EAI Endorsed Transactions
on Energy Web
Research Article EAI.EU

Frequent Pattern Retrieval on Data Streams by using Sliding Window

P. Mahesh Kumar [1,*] and P. Srinivasa Rao [2]

[1] Assistant Professor of CSE, TKR College of Engineering and Technology, Hyderabad, External Research Scholar, CSE, JNTUK, Kakinada, India.
[2] Associate Professor of CSE, MVGR College of Engineering, Vizianagaram, Andhrapradesh, India.

Abstract

In different applications like recommender frameworks and market examination, regular patterns play a significant role in useful mining data. Mining regular patterns from sliding windows over streaming information has become a complex task. In this examination, the sliding window is utilized to build the framework and FP tree applied to mine the dataset's valuable data. The sliding window has the arrangement of patterns put away in the Matrix, which contains the transaction in the sliding information and then applied to the FP tree. In this paper, the Frequent Pattern Retrieval strategy is planned by utilizing an FP tree approach and a sliding window model to extract noteworthy examples from data streams. The proposed technique accomplished less runtime with low memory use for the Breast disease dataset and different datasets to run the least utility edge contrasted with different existing procedures.

Keywords: Frequent Pattern Retrieval Algorithm, Information Extraction, Sliding Window Stream Data, Candidate Patterns.

Received on 05 November 2020, accepted on 09 January 2021, published on 13 January 2021

doi: 10.4108/eai.13-1-2021.168091

[*]Corresponding author. Email: maheshkumarp@tkrcet.com

1. Introduction

In data mining, Hidden data retrieved from databases used by research, predominantly frequent itemset mining has been necessary in different fields like traffic, medical, networks, and association rules in mobile computing [1]. The data streams are often subjected to frequent probability distribution changes, which is defined as one of the main characteristics [2]. Therefore, the process of identifying the patterns in those data streams leads to a high-challenging task due to the presence of a vast amount of data that are scanned often with high memory consumption [3]. Frequent Pattern Mining (FPM) provides useful information, and studies have been carried out, such as mining high utility patterns [4], [5], [6]. Window techniques such as sliding windows, landmark window, and damped windows are proposed in FPM to capture essential information in the data streams [7], [8]. Various existing techniques process the chunk data items, and several strategies are used to balance the class distribution of the latest chunk [9]. Besides, multiple scans of the entire database are available in these approaches with static databases, where incoming data are processed and provided the real-time response by existing algorithms in data streams [10]. Mining low frequent high utility patterns are more important in analyzing the "top-end" brand. In business areas, the "top-end" brand price is more and has a lower number of sales [11], [12], [13]. Trees are flexible to allow willful vertical and horizontal expansion [14], spur the structure of data into the hierarchy.

Giving probability to each detected point to show circumstances helps us make the perfect decision instead of ignoring uncertainty [15]. Data uncertainty is caused by numerous factors such as data staleness, sampling errors, network latency, and measurement precision limitations [16], [17]. Furthermore, frequent data mining is the initial stage for Associative rule mining automatic summarization and concept drift [18]. The deficiencies of long consumption time, large redundancy probability, and large Root-Mean-Square Error of Approximation (RMSEA) are highly available in the results of data mining applications [19]. Many techniques, such as Multi-Core Algorithm for Frequent Itemsets [20], [21], Outlier detection algorithms for monitoring distance over data streams [22], were developed. However, frequent mining for uncertain data based on the extended support threshold needs more improvement. These methods are using some filtering constraints to find Frequent Patterns (FP) [35], [36], [37], [38]. Although this is a challenging task to find the use pattern, and different patterns carry different importance [23], [24], [25], [26]. In this research, a scheme is proposed based on the FP tree to extract significant information from the dataset. Sliding window data are stored in the Matrix, and the Matrix is given as input to the FP tree, which increases the system's performance. The experimental analysis shows the proposed FP Retrieval has a best performance than the existing method.

The pattern's organization is a literature survey of recent research in pattern mining in the second section, the proposed method explanation in the third section, experimental analysis in the fourth section and finally in the fifth section conclusion and future work are discussed.

