Mining Frequent Item Sets in Asynchronous Transactional Data Streams over Time Sensitive Sliding Windows Model

QAISAR JAVAID*, FARIDA MEMON**, SHAHNAWAZ TALPUR***, MUHAMMAD ARIF****, AND MUHAMMAD DAUD AWAN*****

RECEIVED ON 11.03.2016 ACCEPTED ON 11.05.2016

ABSTRACT

EPs (Extracting Frequent Patterns) from the continuous transactional data streams is a challenging and critical task in some of the applications, such as web mining, data analysis and retail market, prediction and network monitoring, or analysis of stock market exchange data. Many algorithms have been developed previously for mining FPs (Frequent Patterns) from a data stream. Such algorithms are currently highly required to develop new solutions and approaches to the precise handling of data streams. New techniques, solutions, or approaches are developed to address unbounded, ordered, and continuous sequences of data and for the generation of data at a rapid speed from data streams. Hence, extracting FPs using fresh or recent data involves the high-level analysis of data streams. We have suggested an efficient technique for the window sliding model; this technique extracts new and fresh FPs from high-speed data streams. In this study, a CPILT (Compacted Tree Compact Pattern Tree) is developed to capture the latest contents in the stream and to efficiently remove outdated contents from the data stream. The main concept introduced in this work on CPILT is the dynamic restructuring of a tree, which is helpful in producing a compacted tree and the frequency descending structure of a tree on runtime. With the help of the mining technique of FP growth, a complete list of new and fresh FPs is obtained from a CPILT using an existing window. The memory usage and time complexity of the latest FPs in high-speed data streams can efficiently be determined through proper experimentation and analysis.

Key Words: Data Mining, Data Stream, Frequent Pattern, Transactional Data, Sliding Windows.

1. INTRODUCTION

Data streams are realtime, continuous, possibly infinite, fast, changing, and ordered, with a huge amount of sequences of items [1,2]. A data stream rapidly changes with time, so that acquiring all the elements in it is impossible. Each element in it is examined once for a time. Given that new data elements are constantly and continuously engendered in a stream, the memory usage for stream mining should be limited [3]. Stream mining must ensure that a new data stream be available immediately whenever a request is made for such stream [4]. This requirement makes the task more challenging in some applications, such as knowledge

* Department of Computer Science & Software Engineering, International Islamic University, Islamabad.
** Department of Electronic Engineering, Mehran University of Engineering & Technology, Jamshoro.
*** Department of Computer Systems Engineering, Mehran University of Engineering & Technology, Jamshoro.
**** Department of Computer Science, University of Gujrat, Gujrat.
***** Faculty of Computer Science, Preston University, Islamabad.

discovery, fraud detection, and business improvement. FPs (Frequent Patterns) have been extracted from large data sets by many researchers using different techniques. Many aspects maybe related to discovering FPs from large data sets. However, the major aspects that are relevant to recurrent pattern finding are storage and run time. Hence, researchers focus on finding FPs that take less time and storage. In the present study, various algorithms for finding FPs in large data sets are discussed [4]. Sequential patterns are basically found from continuous data streams and from transitional and normal data sets. The execution times of different algorithms for FP mining (FPM) are compared in this work. Based on speed, the performances of different FP algorithms are also compared. FPs are also found in real time to show the difference from an ordinary system[4].

The data collected from various sources, such as sensor data and weather or satellite data, are basically huge and inexact. Database sizes are growing rapidly, and such databases may be used for knowledge discovery that requires les storage and time. Different objects may have various relationships with one another, which may lead to association rules in diverse databases. Different patterns of objects are discovered through such types of relationships among these objects. Such type of pattern matching can be utilized indifferent applications of decision support and weather forecasting. Association rules [5] may contain FPM as sub-problem and can be utilized to search out the frequent items from the large databases. Association rules are induced from the concept of market basket analysis, which is mostly used in identifying customer behavior with respect to purchasing different products from the market. For example, if a child purchases a book, then the child is most likely to purchase a pen also. When FPs become exponentially large, a major problem arises in FPM. Finding interesting patterns in exponentially large FPs become a problem, and pruning unimportant patterns from large patterns becomes important in FPM. Thus, finding interesting FPs from data streams is the end result of FPM. Data streaming is characterized by continuous, unbound, and high-speed data. Data stream sources are several distributed areas from which data streams emerge. Hence, active storage is insufficient to store all stream-related data. Stream-related data arrive newly with the advancement of time. For such data, scanning is necessary only once, thereby consuming limited storage type and responding in real time. Finding frequent item sets in several applications is becoming important because of the significant increase in data streams. At present, the data stream mining field is becoming a challenge in some applications, such as fraud detection, KKD, online transaction mining, trend learning, estimation and transaction prediction, and analyzing different item sets. Moreover, given the continuous, high-speed, and unbounded features, data stream mining has become difficult. In algorithm [1], FPM is likely to be close to the proposed system, which is generally used to extract FPs using a specific data stream. In FPM, a SW (Sliding Window) mechanism is used, based on which the parts of the window are made by dividing the windows into parts of equal size in fixed numbers that contain transactions with non-overlapping batches. The prefix tree structure [6] for canonical order is mostly used to store information in the current window. For every batch, every node of most trees maintains a list that stores the frequency count. Tree traversal can be avoided to extract information from a tree in an old batch. Therefore, the tracks of the last visited batches are maintained via FPM, and an extra pointer is used for every node to count the last restructured batch number. Frequency list contents are changed for SW reflection using nodes. The FP-growth technique is utilized for mining when the information is captured using SW [7], that is, the FPs from a complete information sets. However, FPM has several limitations. First, information or item sets are stored in the canonical order, and using

FPM for structures related to a highly compacted tree provides no guarantee, which is quite important in manage data streaming to avoid the overhead related to massive storage, decrease search space, and ultimately hasten FP growth-based frequent pattern mining operations. Secondly, the lists of frequency counts in FPM [1] are used to store information for a batch that considers each node; with the usage of such types of lists, the tree size increases considerably. Third, another issue in FPM is related to storage overhead, where for every node, FPM uses extra batch pointers that indicate the last visited batch related to such node. Fourth, during tree updating, FPM cannot visit all the nodes related to a tree, and FPM practically cannot perform process related to shift frequency count lists for each node. Hence, the updating process for a frequency list related to specified nodes is not performed, given new incoming batches that are not visited. For the current window, FPM may leave some invalid nodes during updating. Hence, for mining, FPM consumes more time than such types of trees based on structures organized in such a way that the frequency depends on the order of items. Moreover, FPM construction is totally based on an assumption that does not consider the limitations of the main memory, which is unrealistic in considering or processing huge amounts of data, such as a data stream. By contrast, CPILT, the proposed tree in this study, exactly provides similar information on data streams and performs similarly to FPM, such action with the storage only in an FP tree with a strong compact structure, thereby presenting an efficient data structure for strong storage. Most proficient FP growth-based mining platforms are provided by a highly compacted tree structure. Moreover, efficiency is achieved in a path that presents a transaction by maintaining only the frequency count list for the last node rather than maintaining specific information for every node. Furthermore, an extra patch pointer is not required by CPILT for every node to maintain the last updated track. CPILT is regularly updated with the mechanism by extracting such type of transactions that expire after every window slide. This feature guarantees that garbage nodes do not exist in a tree and that a clear tree status is ensured for mining.

Time and memory efficiency for FPM in a distributed transactional data stream results in accuracy. Handling the continuous flow of stream in data stream mining is related to the issue of database management. Traditional database management systems are insufficient to handle such high data rates. Efficient indexing and novel techniques of querying and storage can be utilized to handle the problem of fluctuation in the flow of streams containing information. Algorithms of different sorts are proposed to tackle this issue. Techniques designed for time and space efficiency should be accompanied and complemented with excellent accuracy in terms of results.

2. RELATED WORK

Data streams are realtime, continuous, possibly infinite, fast changing, and ordered, with huge amounts of sequence items [8]. A data stream rapidly changes with time, so that acquiring all the elements it contains is impossible [9]. As such, each element in a data stream is examined individually [10]. Given that new data elements are produced continuously via streaming, the memory usage for stream mining should be limited [3]. In stream mining, a new data stream should be guaranteed to be available immediately whenever a request is made for such stream. This requirement makes data streaming more challenging in applications, such as knowledge discovery, fraud detection, and business improvement [11]. To mine frequent item sets, many authors have suggested different techniques. Ran, et. al. [12] have suggested the “lossy weight algorithm”, which is mostly used in finding frequent item sets based on weight. The “SW” approach has also been proposed [4,13] and is mostly used in data stream mining. This approach is subdivided in to two types: the "transaction-sensitive" SW and the "time-sensitive" SW. Jiayin, et. al. [14] proposed using this algorithm in mining frequent item sets in SW; they effectively proposed a new approach to extracting frequent item sets using the SW technique.

