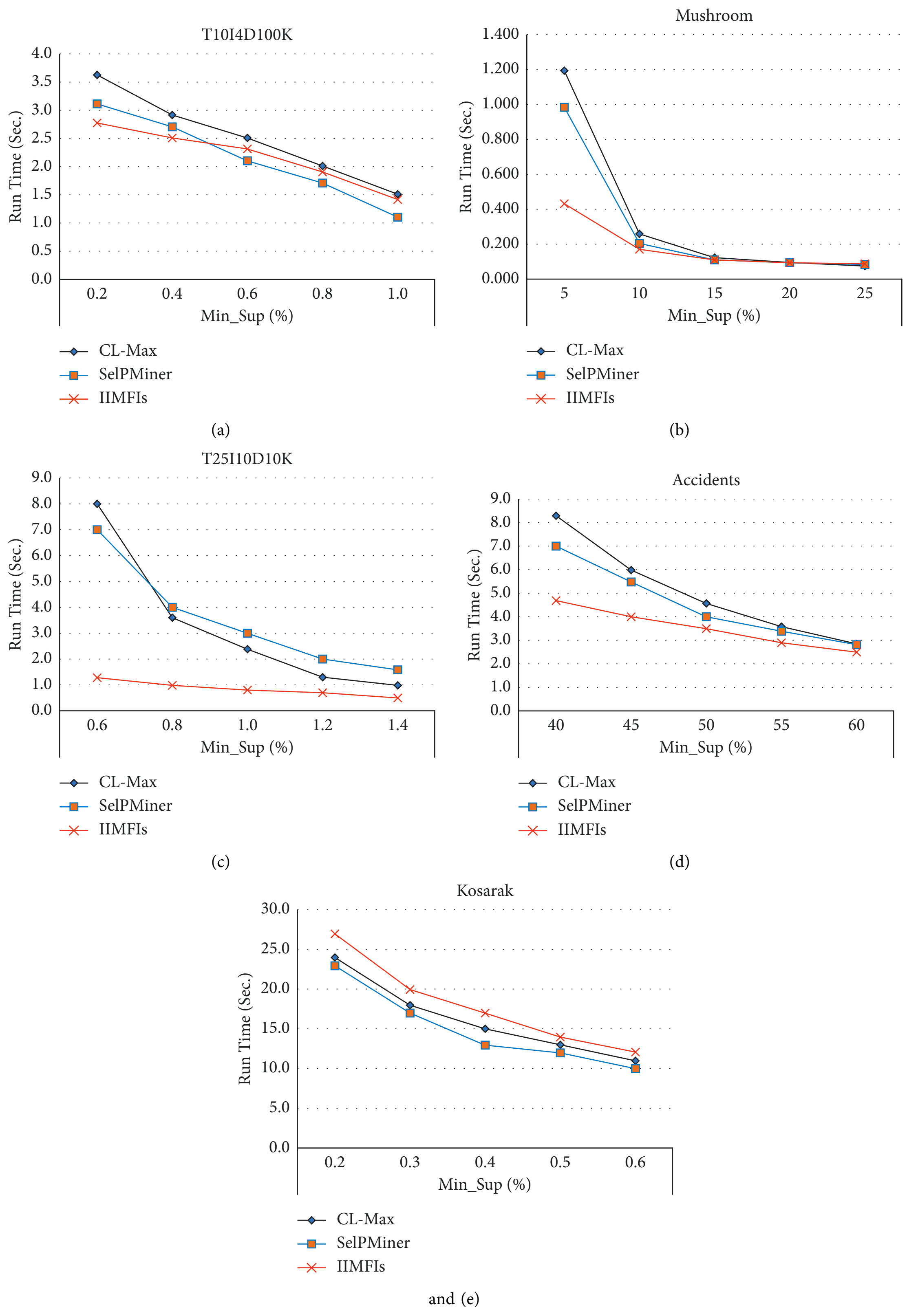


Figure 7: Comparing the runtime (Sec.) between CL-Max, SelPMiner, IIMFIs algorithms on the datasets at static state. (a) T10I4D100K, (b) Mushroom, (c) T25I100D10K, (d) Accidents, and (e) Kosarak.