2. Literature Survey

Wang. Q and X Wang [26] developed Parallel Mining Collaborative frequent itemsets in Multiple Data stream (PMCMD-Stream). Two algorithms are developed to generate and analyze the potential frequent itemsets from the streams of data. In this technique, they implemented the sliding window method on bit-sequence, which is a single-pass technique. This method increases the efficiency of parallel mining of frequent collaborative itemsets for the multiple data streams and uses low memory. The experimental result of the PMCMD method shows that this technique would use low memory and higher performance than the PFP and H-stream methods. This method also has the flexibility and best efficiency. In the distributed environment, a collaborative technique doesn’t handle more scales of data streams.

Yun, U., Kim, D., Yoon, E., and Fujita, H [27] developed a data mining technique, namely, Mining significance, based on the High Average Utility Pattern Mining (HAUPM) to find the useful information from data streams. This method contains the damped window technique to increase the efficiency in extracting potential patterns from the data's stream environments. Furthermore, the HAUPM provides the data structure and pruning technique to increase the mining technique's efficiency. In the developed method, the user obtained the potential patterns needed to identify the symptoms related to the diseases. The experimental result shows that the HAUPM technique has the best performance than the existing scalability and memory usage method and runtime. The classification or clustering techniques can be applied to increase the reliability of the pattern information.

U. Yun *et al.* [28] developed the Sliding window technique based on High Average Utility Patterns as an SHAU (Sliding window based on High Average Utility) - Tree algorithm to analyze the recent high average utility pattern in the data streams. The sliding window technique is used to divide the stream data into many patch data and kept only the latest batch in its window. In the global SHAU-Tree, batch-lists of nodes are employed to handle every batch. This technique mines the latest and potential patterns in the stream of data. An approach is made to increase the algorithm's performance by minimizing the over-estimated average utilities that are maintained in the proposed data structure. Multiple batches are used to maintain the recent streaming data by using SHAU-Tree. The experiments are conducted on four real-world datasets such as chess, retail, chain-store, and mushroom to validate the sliding window technique's effectiveness. This technique's primary challenge is to maintain the memory usage with minimum sizes, but the stream data sizes are usually very large. Therefore, the developed method requires more space for data storage.

H. Li *et al.* [29] established an efficient algorithm for mining the uncertain data streams in the sliding window of Probabilistic Frequent Itemset Tree (PFIT). In the sliding window, data structure was applied to maintain all the PFI. A depth-first approach was proposed to develop from bottom-up the PFIT and maintain dynamically. To minimize the time complexity in PFIT over Stream (PFIToS). According to the heuristic rule, then another algorithm called PFIMoS+ is developed to improve the PFIMoS efficiency when the minimum support is low or dense data. If the relative minimum support was low, then these two algorithms exhibit poor performance.

Zhi-Hong Deng [31] proposed DiffNodeset for frequent mining itemset from the data. An efficient algorithm called dFIN is implemented to find the frequent itemsets by hybrid search strategy with a set-enumeration tree to achieve high efficiency. The advantage of DiffNodeset is that its memory requirement is too low when compared to the Nodeset. The DiffNodeset is suitable for frequent mining due to its lowest memory requirement. The extensive experiments are carried out on five datasets, namely T40I10D100K, pumps, chess, kosarak, and mushroom datasets in terms of memory and runtime usage. The validated result shows that the dFIN is significantly faster with different mining supports than the existing algorithms. The *DiffNodeset* gives a poor performance in parallel or distributed architecture.

3. Proposed Methodology

Data streaming is common in many applications, and to handle the streaming data, we required data mining techniques. Frequent mining is the technique that measures the pattern that frequently occurs in the dataset. In this research, itemsets present in the dataset are applied in the sliding window technique. The sliding window values are stored in the Matrix, and the matrix values are applied to the FP tree. The architecture of the proposed Matrix FP Tree (MFPT) in the pattern mining method is shown in Figure 1.