In the past decades, static mining FP [6,12] and incremental databases [15] have been addressed in an excellent manner. A parallel a prior algorithm is used for the FP algorithm [16] to find association rules [5] among patterns. This technique is used to find FPs with *k* length from a set of already created candidate patterns. The main limitations of the performance of a priori-like methodologies require multiple database scans for favorable results, and many candidate patterns are proven to be infrequent after database scanning. To minimize this problem, Han et al. suggested an frequent pattern tree and an FP-growth-based algorithm [7]. The passes of database scan will be reduced in two pass by using FP-growth tree used in [6], and the candidate generation requirement can be eliminated. Introducing such highly compact structure for an FP tree has introduced in turn a new research method for mining FPs with the structure of prefix tree [6]. General and research issues related to mining frequent patterns in data streams are revised [15,17]. The scope of the present work focuses on data stream mining using the SW mechanism [18], but the literature review here mainly focuses on studies related to window-based methods. Most studies focus on models of landmark window [17] and SW [15], which are mainly used to find FPs in a data stream. In [19] represents the first attempt to mine FP from the entire history related to data streaming. Lossy counting [8] and sticky sampling are single-pass algorithms developed based on the anti-monotone property. These two algorithms, which have some error bound, deliver approximate results. In this paper the [20] uses a lattice structure, which refers to a frequently enumerated tree divided into several equal stored pattern classes with the same transaction ID within a single class. Another algorithm [21]was implemented to find FPs using a landmarks window model. Each transactions are divided into small *k* transactions inserted in the summary data structure of an extended prefix tree called the frequent item set forest. FP growth based tree called FP stream mining was developed by Borgelt, et. al. [7] to mine FPs using the tilted time window technique. As a new batch of transactions arrives, the algorithm uses the FP growth technique to process the data stream. Such batch-by-batch processing technique has a major limitation in handling the stream flow, given that an FP stream requires constructing the FP tree to attain the streaming items in every newest batch. The SW technique in relation to mining FPs has also addressed in the literature. The data stream mining methods for FP are presented by Lin, et. al. [19], using time-sensitive SW; that is, the window size is determined with a fixed time period. Through this approach within the time period of a window, the landing stream is divided into many batches, and mining FPs is performed individually in every batch. A discounting mechanism is used, and the method removes old information using the approximate table, which provide the approximated counts for the expired and old informations. The SW technique [13] is used to adaptively find FPs for online transactional data stream. Such algorithm requires minimum threshold support from every window and significant support to adaptively maintain approximate FPs. Most of the above-mentioned methods are used to approximately find FPs [22] with some error bound or threshold for additional pruning. Some techniques [23-26] are used to find the exact set of FPs using the data stream. Through the apriori-based MFI-Trans SW algorithm [27], a complete set of FPs is found using the bit sequence that track the incidence of all the item sets against the data transactions related to the current fixe size SW. Left-shift bit-wise operation is performed against all bit sequences to remove the old information and store the newest one. The test methodology and candidate generation with level-wise are applied through MFI-Trans SW to find FPs using the current window. Thus, this algorithm suffers from a priori limitation in creating huge candidate patterns, especially mining such type of data stream that contain many more and/or long FPs and/or that have a low threshold support. Moreover, the transaction-by-transaction updating mechanism may hinder the performance of the algorithm as the stream speed increases. Given thither bit sequence information on all the window items is maintained through this approach, efficiency in terms of memory cannot be achieved, particularly when distinct items and numerous transactions exist. MFI-Trans SW is used to discover FPs from a datastream, but this algorithm significantly differs from the suggested technique in terms of both data processing and mining approach strategies. FMP [1] is mostly close to the proposed algorithm, which exactly finds FPs from a data stream. The SW technique is used in FMP, and the division of the window is formulated such that a window consists of equalized and non-overlapping fixed numbers of transaction batches. To store the current window information, a prefix-tree structure in canonical order is used. A list is maintained in each tree node to explicitly store the frequency count against each batch. At each node, FPM contains a track for the last visited batch, and an additional pointer is used for the last updated batch number. To reflect the SW frequency list, the contents are shifted in the nodes. When information is captured for a full window, the request of the mining FP growth-based approach is used to mine a complete set of recurrent patterns through a tree.

3. MATERIALS AND METHOD

3.1 Compact Pattern Item List Tree

The definitions of the basic terminologies used to clarify the FP idea in streams are presented in this section. Suppose $I= \{i1; i_2;...; i_n\}$ is a set of items that are also recognized as literals; in some application fields these items are used as parts of the information. Set $C=\{i_l; ... ; i_k\}$ *†"I,ld"k* and *l " [1,n]* is called an item set/patterns or *q*-item sets, if the set consists of*q* items. *t = (tid)* in transactions ID, which is tuple, whereas *D* is used for the patterns and (*tid*) is used for a transactions ID. Suppose *C †"D,t* can be assumed to consist of *C*; therefore, *C* may fall under *t* transaction. The item numbers in *D* suppose the size of *t* (transaction *t*). Formally, a data stream *DS* can be well-defined in the limitless arrangements of the transactions, given the streaming of data items= $[t_1; t_2; ... ; t_m]$, where *ti; i" [1; m]* reaches i^{th} transaction. Set for all arrival transaction IDs among the i^{th} and j^{th} transaction (*where j>i*) may be referred to as window *W*, and the window size can be represented as |*W*| = *j _ i*, which shows the total number of transactions between the i^{th} and j^{th} arrival transaction. If window *W* consists of the same size and equal number of non-overlapping bunches of transactions, the it is termed as plates. Suppose window *W* consists of the same size and equal numbers of plates. Furthermore, let window *W* contain *m and n* transaction and plates respectively (*m mod n=0*), then *m/n* transactions are found in a plate; thus, the plate size would become |m/n|. Suppose a window is slid plate by plate ,then each slides of window shows a newest plate, and the existing old plate in the present window is removed. Support (Sup)*W(C)* represents the *C* pattern support in *W* window, which shows the transactions number in window *W* that consists of *C* patterns. Hence, a pattern in window *W* is an FP if support á is not less than the absolute minimum á threshold min sup á as *0 d" á d"* |*W*|, which is supposed by a user. Given *DS* |*W*|, a min-sup and the complete set of patterns in window *W* reveal a support count of not less than sup *á* value (i.e. minimal support). The problem is to mine a precise set of regular current patterns; the FW (Frequent Window) in the data streams; thus, the SW mechanism is utilized.

3.2 Designing and Constructing CPILT

The mechanism of efficiently extracting plates is explained using CPILT. Through the methods of dynamic tree rearrangement, a tree structure for frequency descending is utilized; performing this task shows how CPILT periodically reorganizes itself. Here, mining performance analysis is also noticed when CPILT achieves its goal using its mechanism for dynamic tree rearrangement. Only one scan for a database is required to mine FPs using the CPILT structure. At the beginning, the transactions from a data stream are placed in CPILT (in lexicographical order) using a predefined order of items. A list is utilized to maintain the order of the items in CPILT; such type of list is called an F-list, and it contains the frequency count of the items. When transactions from a data stream with some numbers are placed and the order of the F-list changes considerably compared with the existing frequency-descending item order, the CPILT is readjusted dynamically, considering that the existing frequency descending item order and the order of the items are updated in the F-list with respect to the current one. As described earlier, CPILT is framed to mine FPs from a data stream. SW can be divided into non-overlapping plates with the same size, and one plate can be utilized for the transactions of every batch in a data stream. Information on the plates are maintained separately in CPILT. The update process of a tree is performed with the removal of the expired transactions of the plates when window slides and transactions are simultaneously placed in a new arriving plate.

A brief description of the CPILT structure is required before elaborating the erection process of CPILT. CPILT consists of one node as the root node, which is referred to as the null node. The children of root node can be labeled as item set of prefix sub-trees, and F-list is a exclusive item set that has a relative frequency as compared with F-lists and a pointer, that shows the first node in the CPILT that has item sets. Similar to the FP growth-based tree, CPILT tree structure also bears the nodes that present an item set with the limited numbers of badges (i.e. support and passes) for such sets of items for the present window, which is root of the node in the CPILT. A novel idea is used to form an F-list for a transaction. For such purpose, information on the plate-based supports count is considered only for tail items. The tail item is explained below.

3.2.1 Definition-1: Tail Item

Let $\{i_1; i_2; ... ; i_n\}$; which is, DS data streaming transaction contain item sets in the form of a pre explained, sorted orders and in the organized manners. The item i_n, is to be fined in the last of the transaction, and it is called the tail item of the transaction. For example, when a transaction is sorted lexicographically,$\{a,b,c\}$ in this transaction c is the tail item and when this transaction is reversed the item a will become the tail item. Two kinds of nodes can be maintained through CPILT.

3.2.2 Definition-2: Tail Node

Suppose $t = \{i_1; i_2; ... ; i_n\}$ represents the arranged transaction, and i_n is called the tail item. If the lexicographically given transaction t $\{a; b; c\}$ is insert into the CPILT, then c node in the CPILT becomes the tail node.

3.2.3 CPILT Organization

To maintaining the order of frequency in descending way, CPILT is restructured by itself dynamically. The CPILT construction process has two phases: insertion and rearrangement. The stream contents are captured in the insertion phase in a tree with respect to an F-list sorting order, and the current sorting order is the F-list sorting order. The rearrangement phase is utilized to restructure a tree in frequency descending order with the help of already obtained information. During the process of tree construction, both phases are executed repeatedly many times. Given that the data stream has a dynamic nature, the transition from the insertion to the rearrangement phase can be executed when the plate information is captured dynamically. CPILT formation from a data stream can be explained through an example. In **Fig. 1**, the step-by-step formation methodology of CPILT is shown with the same data stream and related transaction IDs.