Table 15: Effect of NM on reduction in MFIs and rules when minNovlty = 0.2 and rules having Conf ≥ 0.7.

Dataset	Min_Sup (%)	MFIs	MFI rules	IMFIs	IMFI rules	Reduction (-)	
						MFIs (%)	Rules (%)
T10I4D100K	0.8	432	100	36	89	92	11
	0.9	397	28	11	28	97	0
	1.0	370	18	7	18	98	0
	1.1	348	14	7	12	98	14
	1.2	322	2	1	2	100	0
Mushroom	5	1442	10053043	315	307770	78	97
	10	547	1106131	194	168846	65	85
	15	321	152798	131	101146	59	34
	20	158	67028	73	55358	54	17
	25	105	12512	56	7216	47	42
T25I10D10K	0.2	31424	7208014	30626	3520646	3	51
	0.3	19499	2693101	18217	1296805	7	52
	0.4	13824	2021162	12583	535968	9	73
	0.5	10913	3108115	8910	223342	18	93
	0.6	7879	836519	6613	74178	16	91
Accidents	40	762	256044	218	35291	71	86
	45	427	100146	138	18552	68	81
	50	216	37616	87	9703	60	74
	55	125	15878	55	5394	56	66
	60	78	7036	39	2762	50	61
Kosarak	0.4	467	3282	427	2549	9	22
	0.6	224	1420	212	1362	5	4
	0.8	127	755	119	688	6	9
	1.0	88	480	82	464	7	3
	1.2	67	276	64	269	4	3

runtime needed to do the same in the other algorithms because the updated trees in our algorithm were constructed from conditioned MFIs.

4.2. Experiment 2 (Runtime Static State). In this experiment, the computation of IIMFI runtime is performed against CL-Max and SelPMiner static algorithms. Figures 7(a)–7(e) show the results of the runtime (Sec.) on the five datasets at a static state. Each dataset has a different Min_Sup value. In this experiment, IIMFI algorithm's useNM = false.

Experimentally, when comparing the total runtime of the three algorithms, we notice that their performance fluctuates depending on the dataset characteristics and the mining structure of the algorithms whether they are tree-based or not. T25I10D10K and Accidents datasets are dense having an unsymmetrical distribution of the item count and transaction count (929 and 9976 for T25I10D10K; 468 and 340183 for Accidents, respectively) with relatively long patterns (average length is 24 for T25I10D10K and 33 for Accidents), while the datasets T10I4D100K and Kosarak are sparse with a variation of item count and transaction count (870 and 100000 for T10I4D100K; 41270 and 990002 for Kosarak, respectively) and characterized by relatively short patterns (average length is 10 for T10I4D100K and 8 for Kosarak). Mushroom is a sparse dataset characterized by symmetrical distribution of the maximal FPs with relatively long patterns (average length is 23) and small item count and transaction count (119 and 8124, respectively). Regarding the mining structure of the algorithms, IIMFIs and SelPMiner are tree-based, whereas CL-Max is not. Figure 7(a) shows the experimental results on T10I4D100K. As T10I4D100K contains smaller number of items, the performance of the algorithms fluctuates so that IIMFIs prove faster than CL-Max algorithm at all Min_Sup and outpace SelPMiner at lower Min_Sup = 0.2% and 0.4%. However, at Min_Sup = 0.6% to 1%, SelPMiner performs faster than IIMFIs and outperforms CL-Max algorithm at all Min_Sup. Also, on T25I10D10K, IIMFI algorithm shows superiority over CL-Max and SelPMiner algorithms at all Min_Sup, as in Figure 7(c). CL-Max algorithm outpaces SelPMiner at Min_Sup = 0.8% to 1.4%, except for 0.6%, where SelPMiner outperforms CL-Max algorithm. As for the performance of the three algorithms on Accidents dataset as illustrated in Figure 7(d), the IIMFI algorithm is generally faster than CL-Max and SelPMiner algorithms at all Min_Sup, while SelPMiner algorithm shows comfortably less runtime than CL-Max at all Min_Sup. When approaching the higher Min_Sup values, they have similar runtime consumption. As for Mushroom dataset (Figure 7(b)), IIMFIs take less execution time at lower Min_Sup = 5% and 10%, but similar runtime of the other algorithms at higher Min_Sup = 15% to 25%. This may be attributed to the characteristic parameters of Mushroom dataset as sparse but large, having fairly long pattern average, which makes it appropriately workable for all algorithms. Figure 7(e) shows that SelPMiner algorithm has faster pruning than IIMFIs and CL-Max algorithms at all threshold values when implemented on the Kosarak dataset. IIMFIs take the longest time and are outpaced by both