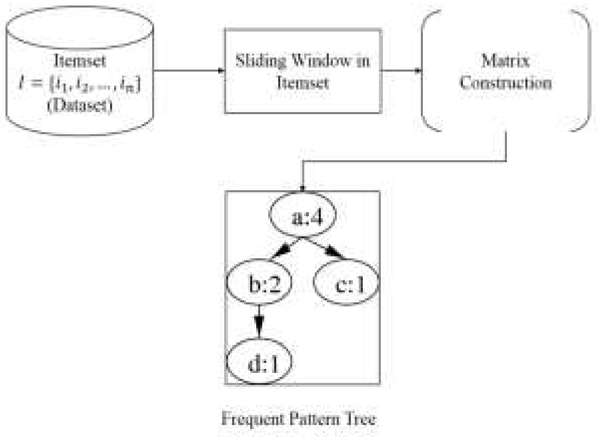

Figure 1. The architecture of the proposed MFPT method in pattern mining

3.1. Matrix Construction based on Sliding Window

Let a set of items is denoted as I={i_1,i_2,…i_m}. Suppose a stream of transactions is denoted as *DS*, received in the sequential order [27]. *I* am superset of each transaction *T*. Transaction *T* in *DS* presumed to have itemsets *X* if $X \subseteq T$, which is also a subset of *I*. Unique number for every transaction named T_{id}. Sliding window *W* over data stream has the latest transactions of the stream [30], [32] with the window size |*W*|. Adding new transactions on the left side of the window and removing the old transactions from the window's right side so that the window slides on stream data. To improve the performance, adding and removing transactions conducted on a batch (pane) of transactions. So the most recent of the input stream transactions are available in the *n* panes of the window. The first transaction id (T_{id}) of each pane as pane id (P_{id}) of that pane and the first P_{id} of the window as window id W_{id}.

An itemset *X* is considered to be frequent in *W* if $Freq(X) \geq n \times |P| \times s$, where *Freq(X), n,* |*P*| and *s* are frequency of *X* in *W*, number of panes, pane size, and support threshold, respectively. The parameters of the mining algorithm are the number of panes in each window and the pane size. And these two parameters are fixed. The problem is finding all FP that present in pane-based transactional window W where the user-specified the supports. Results are continuously updated when the window is processed. The set of frequent itemsets is stored and updated using an ordered prefix tree. That prefix tree node contains a single item and represents an item set constructed by considering that item of a path node from the root to the node. A small amount of memory is required for prefix sharing of an itemset. All items in each path and the level are arranged based on the canonical order. Due to the rapid rate and unbounded stream of transactions, maintaining a frequent itemset belonging to each movement's activate window is acceptable.

Matrix Construction: In this method, we use a matrix to store an uncertain data stream's information. The matrix size is (|*SW*|+1)**m* (row (|*SW*|+1) stands for the support of each, column *m* is the maximal size of items). The support of each item is added to matrix *A* by scanning the transaction. In turn, the probability of item $\{i_k\}$ appearing in T_d is written as $A_{d,k;}$ otherwise, $A_{d,k}$ is written as 0 if item $\{i_k\}$ does not appear in $T_{d.}$ After finishing the current frequent itemset mining process, the sliding window switches to new transactions every time. After constructing the Matrix, each item's support is calculated. These items whose support is less than the predefined *minSup* are not considered in the next "extension" process, and these frequent 1-itemsets are added into *FI_L* (frequent itemset library). For example, matrix construction as follows: the minSup is set to 0.7, and the sliding window size is 5. Transactions t_1, t_2, t_3, t_4, and t_5 are scanned, and items are written successively to matrix *A,* and then the support of each item is calculated. After calculation, 1-itemset{d} is infrequent due to sup({d})=0.5<0.7, and it is discarded to reduce the "extension" process. Then, frequent 1-itemsets of {a},{b}.{c},{e} and {f} are saved into *FI_L.*

3.2. Construction of FP-Tree

The most important technique, called the FP-Tree algorithm, is developed by the researcher Han, proposed in [33], [35]. This method provides the compact of frequent information in the dataset. The FP-tree is explained as follows.