Given the example under explanation, we assume that after inserting each plate, the rearrangement of the tree is performed. We discuss several types of rearrangement criteria that may be useful for applications for data streams with a dynamic nature. We also assume that tree formation starts with the insertion of the first-plate transactions in such type of items containing a predefined order, that is, the lexicographical order of the items. CPILT is initially empty (i.e. CPILT is initialized with a root node with a null value), as shown in Fig 1. In our method, the FP-tree construction methodology is adopted to insert the transactions in a sorted order in the CPILT. Pointers for the node traversal are not shown, but they persist in same way as in FP growth tree. Symbolically, I sort is used for the frequency descending F-list, and I is used for frequency independent. The first phase consists of the construction of the CPILT insertion. In the insertion phase, the inserting process starts with insertion of the 1st *{a,c,d,e}* and 2nd *{a,b,c,d}* transactions in plate one in the lexicographical order of the item sets. Exact formation of CPILT and F-list after the insertion of plate 1is explained in Fig.1. Regarding the two transactions for plate 1,*e* and *d* are the two tail items by maintaining the lexicographical order of the items. Hence, the two tail nodes *"e:1;1,0"* and *"d:1;1,0"*maintain the plate counters and the outstanding nodes, which are the ordinary nodes that preserve total support count for the path.

Two plates exist in a window, and the plate counter registers two count values in each plate and tail node. A supporting value exists for the first fields of both plate counters of plate 1, and the other fields relating to the list start with 0. The total count is distinctly measured by the tail node to the related path. Tree rearrangement is performed after each plate; the insertion phase ends here The rearrangement phase then starts, during which the preliminary item sets are not introduced or placed in order relating to the decending order of frequency. CPILT becomes frequency autonomous, with a lexicographical order of items. To obtain 1 sort, the tree formation is reorganized. The item order relating to 1 is changed into frequency descending order. The tree is then rearranged with respect to 1 sort. The F-list with the changes and rearranged CPILT after the first rearrangement phase are shown in Fig. 1. The rearrangement operation may change tail node into ordinary node and ordinary to tail node. Variations may occur in the tree, but the node status in CPILT does not affect any property relating to the formation of CPILT.

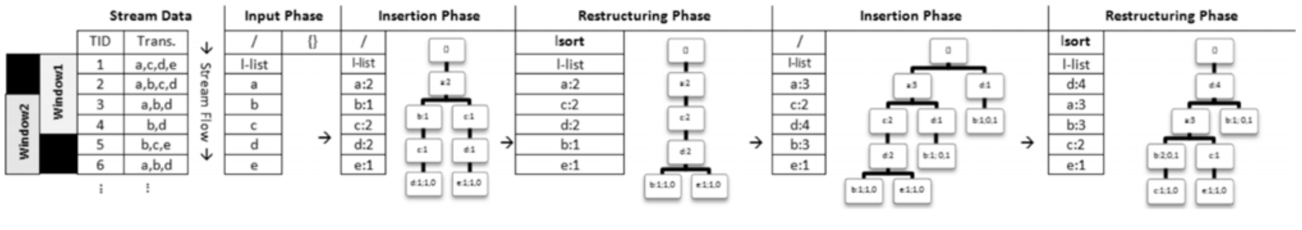


FIG. 1. ORGANIZATION OF CPILT AT WINDOW-1

During the construction of CPILT, two properties are observed.

Property-1: The total count of the frequency in CPILT e" summation of total count of frequency of the children.

Property-2: The total count of the frequency of tail nodes in CPILT > summation of the total frequency of the children at the end of the tree.

In window sliding, CPILT is modified with the exclusion of old plate information and insertion of new plate information.

3.2.4 Extraction of Old Information

To assemble a platform that is set to be mine, the configuration of the CPILT is updated with precise contents in the current window. When the windows are made to slide, CPILT is modified by traversing the entire tree. The plate counter for each tail node is updated to delete old and expired information that contains CPILT. Changes may be reflected if necessary at the remaining nodes in the corresponding path, as the tail nodes maintain the plate information. Fig. 2 shows the basic mechanism of extracting old information from plates in CPILT and the refreshing algorithm for such purpose. For each tail node, when the update operation for the plate counter starts, the refreshing operation for CPILT begins as well. This operation starts when the lowest items exists in F-list. The 1st value in the plate counters is removed relating to each tail node of item. The oldest plate expiration is shown by shifting the remaining values by one slot to the left in the list. When the update is performed, any tail node becomes an ordinary node, and zeroes exist in its plate counter for all entries. When the total count for a node becomes zero, the deletion operation for that node is performed. In the same manner, the deletion operation for any node can be performed from CPILT when the support for any ordinary node becomes zero.

In the tree update process, no operation is performed in CPILT for an ordinary node. The mechanism for handling expired plate for CPILT is considered. For Window-1, the stream data and CPILT construction in Fig. 2 are considered. When the window sliding operation starts, we suppose that the oldest plate (i.e. *tid1* and *tid2*) expire and that a newest plate (*tid* 5 and *tid* 6) become visible. A list is utilized to facilitate the traversal of the fast tree and minimization of the number of visited nodes; the bottom of the list is used for the start. At the bottom of the F-list, most *"e"* items are visited and processed one by one for the tree update process. We admittance *"e: 1; 1, 0,"* which is an accessible node in the CPILT and the tail nodes. The value present in 1st slot of the plate counters of node (1) is removed, and alike values related to the node decrease with respect to the total count (1), thereby resulting in zeroes in all the node records. As such, the deletion operation is completed for the *"e"* node of the tree.

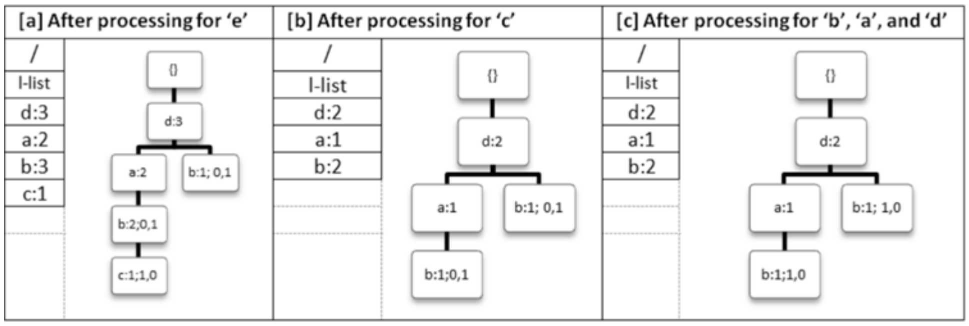


FIG. 2. EXTRACTING OLD PLATE INFORMATION FROM CPILT

Same value is decremented from the support count value related to every node that starts from this to root node. The subsequent immediate node to root node in path, that is,*"c:1,"* is further processed. The deletion operation for this node is performed when the total support count for this node is zero. For the outstanding nodes *(*i.e. *items"a" and "d")*, the update procedure is executed consequently. The F-list is updated by following every update process, and the changes are saved. The update operation for item *"e"* is terminated with the completion of the adjustment for the F-list when no more nodes belong to *"e"* in the tree. The subsequent CPILT is presented in Fig. 5. "c" is the subsequent node in the tree containing a particular node *"c: 1; 1, 0,"*which is also called the tail node. Therefore, the same method accepted for the *"e"* in tree is also accepted for the update process. *"b: 1; 0, 1"* is the first node of *"b"* that contains the subsequent item and is also a tail node. Fig. 2 shows that the first value of the plate counter is zero. The transactions in the 1st plate that are shown by such type of tail node do not appeared. Hence, the total counts of the value in this node and remaining nodes toward the root node in the path of the update process is not needed.

The modifying method for the counters of the plates are executed by shifting the corresponding values to left, and a 0 is placed at end to store the upcoming plates for the new information. The total count for the contents and plate counter of node are*1* and *1, 0*, respectively, after the process is completed, as shown in Fig. 2. The subsequent node containing the item *"b"* is also a tail node that has the same information as the previous one. Hence, the same procedure is adopted. Item "a" consists of a single node *"a: 1,"* which is the subsequent item in the tree and is an ordinary node. An operation need not be performed for such node. After skipping "a," we achieve a single ordinary node *"d: 2"* in the tree. Similar to the previous item, the node is also skipped, and the update process for CPILT is terminated because the F-list contains no remaining items. The final CPILT in Fig. 2 is shown after plate 1 extraction, and this tree is prepared similar to an updated tree that is ready for capturing information on the upcoming new plate according to the mechanism of CPILT formation.

The formation of CPILT when new information on the plate included is presented in Fig. 2 (*tid 5 tid 6*) are inserted in the CPILT reorganization in Fig. 2. The tree is reorganized after the insertion of new plate information infrequency descending order, as shown in Fig. 2. Considering the plate extraction methods and the construction of CPILT.