SelPMiner and CL-Max algorithms at all threshold values. This may be attributed to the search pattern of SelPMiner algorithm of selective partitioning, based on itemset-count tree, which is apt for compact datasets. Its MFI-tree structure works very well, especially when the dataset is sparse but very large. On the other hand, IIMFIs are working efficiently when a dataset is dense, so that at lower thresholds IIMFIs show an efficient runtime reduction rate compared with SelPMiner and CL-Max algorithms. The overall evaluation of experimental tests shows satisfactory results that our algorithm is still effective and satisfactorily fast.

4.3. Experiment 3 (Effect of NM on MFIs and Rules). In this experiment, we evaluated the effect of the NM on the reduction in MFI count and rules generated from MFIs. The experiment was applied to all the datasets with different multiple Min_Sup. In this experiment, useNM = true is used in the proposed algorithm, utilizing two minNovlty values 0 and 0.2, and minConf = 0.7. Table 15 shows the effect of NM on the reduction in MFIs when we applied minNovlty = 0.2. MFIs refer to the count of extracted MFIs without NM; i.e., minNovlty = 0; IMFIs refer to the count of the extracted MFIs with dynamic pruning when minNovlty = 0.2. MFI rules refer to the number of rules generated from MFIs where these rules have Conf ≥ minConf; IMFI rules refer to the number of rules generated from IMFIs where these rules have Conf ≥ minConf. Reduction (-) indicates the effect of NM at minNovlty = 0.2 on the pruning of MFIs and, subsequently, the reduction in the overall rules in each dataset at different Min_Sup values. The results in Table 15 show that since we are dealing with fixed values in static datasets, NM values are generally high.

As Table 15 reveals, there is a direct effect of NM on the count of MFIs and rules when minNovlty = 0.2 compared with the case when minNovlty = 0. NM reduces the count of MFIs and rules in all datasets at all dregs of Min_Sup. The highest effect is on T10I4D100K (92%–100%), while the least effect is on Kosarak (4%–9%). The difference in the percentage of the NM effect from one dataset to another is due to the intensity and degrees of Min_Sup in each dataset. It has been found that in some datasets, the effect is direct, i.e., the higher the degree of Min_Sup is, the greater the effect of the NM, as with T10I4D100K and T25I10D10K datasets. However, some of the datasets have the opposite effect; i.e., the higher the Min_Sup is, the lower the impact ratio of NM, as with Mushroom, Accidents, and Kosarak datasets. This is due to the count of MFIs at each Min_Sup and the average length of MFIs in each dataset.