Consider the item set as $I=\{i_1,i_2,\ldots i_m\}$ transaction as *TN* in database *DB* and $Tran \subseteq 1$ describes every transaction *Tran* in an item stem. A set of items is present in the pattern X, which it describes as X⊆1 The Eq. (1) shows the minimum support threshold as σ is less than or equal to X, which supports FP that appears in the *TN* transactions.

$$(\text{Supp}\{X\}/\text{TN}) \geq \sigma \qquad (1)$$

Where σ is a user threshold.

A root in the FP-tree is assigned as null, and leaves of the root are described as references set with the prefix. Every node consists of two major fields in sub-trees: item name and counts [34]. The item name mentions which one to represent, and the numbers of transactions are recorded by count, which represents the path to reach this node.

FP-tree constructed as follows:
Scan *DB* once, and then each item support is identified. According to Eq. (1), if there is a presence of frequent items, then store it in a list called *F*. To form the *F-List*, arrange *F* in non-support ascending order.

For the FP tree, the root is created as *T,* and the values are represented as null. Based on *F-List*, frequent items are selected and sorted for every *Tran* in the *DB*.

Let consider, *[p|P]* is the ordered list of *Tran*, where most frequent items are defined as *p* and the remaining values in the transaction as *P*. The next step is to call the function as insert_tree(*[p|P], T*).

The definition for the function insert_tree(*[p|P], T*) is described as Suppose a child node as *N* is already present in the *T*, then the item name of *p* is equal to the number of item name of *N* and finally increment the *N* by one. If the item name is not the same, then a new node *N* is created, and their parent link is set to *T*. Suppose the values are presents in *p*, then the function called insert_tree(*P, N*) will proceed.

Suppose there are TN transactions [x1,x2,…,xTN] in the database. In that case, each transaction has the value of *n* explanatory variable V_e,$1 \leq e \leq n$, and one V_e as response variable is used to indicate whether accidents have occurred or not. According to Fuzzy C-means as the FCM clustering algorithm, the discrete variables are converted from the continuous variables. The FP tree is constructed based on *TN* transactions with *n* explanatory variables.

In the example, *QFP*s are available, *PQ* describes every pattern, and one branch in the tree is illustrated as $1 \leq q \leq Q$. Numerous nodes as *n* present in every branch, where every node as node *l* is labeled by $f_{l,p}$ as count, and $i_{l,p}$ as the item name. The item name il depicts the variable name$_{,p,}$ and its state of discrete is related to branch and node, where the total number of records to reach the final node from the previous branch are described by count $f_{l,p.}$ In the FP, *l=1,2,..n* where the node order is represented as *l* and two values are considered by the pattern status indicator *p*. If *p=0*, then the node is illustrated as a shared node by more than one FP; otherwise, it is defined as an exclusive node, i.e., *p=q*. In the FP, there are *k* exclusive nodes available and marked as p_1 and p_2.

Once the FP tree construction is over, then exclusive nodes and shared nodes are identified. Later we provide a score to exclusive nodes to differentiate these exclusive nodes and FPs from one another.

Algorithm – 1 Frequent Pattern Retrieval Algorithm

Input: Matrix of window itemset, threshold frequency f_{th} min pattern length *m*, max pattern length *n*.
Output: FP present in the dataset.

```
for each sequence of itemset $a$ in matrix $M$ do
  if $length(a) \geq m$ and $length(a) \leq n$ then
    $f_a \leftarrow frequency\ of\ a$
    if $f_a \geq f_{th}$ then
      for each sequence $b$ that overlap with $a$ do
        if $length(b) = length(a)$ then
          $f_b \leftarrow frequency\ of\ b$
          $d \leftarrow union(a,b)$
          $f_d \leftarrow frequency\ of\ d$
          if $is\_terminal\_node(a)=true$ then
            $patterns.add(a)$
          else if $f_d \geq f_{th}$ then
            $patterns.add(d)$
          else if $f_a > 10 \times f_b$ then
            $patterns.add(a)$
          end if
        end if
      end for
    end if
  end if
end for
```

The FPretrieval algorithm is explained above. The Matrix is given as input to the FP method, and output is obtained as the FP[33]. For all the value of the itemset in the matrix *M*, pattern frequent is measured. If the length of the value of itemset *a* is higher than the minimum pattern *m* and lower than maximum pattern length *n*, then the frequency of *an* itemset is denoted as f_a. When the itemset's frequency is higher than the threshold frequency, the frequency of the itemset *b* is measured, and identify the overlapping element between itemset *a* and *b*. If the frequency of the overlapped element value is higher than the threshold, it is stored in the variable *d.* If the itemset's frequency is ten times higher than the frequency of *b,* then it is stored in *a*. This process continues until the streaming is stopped for the method. In two variables such as *a and b,* the value of FP and overlapped FP is stored.