Depending on FPM and the process of CPILT formation, the storage efficiency that CPILT can obtain is due to (1) tree formation with high compactness, (2) conservation of the plate counters based on the tail node (i.e. avoidance of the "garbage" node), and (3) obtaining liberation for pointers regarding the last plate of every node. The frequency descending order of CPILT, which is rearranged dynamically, allows as much as possible prefix allocation in the concerning transactions in the current window.

3.3 Mining Frequent Patterns FPs

A methodology of pattern growth can be utilized to produce extremely dense CPILT that facilitates and ensures frequent item sets. Similar to the FP growth [7] mining approach, we mine CPILT repeatedly to produce FPs through the PBs (Provisional Pattern Bases) and CTs (Conditional Trees) for the further scanning of records.

The basic operations relating to FPM using CPILT are as follows: (i) counting the length-1 of the recurrent objects, (ii) making the provisional PB for the each prearranged object (iii) making the conditional patterns for each provisional PB. The recurrent patterns at an occasion is created using CT. Regularity-engendered FPs are checked to find regular FPs.

Property-2: The TID list keep the occurred data against each node in CPILT alongside the route not less than for the list traversal. Let $X = \{b_1, b_2, \ldots, b_n\}$ represent a route within CPILT and b_n represent the tail node that carries the *TID* list for the route. When the *TID* list is moved up to the bn-1 node, the bn-1 node contains the incident data against route *X = {b1, b2,…,bn-1}*.

Proof: According to property-2, the incident data of route *Z* are kept by the node bn in the *TID* list representing the minimum traversals that it covers. Thus, traversing data identical to *Z* minus any failure can be conserved precisely in the bn-1 node in a similar *TID* list. The lowest extreme in the F-list can be used to create a provisional PB. A small F-list for the entry is also created at the time of creating a provisional PB. In our explanation, *"D"* becomes the nethermost entry in the F-list. Fig. 3(a) shows a conditional PB tree for "D." With respect to lemma 2, when the *TID* list for *D* is strapped up in relation to its parent nodes *A, C, and E*, each parent node of D is changed into a tail node. The immediate FP for D is (D:1, 6, 7, 8, 9),that is, D occurs in 1,6,7,8, and 9 transactions, and its support is 5.Four routes exist for CPILT: *(A:1, D:1), (B, C, D:6, 9), (E, C, D:7), and (E, D:8)*. The number after ":" shows every occurring sub pattern tid. The conditional PB for *D {(A:1), (B, C:6,9), (E, C:7),* and *(E:8)}* does not create frequent items. The mining for D is terminated, and D is the only1 FP. We then check the regularity for pattern D. If we know the occurring transactions for D, then we can easily calculate the regularity of D according to definition 1 [25], that is, 2.54. According to this example, the threshold value of the regularity is set as*max_variance = 1.0*. Given*Reg (D) > 1.0*, *D* does not become a regular pattern. The immediate FP for *C*is*(C: 2, 4, 6, 7, 9)*, with tree routes of *(B, E, A, C:2, 4), (B, C:6, 9), and (E, C:7)*. The conditional PB for *C* is *{(B, E, A:2, 4), (B:6, 9), (E:7)}*, as shown in Fig. 3(b). The CT for C leads one branch *(B: 2, 4, 6, 9)*, and the engendered frequent patterns are *(BC:2, 4, 6, 9)* and *(C:2, 4, 6, 7, 9)*.We calculate the regularity of *BC* and *C* using definition 1, and the respective values of their regularities are*0.96 and 0.583*. Given that *{Reg (BC), Reg(C)} < 1.0*, BC and C become regular FPs. The immediate FP for node *A* is *(A:1, 2, 3, 4, 5)*, and its conditional PB is *(B, E:2, 3, 4, 5)*, as shown in **Fig. 3(c).** The CT for A leads one branch *(B, E:2, 3, 4, 5)*, and the engendered FPs are*(AB:2, 3, 4, 5), (AE:2, 3, 4, 5), (ABE:2, 3, 4, 5), and (A:1, 2, 3, 4, 5)*. The respective values of the regularity of the patterns are 1*.36, 1.36, 1.36, and 1.25*. With *{Reg(AB), Reg(AE), Reg(ABE), Reg(A)} > 1.0*, such patterns do not become regularly frequent. For node E, (E:2, 3, 4, 5, 7, 8) is derived, and its conditional PB is *(B:2, 3, 4, 5)*. CT for E leads one branch *(B:2, 3, 4, 5)*, and the engendered FPs for it are *(BE:2, 3, 4, 5) and (E:2, 3, 4, 5, 7, 8)*. The corresponding regularity is*1.23 and 0.204*.With*Reg (BE) > 1.0*,E does not become frequently frequent, but with*Reg(E) < 1.0*, E becomes frequently frequent patterns. Node B results in *(B: 2, 3, 4, 5, 6, 9)*, and no conditionals PB exists for it. However, the value of its regularity is *0.77*, which is less than our threshold value for regularity. Therefore, *B* becomes a regular pattern. The conditional PB, CTs, engendered recurrent patterns, and regular engendered FP are summarized. From our example database, as shown in Fig. 3, 10 FPs are obtained. Only four of them are regular patterns.

The statistical formulas of the average and variance can be utilized to calculate the regularity of pattern X. Supposing that *TxPx* is a set for all periods of *X*, then *Px = {p1x, p2x,…, pnx}*, where *n* shows total number of periods for px. The value of the *x* for average period is as follows:

$$\overline{X} = \sum_{k=1}^{N} \frac{P_k^x}{n}$$

Variance for period of X pattern is as follows:

$$\sigma_x = \sum_{k=1}^{N} \frac{\left(P_k^x - X\right)^2}{n}$$

Example: **Table 1** database shows that the patterns *"BF"* happenes in *tid = 2 and tid = 4*.

Thus, *TBF = {2, and 4}*, and the *PBF = {3, 2,and 4}*. For the pattern *"BF,"* the average period *value = 9/3 = 3*, and the variance *value = (3 - 3) 2 + (2 - 3) 2 + (4 - 3) 2 / 3 = 0.67*. The max period for patterns *"BF"* is four (4). When the user specifies the maximum period = 3, "*BF*" is not a regular pattern according to existing method because *Reg(BF) > max Prd*. When the user specifies the maximum *variance = 1.0*, then *BF* becomes the usual patterns because the variance interval for *BF is 0.67*, which shows that *Reg(BF) < max_variance*.

3.4 Experimental Analysis and Results

In this section, the outcome of our complete examination of the performance of CPILT for data streams against artificial and actual sets of data is given in Table 2. Table 2 presents the statistical information on data sets, which are used in the examination.

The 1st three main data sets are attained from [28]. Several years' POS data composed, BMS-POS, are the given worth. Two data sets are achieved from two web locations of e-commerce containing clicking stream data for several months with worth. T40I10D100K is the synthetic data, in which the most number of transactions is possibly the greatest ordinary FPs, and D, I, and T are the parameters used for the transition size of the average. The strength of the entire dissimilar items is shown in the last column of Table 2; they are mostly found in

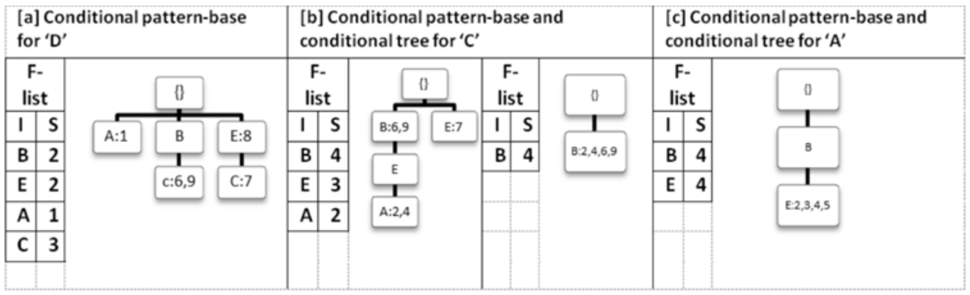

[a] Conditional pattern-base for 'D'
F-list
I S
B 2
E 2
A 1
C 3
{}
A:1
B
E:8
c:6,9
C:7
[b] Conditional pattern-base and conditional tree for 'C'
F-list
I S
B 4
E 3
A 2
{}
B:6,9
E:7
E
A:2,4
F-list
I S
B 4
{}
B:2,4,6,9
[c] Conditional pattern-base and conditional tree for 'A'
F-list
I S
B 4
E 4
{}
B
E:2,3,4,5

FIG. 3. CONDITIONAL PB AND CT

TABLE 1. TRANSACTIONAL DATABASE

ID	Transaction	ID	Transaction	ID	Transaction
1	AD	4	ABEF	7	CDE
2	ABEF	5	ABCE	8	DEF
3	ABCE	6	BCD	9	BCD