5. Conclusion and Future Work

The study introduces a novel approach for mining incremental interesting MFIs by extending FP-Growth and FP-Max to reduce the scanning time and results of MFIs in datasets arriving at different times. The approach framework is structured in five phases representing the approach algorithm IIMFIs, which constitute the proposed design of this work. The proposed approach is incrementally self-adjusting in nature that integrates the previous and current data by adding and updating only new items or items with clearly defined support and incremental support values. Based on developing a tree structure mining method, the proposed approach has advantageously integrated objective and subjective measure (novelty measure) test in dynamic pruning so that only interesting MFIs are produced and only interesting rules are generated.

For evaluation purpose, three experiments were conducted on five datasets to test the efficiency of the proposed IIMFI method. Two experiments were performed to test the runtime efficiency of IIMFIs against two MFI incremental methods and two state-of-the-art static algorithms and an experiment to test the effectiveness of IIMFIs in reducing the number of MFIs and discovered rules by incorporating NM during mining process. The experimental results are promising, revealing that the proposed IIMFI method NM generally has a direct effect on the reduction in MFI count and rules in all datasets at all dregs and Min_Sup. However, the varying effect of IIMFI performance and NM on the number of MFIs differs from one dataset to another, which can be attributed to the nature of the dataset, its average length (density), and number of items.

Future work will be on testing the effectiveness of the proposed algorithm on other datasets with different degrees using the concept of "pre-large" to avoid errors in calculating the degree of the incr_Minsup, or using list structure for looking at the price or quantities of the items. More experiments may be conducted to evaluate the impact of subjective measures on objective measures. Future work is also suggested for developing a proposed algorithm that deals with parallel processing of data.

Data Availability

The dataset used in the experiment were taken from FIMI repository: URL: https://fimi.uantwerpen.be. The code for our proposed algorithm is available at: https://github.com/alhussein1977/IIMFIs-Algorithm."

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this study.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 207–216, 1993.

[2] W. Kreesuradej and W. Kreesuradej, "Incremental association rule mining with a fast incremental updating frequent pattern growth algorithm," *IEEE Access*, vol. 9, pp. 55726–55741, 2021.

[3] J. Han, M. Kamber, and J. Pei, "6-mining frequent patterns, associations, and correlations: basic concepts and methods," in *In Data Mining (Third Edn)*, J. Han, M. Kamber, and J. Pei, Eds., Morgan Kaufmann, Boston, MA, USA, pp. 243–278, 2012.

[4] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International*

Conference on Very Large Data Bases, pp. 487–499, Santiago, Chile, September 1994.

[5] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 1–12, May 2000.

[6] H. Liu, L. Cui, X. Ma, and C. Wu, "Frequent itemset mining of user's multi-attribute under local differential privacy," *Computers, Materials & Continua*, vol. 65, no. 1, pp. 369–385, 2020.

[7] Z. H. Lv and S. L. Lv, "PrePost+: an efficient N-lists-based algorithm for mining frequent itemsets via Children-Parent Equivalence pruning," *Expert Systems with Applications*, vol. 42, no. 13, pp. 5424–5432, 2015.

[8] Z. H. Lv and S. L. Lv, "Fast mining frequent itemsets using Nodesets," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4505–4512, 2014.

[9] Z. H. Deng, "DiffNodesets: an efficient structure for fast mining frequent itemsets," *Applied Soft Computing*, vol. 41, pp. 214–223, 2016.

[10] N. Aryabarzan, B. Minaei-Bidgoli, and M. Teshnehlab, "negFIN: an efficient algorithm for fast mining frequent itemsets," *Expert Systems with Applications*, vol. 105, pp. 129–143, 2018.

[11] Y. Xun, X. Cui, J. Zhang, and Q. Yin, "Incremental frequent itemsets mining based on frequent pattern tree and multi-scale," *Expert Systems with Applications*, vol. 163, Article ID 113805, 2021.

[12] M. J. Zaki and C. J. Hsiao, "CHARM: an efficient algorithm for closed itemset mining," in *Proceedings of the 2002 SIAM International Conference on Data Mining*, pp. 457–473, VA, USA, April 2002.

[13] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," *Lecture Notes in Computer Science*, vol. 1540, pp. 398–416, 1999.