4. Experimental Analysis

In this section, the validation of the proposed method is analyzed compared with existing techniques on various UCI datasets. Initially, the experimental setup and dataset descriptions are briefly explained. The parameters, such as runtime evaluation and memory usage, are used to validate the proposed algorithm.

4.1. Experimental Setup

FP Retrieval algorithm has been evaluated and compared with different algorithms. The experimental setup has been done using a PC with a 2.2 GHz i7 processor with 16GB RAM and Python 3.6 Jupyter Lab Environment. The proposed FPR algorithm is compared with existing techniques, namely Incremental Two-Phase Average utilities (ITPAU), Incremental Mining of High Average-Utility Itemsets (IMHAUI), Utility Pattern Growth* (UP-growth*), and MPM. The ITPAU algorithm is an incremental algorithm of HAUPM, where the FUP algorithm and Apriori-like approach is employed to minimize the total number of data scans. The UP-Growth* is developed by modifying the original version of the UP-growth algorithm, where the DGN strategy is only used in this approach. During the candidate generation, only two scans are required by DGN employed in the first version of the UP-Growth algorithm. Also, the IMHAUI is used that is also an incremental HAUPM algorithm to handle the data in stream environments.

4.2. Dataset Description

The six real datasets are used for experiments, taken from the UCI Machine Learning Repository, and these datasets can be downloaded from the link: https://archive.ics.uci.edu/ml/index.html. For instance, breast cancer data is presented in the Brest Cancer Wisconsin dataset, where the information about heart disease is obtained in the Heart Cleveland dataset. The list of datasets collected and their statistics given in Table 1.

Table 1. Database Description (N.T: Number of Transactions, N.I: Number of Items, T.W: Transaction Width)

Characteristics	Database					
	Breast_Cancer_ Wisconsin	Liver_Disorders	Heart _Cleveland	Hepatitis	Accidents	Connect
N.T	699	345	303	155	340183	67557
N.I	92	148	274	343	468	129
T.W	10	7	12	19	33.8	8.1
Size	22 KB	9 KB	14 KB	10 KB	56.8 MB	14.6 MB

Pre-processing should be carried out to render these datasets with non-binary object information because the dataset originally has binary item information with FPM datasets. The attribute values are assigned to Accidents, Connect, Liver disorders, Hepatitis, and Breast cancer. The anonymous traffic accident data are available in the Accident datasets, where online connection data are presented in Connect.

A different perspective of evaluation has been done to verify the proposed methodology's effectiveness with different databases. Parameters such as runtime and memory usage have been calculated for different utility threshold parameters and have been compared with existing algorithms to analyze results.

4.3. Runtime Evaluation

First, we look at the runtime execution of the algorithm for various datasets in Figure 2. to Figure 7. This demonstrates the algorithm's execution time under fluctuated minimum support values. In figures, minimum support on the x-axis and execution time (in seconds) on the y-axis. Here, minimum support is the percentage of the total number of transactions of the given dataset. By combining those percentages and the total number of data set transactions, we will obtain relative minimum support values. We applied for fixed minimum support as 0.1 in all experiments. Through these outcomes, we can observe that algorithms' runtime will increase continuously and not be considered because algorithms require more runtime to produce a huge number of patterns and process them at the lowest minimum support.

In contrast to the others, ITPAU requires more execution time to produce and test strategy and must keep all information to mine patterns without considering transaction time.

UP-Growth* also requires higher execution time contrasted with the proposed technique since its HUPM based pattern mining procedure creates countless patterns, even though it includes a damped window technique to consider only recent data in its mining process. IMHAUI also has poor runtime compared to the proposed strategy since it does not count transaction arrival time.