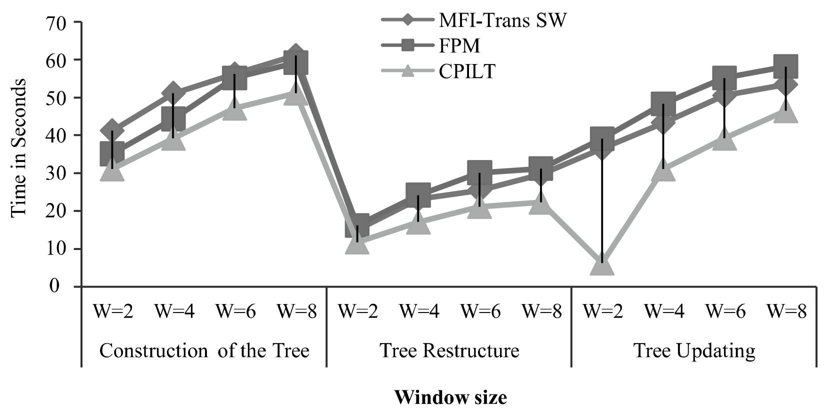
TABLE 2. DATA SET CHARACTERISTICS

Dataset	Transaction#	Items#	MaxTL#	AvgTL#	(ATL/Items)*100
BMS-POS	515,598	1658	165	6.54	0.38
BMS-WebView 1	58,603	499	268	2.51	0.52
BMS-WebView 2	78,513	3341	162	6.01	0.15
T40I10D100K	100,000	943	78	38.62	4.21

every transaction of each data set. In our calculation, we generally use transactions of average length because the connections of the database may vary to some extent. Either the data set is scanned or compressed. When in a transaction, the data set is typically in a manner in which it may be composed of a few items in 1; however, when it is composed of numerous different items, it is called sparse. The dense data set contains numerous items in a transaction with different rare items. When such type of data set has a comparatively small value (e.g. <= *10.0*), the data set is called infrequent; otherwise, the data set is considered dense. Consistent results for all the data sets are shown in Fig. 4 using certain features of the data set with well-defined measurements. For the writing program, Microsoft Visual C++ 6.0 is used. The program is run on Windows 7 with CPU of 2.66 GHz with 1 GB memory. CPU and I/Os are quantified by runtime, which also contains the rearrangement of the tree for only CPILT and tree building. We mainly compare different requirements at runtime, and we focus on FPM [1], MFI-Trans SW[27], and our suggested CPILT. Various studies in the literature [13,16] indicate that Apriori works well for such type of data sets, when the item numbers are reduced, but it does not work well when the patterns are extensive. FPs may be large and/or insignificant maintenance thresholds. The performance matter can be fixed in the method based on the spreading growth of a tree with a frequency descending arrangement [29]. Consequently, straight collective mining patterns can be produced from the static window of a fixed size using both MFI-Trans SW and CPILT, which perform outclass in many circumstances compared with FPM. In the case of plate-based SW in case of MFI-Trans SW and CPILT, we mostly ignore their associated runtime. When the memory obligation of the window is not dependent on the method of SW, the memory evaluation can be created between them. FPM is quite parallel to our suggested CPILT, as is mentioned in [6]. In this study, we relate CPILT to FPM. The experimental analysis can be divided into two parts. First, CPILT density is illustrated with respect to the amount of nodes and memory. Second, the act of FPM and CPILT is illustrated from the perspective of runtime.

3.5 Runtime Competence

We match the total competency of the runtime of CPILT, MFI-Tran SW, and FPM with the construction,



4. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON T40I10D100K DATASET WITH TRANSACTION LIMIT 50 K restructure, and update of a tree when significant changes occur in the size of the windows. Figs. 5-12 on the four mentioned datasets with 50 and 100 K transaction limits show the outcomes. The motive for the enhancement of the efficiency of CPILT is explored. The runtime distribution of CPILT, MFI-Tran SW, and FPM is reported in Figs. 5-12 using different window sizes, and dissimilar data sets are consumed. The Figs. 5-12 show the distribution of the runtime of tree building, tree apprising (i.e. termination period for the window removal in CPILT and removal period of the node of the unwanted/garbage in the case of FPM and MFI-Tran SW), tree reform, and the overall period. The cost of tree reformation involves the cost of dynamic selection for a suitable technique of tree reformation. The average essential time for all the running and dynamic windows is shown in the statistics and demonstration of the column relation. During the tree construction phase, restructure phase tests are performed using different sizes of windows for different data sets, thereby rationally preserving the constraints (i.e. w and p) at great values. The performance of CPILT is also assessed with the disparity of the magnitude of the windows (i.e., with changing values for w and p). The performances of FPM and CPILT are mostly alike for great values of windows using different data sets. The breach in the efficiency during these processes may be amplified when the values of the window are reduced. For most data sets, a prolonged runtime for FPM and MFI-Tran SW occur in the case of a shorter value for the windows or a

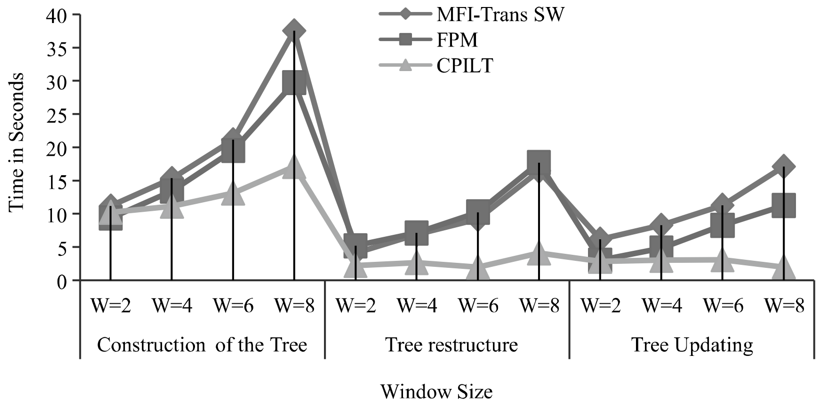


FIG. 5. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON BMS-POS DATASET WITH TRANSACTION LIMIT 50 K

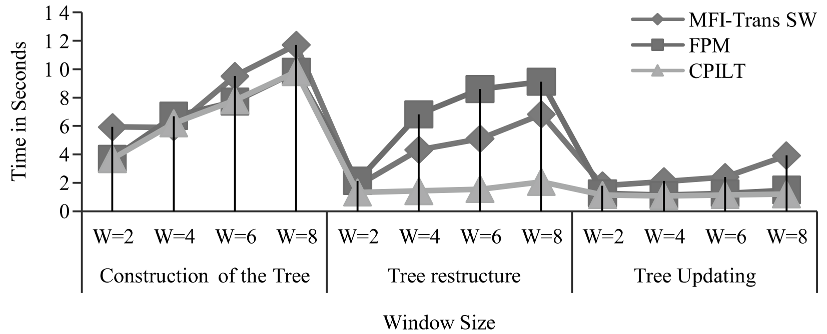


FIG. 6. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON BMS-WEB VIEW 1 DATASET WITH TRANSACTION LIMIT 50 K

greater strength of the frequent item sets. In both circumstances, CPILT generates sensible results within a reasonable time.

The consequences shown in Figs. 4-11 indicate that CPILT outperforms FPM in terms of total runtime with nuances in terms of degree during the tree construction

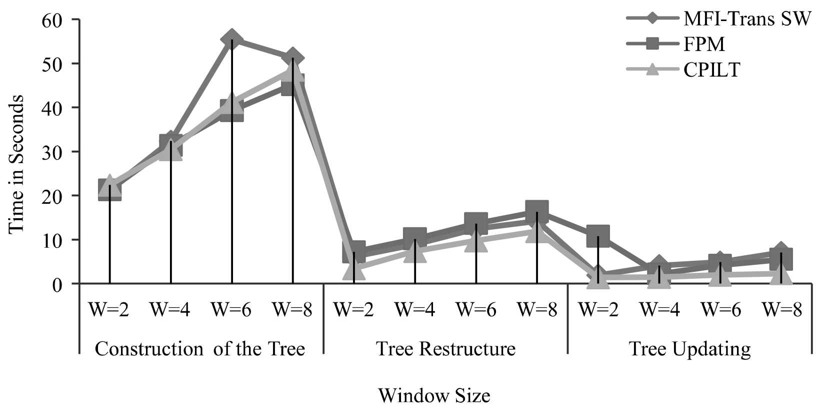


FIG. 7. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON BMS-WEB VIEW 2 DATASET WITH TRANSACTION LIMIT 50 K

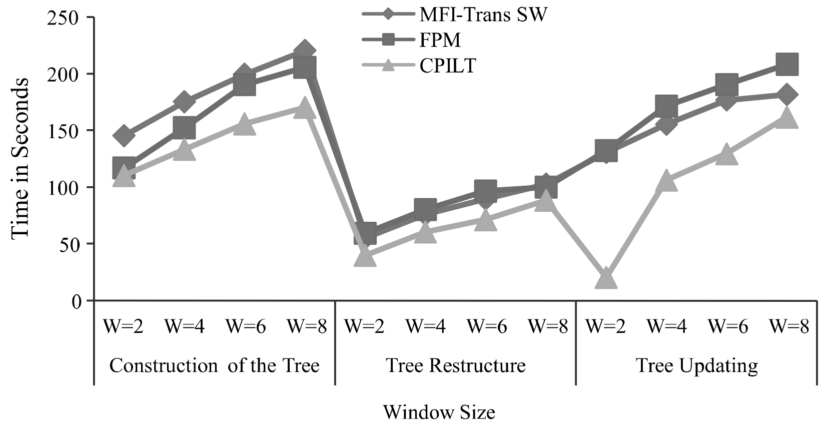
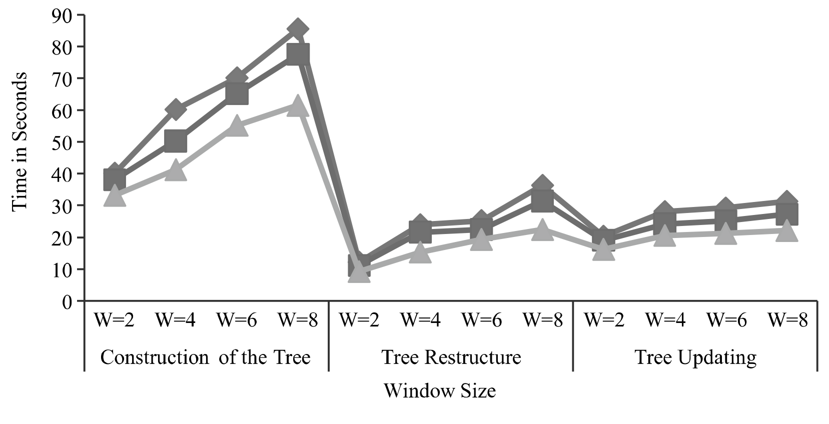