[14] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," in *Proceedings of the Workshop on Frequent Itemset Mining Implementations (FIMI '03)*, pp. 123–132, Melbourne, Florida, USA, December 2003.

[15] J. Wang, J. Han, and J. Pei, "CLOSET+: searching for the best strategies for mining frequent closed itemsets," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pp. 236–245, Washington, DC, USA, August 2003.

[16] N. Minaei-Bidgoli and B. Minaei-Bidgoli, "NEclatClosed: a vertical algorithm for mining frequent closed itemsets," *Expert Systems with Applications*, vol. 174, Article ID 114738, 2021.

[17] U. Lee and G. Lee, "Incremental mining of weighted maximal frequent itemsets from dynamic databases," *Expert Systems with Applications*, vol. 54, pp. 304–327, 2016.

[18] H. Shahraki and M. H. N. Shahraki, "Incremental mining maximal frequent patterns from univariate uncertain data," *Knowledge-Based Systems*, vol. 152, pp. 40–50, 2018.

[19] A. Bai, M. Dhabu, V. Jagtap, and P. S. Deshpande, "An efficient approach based on selective partitioning for maximal frequent itemsets mining," *Sādhanā*, vol. 44, no. 8, p. 183, 2019.

[20] D. Burdick, M. Calimlim, and J. Gehrke, "Mafia: a maximal frequent itemset algorithm for transactional databases," in *Proceedings of the 17th International Conference on Data Engineering IEEE*, pp. 443–452, IEEE Press, Heidelberg, Germany, April 2001.

[21] K. Zaki and M. J. Zaki, "GenMax: an efficient algorithm for mining maximal frequent itemsets," *Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 223–242, 2005.

[22] G. Grahne and J. Zhu, "High performance mining of maximal frequent itemsets," *6th International Workshop on High Performance Data Mining*, vol. 1, pp. 135–143, 2003.

[23] G. Zhu and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347–1362, 2005.

[24] M. J. Hsiao and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 462–478, 2005.

[25] X. Zeng, J. Pei, K. Wang, and J. Li, "PADS: a simple yet effective pattern-aware dynamic search method for fast maximal frequent pattern mining," *Knowledge and Information Systems*, vol. 20, no. 3, pp. 375–391, 2009.

[26] S. M. Fatemi, S. M. Hosseini, A. Kamandi, and M. Shabankhah, "CL-MAX: a clustering-based approximation algorithm for mining maximal frequent itemsets," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 2, pp. 365–383, 2021.

[27] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, "Maintenance of discovered association rules in large databases: an incremental updating technique," in *Proceedings of the Twelfth International Conference on Data Engineering*, pp. 106–114, IEEE, New Orleans, LA, USA, February 1996.

[28] T. P. Hong, C. W. Lin, and Y. L. Wu, "Incrementally fast updated frequent pattern trees☆," *Expert Systems with Applications*, vol. 34, no. 4, pp. 2424–2435, 2008.

[29] T. P. Hong, C.-W. Lin, and Y. L. Wu, "Maintenance of fast updated frequent pattern trees for record deletion," *Computational Statistics & Data Analysis*, vol. 53, no. 7, pp. 2485–2499, 2009.

[30] J. Sun, Y. Xun, J. Zhang, and J. Li, "Incremental frequent itemsets mining with FCFP tree," *IEEE Access*, vol. 7, pp. 136511–136524, 2019.

[31] W. Kreesuradej and W. Thurachon, "Discovery of incremental association rules based on a new FP-growth algorithm," in *Proceedings of the 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pp. 184–188, IEEE, Singapore, September 2019.

[32] T. P. Hong, C. Y. Wang, and Y. H. Tao, "A new incremental data mining algorithm using pre-large itemsets1," *Intelligent Data Analysis*, vol. 5, no. 2, pp. 111–129, 2001.