Thus, we can confirm that the proposed strategy has the most noteworthy runtime exhibitions at all minimum support threshold settings by utilizing both the damped window technique and the HAUPM strategy.

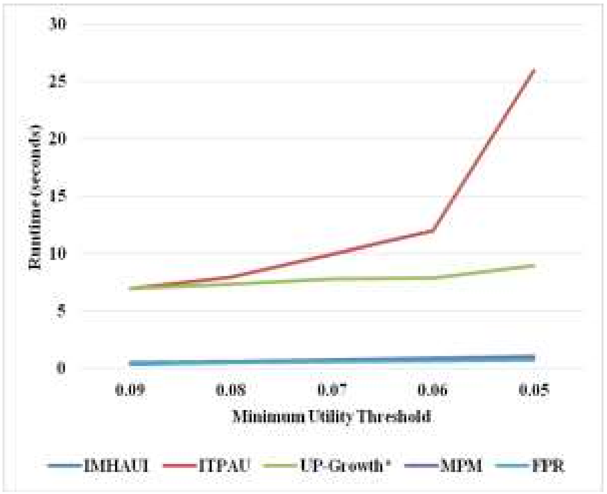

Figure 2. Breast_Cancer_Wisconsin Runtime

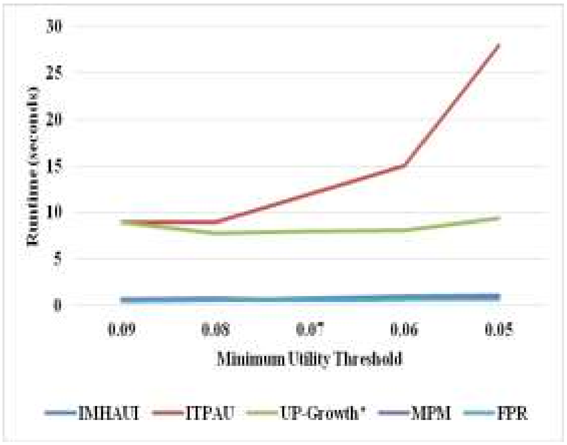

Figure 3. Liver_Disorders Runtime

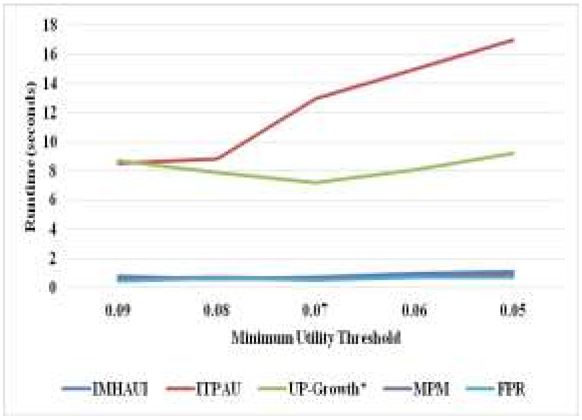

Figure 6. Accidents Runtime

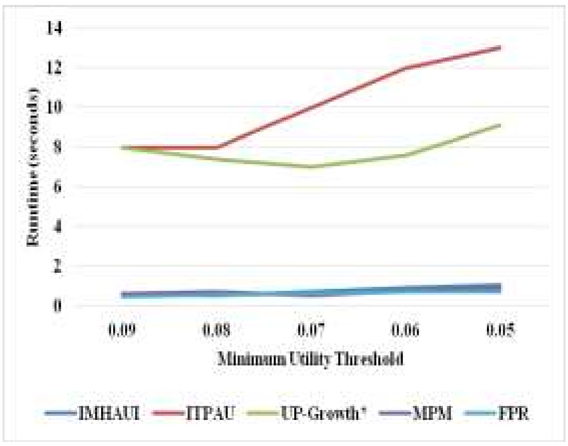

Figure 4. Heart_Cleveland Runtime

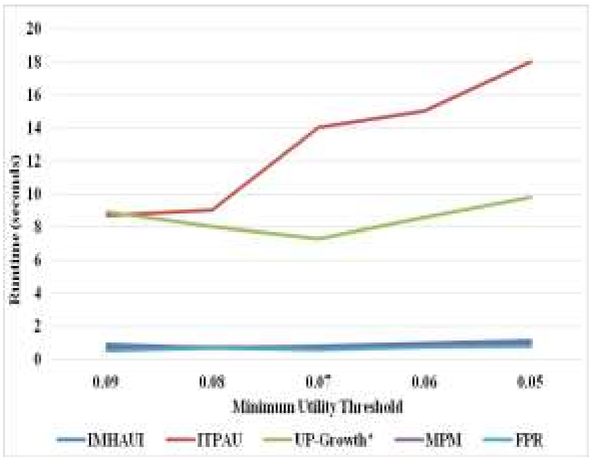

Figure 7. Connect Runtime

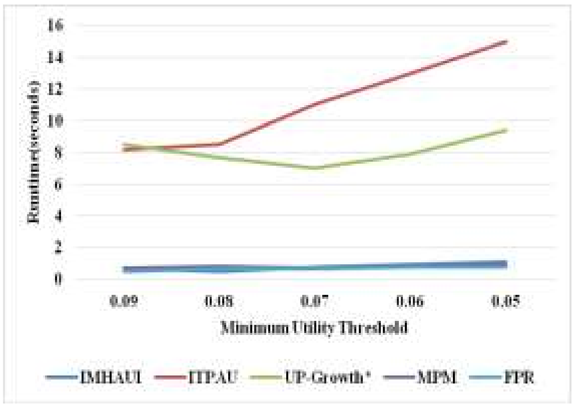

Figure 5. Hepatitis Runtime

From the runtime results that appeared in Figures 2 to 7, we can see that FPR and MPM's contracts don't appear to be enormous. Note that Heart_Cleveland, Liver_Disorders, Breast_Cancer_Wisconsin, and Hepatitis are quite low in terms of transaction numbers. All the contrasting algorithms thus exhibit the best runtime performance. However, as the number of transactions in the dataset increases, the edge progresses toward lowering, the differences between them become increasingly noteworthy.

4.4. Memory Usage

The same experimental settings were considered during the memory usage tests as those of the runtime execution tests. Here Figures 8 to 13 show the memory usage of several techniques. The x-axis indicates the minimum support threshold, and the y-axis denotes the memory space (in MegaBytes) utilized by algorithms. The memory utilization of these algorithms is worst as the minimum support threshold decreases correspondingly as a part of runtime test results in Figure 2 to Figure 7. Specifically, UP-Growth* utilizes more memory because of the High Utility Pattern Mining approach in producing candidate patterns. UP-Growth*, IMHAUI, and ITPAU require more memory contrasted with FPR in Figure 8 to Figure 13. They have to maintain all candidate patterns and retain the valid patterns to decrease the memory scans based on the FUP idea. Consequently, throughout the perception of memory usage tests' investigational effects, we found that FPR has the best memory utilization for preparing results.

The rejection of the execution assessment's effects on the Heart Cleveland dataset for the ITPAU technique from Figure 10 is that the strategy requires too extreme memory and runtime because it cannot work ordinarily. Particularly, algorithm execution in pattern mining relies on the process of candidate pattern mining. As Figure 9, IMHAUI and ITPAU mine similar patterns in number for each situation because they apply average factors in their process. Though, the candidates in the number generated by them are not the same as one another. Remember that in tree-based approaches, Apriori-based methodologies eliminate more candidates than those. Besides, ITPAU produces more candidate patterns for the Heart_Cleveland dataset.