FIG. 8. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON T40I10D100K DATASET WITH TRANSACTION LIMIT 100 K



9. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON BMS-POS DATASET WITH TRANSACTION LIMIT 100 K and tree updating process against all the data sets to be evaluated. For example, in the case of the sparse data sets, BMS POS, BMS WebView 1, BMS-WebView-2, and T40I10D100K, FPM and MFI-TranSW require twofold to six fold longer time than CPILT does. Our experimental results also show that CPILT outperforms FPM and MFI-TranSW in terms of the high values of the windows and small numbers of frequent item sets against data sets with diverse features. For example, against the data set T40I10D100K,whose windows have values of 2, 4, 6, 8, CPILT presents an outclass performance compared with FPM and MFI-TranSW. In apprising a tree, FPM and MFI-TranSW need a long time at every window.

3.6 Memory Adeptness

We prove the memory requirement against FPM, MFI-Trans SW, and our suggested CPILT for dissimilar datasets with varying window scope. We test the memory requirement against the preliminary or primary data structure using the SW methodology using a window with a stable size. We also test the memory requirement using windows of varying sizes for each data set.

The tests are performed throughout the tree construction and structure process, restructuring, and updating in every window by altering the window values of all the datasets, while the window restraint (windoe and plate) are preserved at reasonably immense values. We also

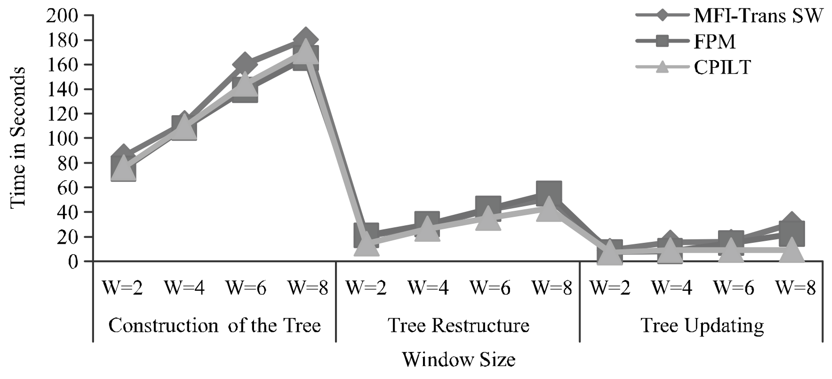


FIG. 10. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON BMS-WEB VIEW 1 DATASET WITH TRANSACTION LIMIT 100 K.

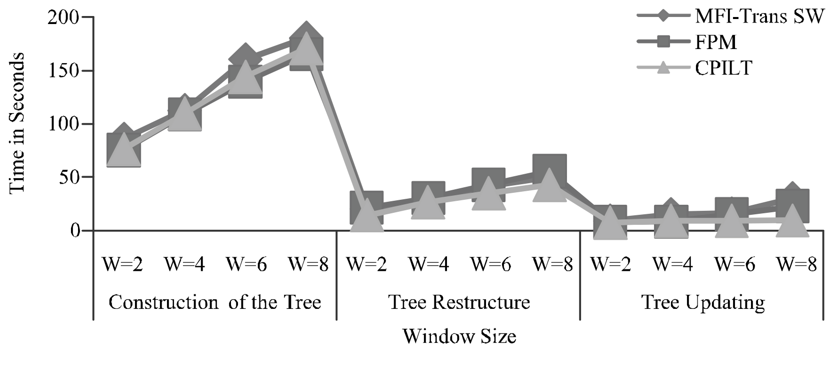


FIG. 11. CONSTRUCTION, RESTRUCTURE AND UPDATING OF TREE ON BMS-WEB VIEW 2 DATASET WITH TRANSACTION LIMIT 100 K

assess CPILT tree performance on the difference of window importance(by altering the values of windows and plates). Figs. 12-19 show the assessments for the physical memory requirement reported for different datasets; marks are displayed. The discrepancy in window size in the form of the plates is shown in the x-axis of the graph. We reserve a static proportion of plates against each single data set. The effects for BMS-POS are shown in Figs. 12-19, with static window sizes at 50 and 100 k (=50 K × 2), corresponding transactions, considering the respective event. For every batch size, we conclude the typical memory requirement for all models of window sliding. For example, using BMS-POS with window sizes of 50 and 100 K (p = 50 K and w1 = 2) transactions, the objects quantity is usually 1319 with respect to all models of SW. With the use of identical data sets on the average, the objects are 1573 with the completion of SW having a static window size of 400 K (p = 50 K and w4 = 8) transactions. In the case of FPM and MFI-Trans SW, we calculate the memory against the windows in consideration of complete objects with bit-arrangement proportions in the windows, even though many supplementary spaces are required by the process of retaining added structures, such as the title and total occurrence of each single object. In the case of BMS-POS, when the window size is 100 K connections, the memory requirements of MFI-Trans SW, FPM, and our suggested CPILT are shown in Fig. 8. A comparison of MFI-Trans SW and CPILT indicates that, when all types of window model are used in maximum data sets, CPILT requires a small amount of memory, especially in the case of sparse data sets. When the data set is sparse, considerable memory is required by CPILT. If the window sizes and number of objects become large, then the memory enhancement of CPILT becomes bulbous relative to that of FPM and MFI Trans SW. Conversely, if the number of discrete objects is small and/or sparse data set is of low slung, then CPILT requires a relatively large memory. The results for the two tree configurations regarding changed data sets using a window with a static size are obtainable as the sum of the nodes. Regular and tail nodes are used for CPILT, whereas steady and garbage/ unwanted nodes are used for FPM and MFI-TranSW. FPM and MFI-TranSW hold 16% to 25% garbage or unwanted nodes, whereas CPILT is free from such garbage nodes. The memory constraint for CPILT can also be reduced by keeping a limited number of tail nodes (i.e., nodes that keep information for the plate counter) relative to the many numbers against regular nodes. When the datasets of T40I10D100K are copied, the only 3% nodes are the tail nodes out of the complete nodes. For further data sets, the tail nodes have a trivial stretching effect from 13% to 34%. Hence, in terms of the quantity of the nodes, CPILT is more memory efficient than FPM and MFI-TranSW for all categories of datasets. The results indicate that the runtime efficiency of CPILT is improved with respect to the organization of its condensed tree.

Fig. 20-23 shows the quality of our proposed algorithm along the number of frequent items, with restoration errors for real BMS-Web View 1, BMS-POS, and synthetic data IBM. The curve in Fig. 20-23 shows that frequent items need to be chosen. Fig. 24 shows that the quality devalued with the number of dissimilar item sets in truncations. The distribution of the data will be very dense when the transaction length is small. May be some duplication between the items and the transactions are exists with low restoration error, displayed in the Fig. 24.

3.7 Restoration Error

We also count the restoration error for the frequent patterns based on the probabilistic model [2]. A restoration procedure for the set of item sets S is the function mapping S to the values between the 0 -1:f: S-$\rightarrow$ [0,1]. The restoration quality is measured by p-norm for the relative errors[2]. We use the above model to minimize the restoration error with respect to the frequent item sets. The restoration error for different data sets with respect to frequent item sets are presented in **Figs. 20-24**.