[33] C. W. Lin, T. P. Hong, and W. H. Lu, "The Pre-FUFP algorithm for incremental mining," *Expert Systems with Applications*, vol. 36, no. 5, pp. 9498–9505, 2009.

[34] A. S. Al-Hegami, V. Bhatnagar, and N. Kumar, "Novelty framework for knowledge discovery in databases," in *Data Warehousing and Knowledge Discovery*vol. 3181, , pp. 48–57, Springer, 2004.

[35] V. Bhatnagar, A. S. Al-Hegami, and N. Kumar, "A hybrid approach for quantification of novelty in rule discovery," in *Proceedings of the Second World Enformatika Conference, WEC'05*, no. 2, pp. 39–42, Istanbul, Turkey, February 2005.

[36] V. Bhatnagar, A. S. Al-Hegami, and N. Kumar, "Novelty as a measure of interestingness in knowledge discovery," *International Journal of Information Technology*, vol. 2, no. 1, pp. 36–41, 2005.

[37] A. S. Alsaeedi and H. A. Alsaeedi, "A framework for incremental parallel mining of interesting association patterns for big data," *International Journal of Computing*, vol. 19, no. 1, pp. 106–117, 2020.

[38] J. Xu and L. Xu, "A novel interestingness measure based on fusion model for association rules mining," *MATEC Web of Conferences*, EDP Sciences, vol. 336, , Article ID 05009, 2021.

[39] B. Tuzhilin and A. Tuzhilin, "Unexpectedness as a measure of interestingness in knowledge discovery," *Decision Support Systems*, vol. 27, no. 3, pp. 303–318, 1999.

[40] B. Bing Liu, W. Wynne Hsu, S. Shu Chen, and Y. Yiming Ma, "Analyzing the subjective interestingness of association rules," *IEEE Intelligent Systems*, vol. 15, no. 5, pp. 47–55, 2000.

[41] T. Li, Y. Ren, Y. Ren, and J. Xia, "An improved algorithm for mining correlation item pairs," *Computers, Materials & Continua*, vol. 65, no. 1, pp. 337–354, 2020.

[42] H. Nam, U. Yun, E. Yoon, and J. C. W. Lin, "Efficient approach for incremental weighted erasable pattern mining with list structure," *Expert Systems with Applications*, vol. 143, Article ID 113087, 2020.

[43] G. Yun and U. Yun, "Single-pass based efficient erasable pattern mining using list data structure on dynamic incremental databases," *Future Generation Computer Systems*, vol. 80, pp. 12–28, 2018.

[44] U. Yun, G. Lee, and E. Yoon, "Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields," *Information Sciences*, vol. 494, pp. 37–59, 2019.

[45] T. Vo and B. Vo, "MEI: an efficient algorithm for mining erasable itemsets," *Engineering Applications of Artificial Intelligence*, vol. 27, pp. 155–166, 2014.

[46] J. C. W. Lin, M. Pirouz, Y. Djenouri, C. F. Cheng, and U. Ahmed, "Incrementally updating the high average-utility patterns with pre-large concept," *Applied Intelligence*, vol. 50, no. 11, pp. 3788–3807, 2020.

[47] J. C. W. Lin, W. Gan, T. P. Hong, and B. Zhang, "An incremental high-utility mining algorithm with transaction insertion," *The Scientific World Journal*, vol. 2015, Article ID 161564, 15 pages, 2015.

[48] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, T. P. Hong, and H. Fujita, "A survey of incremental high-utility itemset mining," *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 2, 2018.

[49] R. V. Priya, A. A.Vadivel, and R. S. Thakur, "Maximal pattern mining using fast CP-tree for knowledge discovery," *International Journal of Information Systems and Social Change*, vol. 3, no. 1, pp. 56–74, 2012.

[50] Q. Niu, "Optimization of teaching management system based on association rules algorithm," *Complexity*, vol. 2021, Article ID 6688463, 13 pages, 2021.