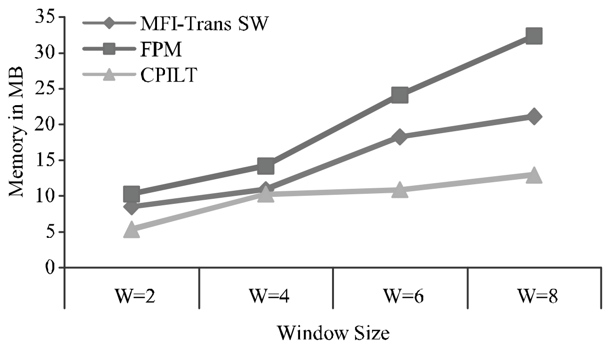


FIG. 12. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 50 K ON T40I10D100K DATASET

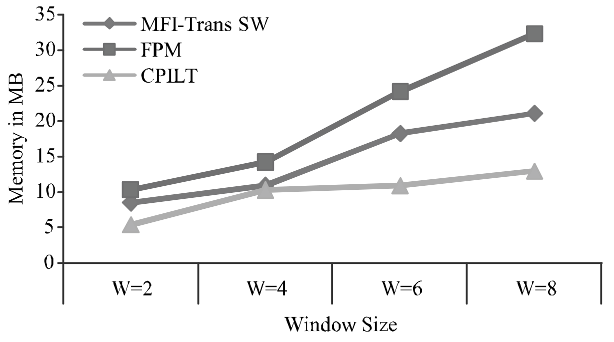


FIG. 13. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 50 K ON BMS_POS DATASET

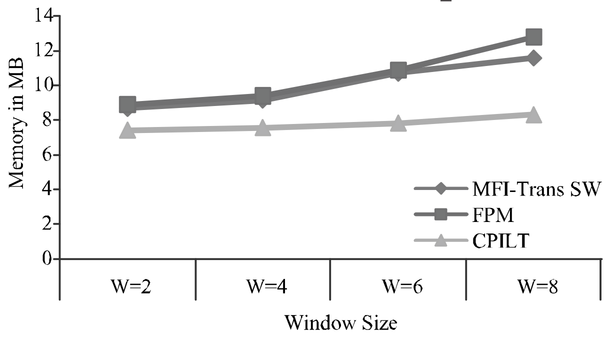


FIG. 14. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 50 K ON BMS_WEB VIEW 1 DATASET

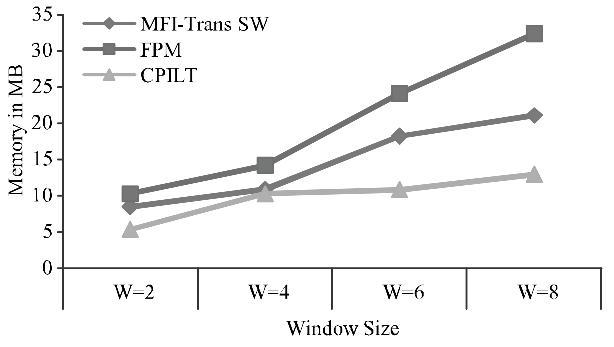


FIG. 15. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 50 K ON BMS_WEB VIEW 2 DATASET

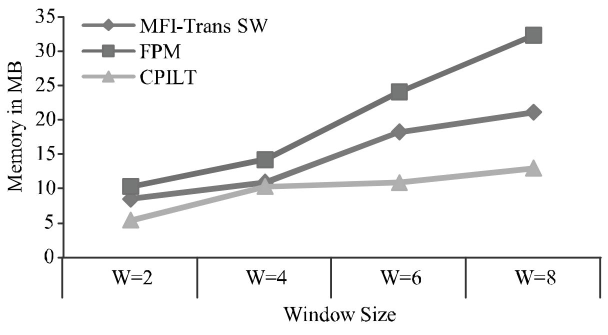


FIG. 16. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 100 K ON T40I10D100K DATASET

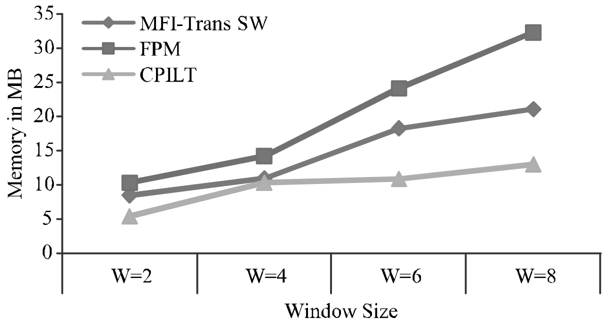


FIG. 17. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 100 K ON BMS_POS DATASET

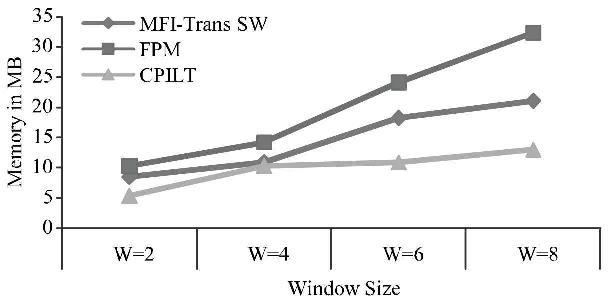


FIG. 18. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTION LIMIT 100 K ON BMS_WEB VIEW 1 DATASET

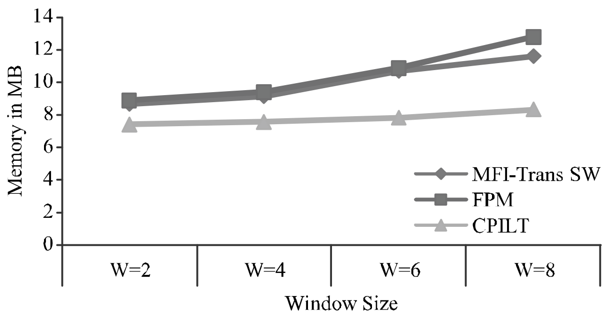


FIG. 19. MEMORY CONSUMPTION ON DIFFERENT WINDOWS WITH TRANSACTZZZZION LIMIT 100 K ON BMS_WEB VIEW 2 DATASET.

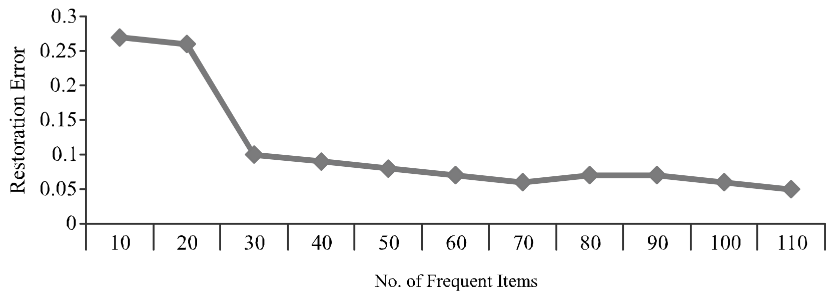


FIG. 20. NUMBER OF FREQUENT ITEMS ON (A) BMS-POS DATA SET

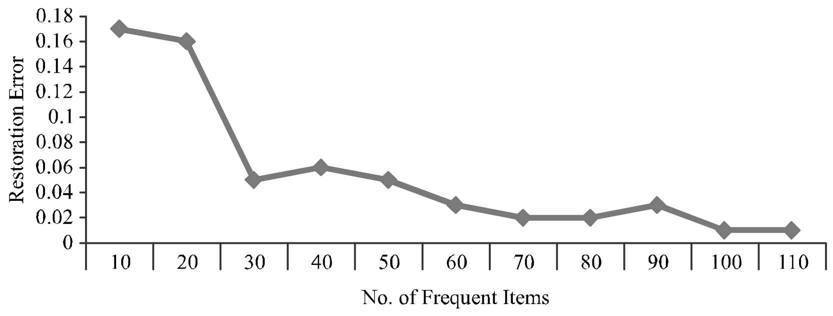


FIG. 21. NUMBER OF FREQUENT ITEMS ON BMS-WEB VIEW 1

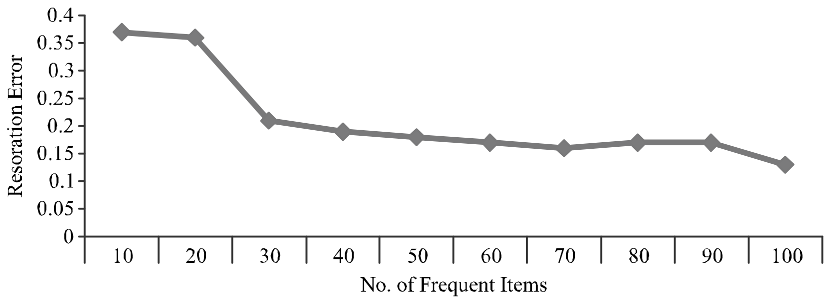


FIG. 22. NUMBER OF FREQUENT ITEMS ON BMS-WEB VIEW 2

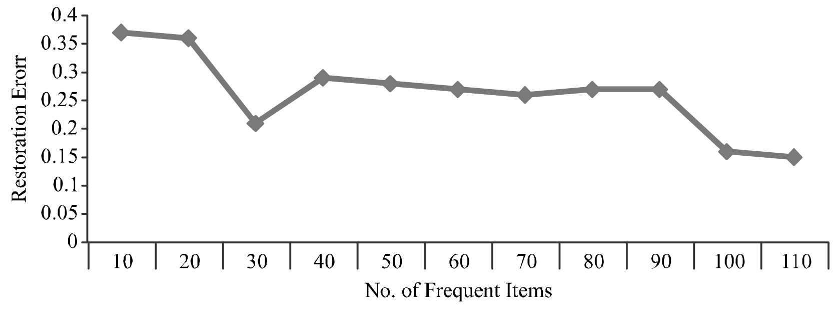


FIG. 23. NUMBER OF FREQUENT ITEMS ON T40I10D100K

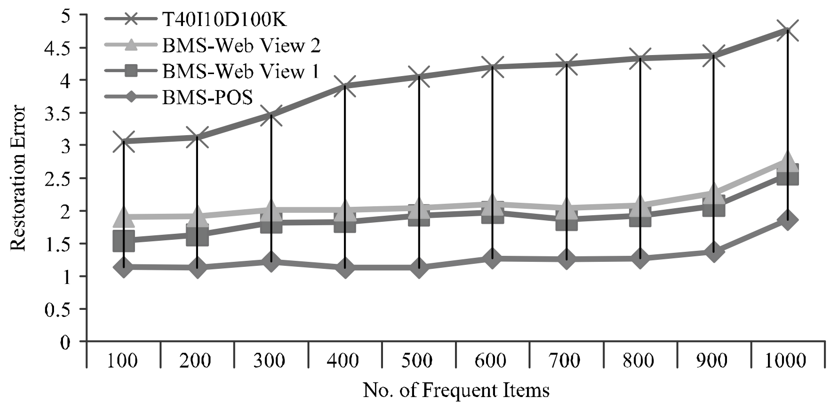


FIG. 24. NUMBER OF DISTINCT ITEMS ON 4 DATA SETS

4. CONCLUSION

In our paper, we presented the newest idea for the mechanism of the dynamic restructure of tree to handle continuous data streams. We proposed and developed a structure for CPILT; this tree restructures itself to achieve a frequency descending compact tree using a single pass. CPILT decreases the runtime and memory capacity for handling high-speed data streams. An effective restructuring mechanism for the structure of CPILT was also proposed and explained. For runtime and memory efficiency, we compared our algorithm with FPM and MFI-TranSW.The analysis of the results showed that our proposed CPILT provides better results than FPM and MFI-TranSW. The future work of our research is to summarize frequent sets of items.

ACKNOWLEDGEMENTS

Authors would like to acknowledge with thanks the anonymous referees for their useful suggestions that led us to enhance the quality of the paper. Authors are also thankful to the International Islamic University, Islamabad, Mehran University of Engineering & Technology, Jamshoro, and Preston University, Islamabad, Pakistan, for providing platform to carry out this research.

REFERENCES

[1] Lee, G., Yun, U., and Ryu, K.H., "Sliding Window Based Weighted Maximal Frequent Pattern Mining Over Data Streams," Expert Systems with Applications, Volume 41, pp. 694-708, 2014.

[2] Jin, R., Abu-Ata, M., Xiang, Y., and Ruan, N, "Effective and Efficient Itemset Pattern Summarization: Regression-Based Approaches", Proceedings of 14th ACM International Conference on Knowledge Discovery and Data Mining, pp. 399-407, 2008.

[3] Faisal, M. A., Aung, Z., Williams, J.R., and Sanchez, A., "Data-Stream-Based Intrusion Detection System for Advanced Metering Infrastructure in Smart Grid: A Feasibility Study", IEEE Systems Journal, Volume 9, No. 1, pp. 1-14, 2014.

[4] Pyun, G., Yun, U., and Ryu, K.H., "Efficient Frequent Pattern Mining Based on Linear Prefix Tree", Knowledge-Based Systems, Volume 55, pp. 125-139, 2014.

[5] Tao, F., Murtagh, F., and Farid, M., "Weighted Association Rule Mining using Weighted Support and Significance Framework", Proceedings of 9th ACM International Conference on Knowledge Discovery and Data Mining, pp. 661-666, 2003.

[6] Liu, G., Lu, H., Xu, Y., and Yu, J.X., "Ascending Frequency Ordered Prefix-Tree: Efficient Mining of Frequent Patterns", Proceedings of 8th International Conference on Database Systems for Advanced Applications, pp. 65-72, 2003.

[7] Borgelt, C., "An Implementation of the FP-Growth Algorithm", Proceedings of 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, pp. 1-5, 2005.

[8] Chen, L., and Mei, Q., "Mining Frequent Items in Data Stream using Time Fading Model", Information Sciences, Volume 257, pp. 54-69, 2014.

[9] Bolaños, M., Forrest, J., and Hahsler, M., "Clustering Large Datasets using Data Stream Clustering Techniques", Data Analysis, Machine Learning and Knowledge Discovery, Springer, pp. 135-143, 2014.

[10] Sun, Z., Mao, K., Tang, W., Mak, L.-O., Xian, K., and Liu, Y., "Knowledge-Based Evolving Clustering Algorithm for Data Stream", 11th International Conference on Service Systems and Service Management, pp. 1-6, 2014.

[11] Chakraborty, S.B., and Shaikh, M., "A Comprehensive and Relative Study of Detecting Deformed Identity Crime with Different Classifier Algorithms and Multilayer Mining Algorithm", Analysis, Volume 3, 2014.

[12] Shie, B.-E., Yu, P.S., and Tseng, V.S., "Efficient Algorithms for Mining Maximal High Utility Itemsets from Data Streams with Different Models", Expert Systems with Applications, Volume 39, pp. 12947-12960, 2012.

[13] Yun, U., Shin, H., Ryu, K.H., and Yoon, E., "An Efficient Mining Algorithm for Maximal Weighted Frequent Patterns in Transactional Databases", Knowledge-Based Systems, Volume 33, pp. 53-64, 2012.

[14] Feng, J., Yan, Z., Kang, Y., Wang, J., and An, L., "MFISW: A New Method for Mining Frequent Itemsets in Time and Transaction Sensitive Sliding Window", 6th International Conference on Fuzzy Systems and Knowledge Discovery, pp. 270-274, 2009.

[15] Ahmed, C.F., Tanbeer, S.K., Jeong, B.-S., and Lee, Y.-K, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases", IEEE Transactions on Knowledge and Data Engineering, Volume 21, pp. 1708-1721, 2009.

[16] Ye, Y., and Chiang, C.-C., "A Parallel Apriori Algorithm for Frequent Itemsets Mining", 4th International Conference on Software Engineering Research, Management and Applications, pp. 87-94, 2006.

[17] Dang, X.H., Ong, K.-L., and Lee, V., "An Adaptive Algorithm for Finding Frequent Sets in Landmark Windows", Scalable Uncertainty Management, Springer Verlag Berlin Heidelberg, pp. 590-597, 2012.

[18] Deypir, M., and Sadreddini, M.H, "A Dynamic Layout of Sliding Window for Frequent Itemset Mining Over Data Streams", Journal of Systems and Software, Volume 85, pp. 746-759, 2012.

[19] Jea, K.-F., Li, C.-W., Hsu, C.-W., Lin, R.-P., and Yen, S.-F., "A Load Shedding Scheme for Frequent Pattern Mining in Transactional Data Streams", 8th International Conference on Fuzzy Systems and Knowledge Discovery, pp. 1294-1299, 2011.

[20] Thanh, L.H., and Calders, T., "Mining Top-k Frequent Items in a Data Stream with Flexible Sliding Windows", Proceedings of 16th ACM International Conference on Knowledge Discovery and Data Mining, pp. 283-292, 2010.

[21] Deypir, M., Sadreddini, M.H., and Hashemi, S., "Towards a Variable Size Sliding Window Model for Frequent Itemset Mining Over Data Streams", Computers and Industrial Engineering, Volume 63, pp. 161-172, 2012.

[22] Rashid, M.M., Karim, M.R., Jeong, B.-S., and Choi, H.-J., "Efficient Mining Regularly Frequent Patterns in Transactional Databases", Database Systems for Advanced Applications, Volume 7238, pp. 258-271, 2012.

[23] Wang, X., Yue, K., Niu, W., and Shi, Z., "An Approach for Adaptive Associative Classification", Expert Systems with Applications, Volume 38, pp. 11873-11883, 2011.

[24] Cesario, E., Grillo, A., Mastroianni, C., and Talia, D., "A Sketch-Based Architecture for Mining Frequent Items and Itemsets from Distributed Data Streams", 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 245-253, 2011.

[25] Parmar, M.A., Sutaria, M.K., and Joshi, M.K., "An Approach for Finding Frequent Item Set Done By Comparison Based Technique", International Journal of Computer Science and Mobile Computing, Volume 3, pp. 996-1001, 2014.

[26] Zhang, L., Wang, M., Gu, Q., Zhai, Z., and Wang, G., "Efficient Mining Frequent Closed Resource Patterns in Resource Effectiveness Data: The MFPattern Approach", Proceedings of 1st Symposium on Aviation Maintenance and Management, Volume 2, pp. 31-41, 2014.

[27] Dhull, A., and Yadav, N., "Mining Maximum Frequent Item Sets Over Data Streams Using Transaction Sliding Window Techniques", International Journal of Computer Science and Network Security, Volume 14, No. 2, pp. 85-85, 2014.

[28] Zheng, Z., Kohavi, R., and Mason, L., "Real World Performance of Association Rule Algorithms", Proceedings of 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 401-406, 2001.

[29] Gouider, M.S., and Zarrouk, M., "Frequent Patterns Mining in Time-Sensitive Data Stream", International Journal of Computer Science Issues, Volume 9, Issue 4, No. 2, pp. 117-124, 2012.