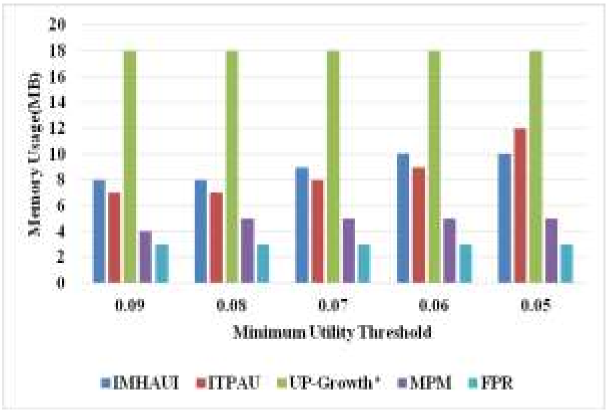

Figure 8. Breast_Cancer_Wisconsin Memory Usage

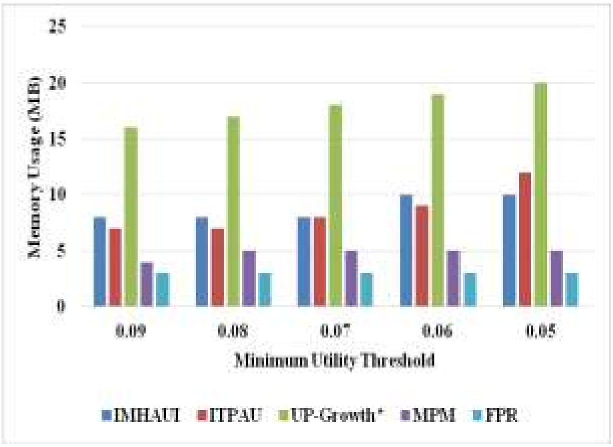

Figure 9. Liver_Disorders Memory Usage

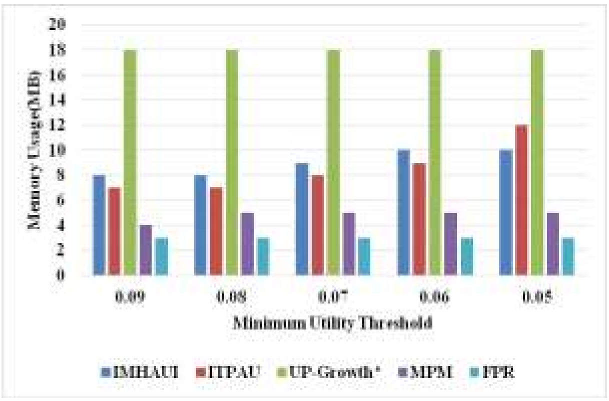

Figure 10. Heart_Cleveland Memory Usage

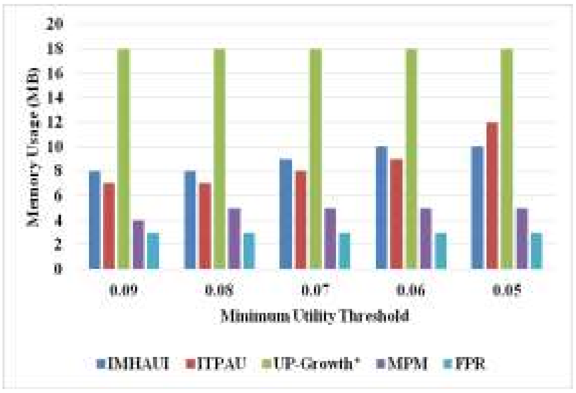

Figure 11. Hepatitis Memory Usage

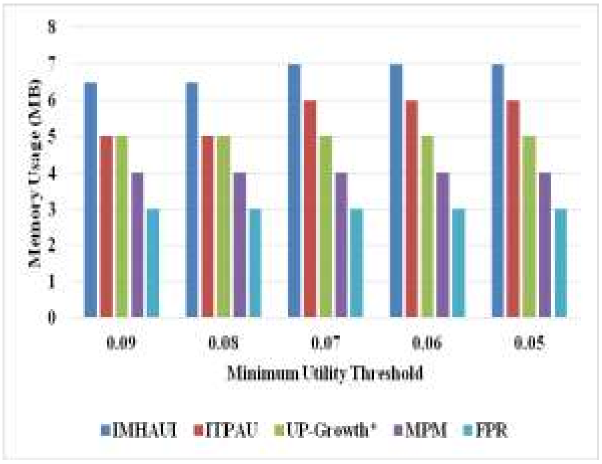

Figure 12. Accidents Memory Usage

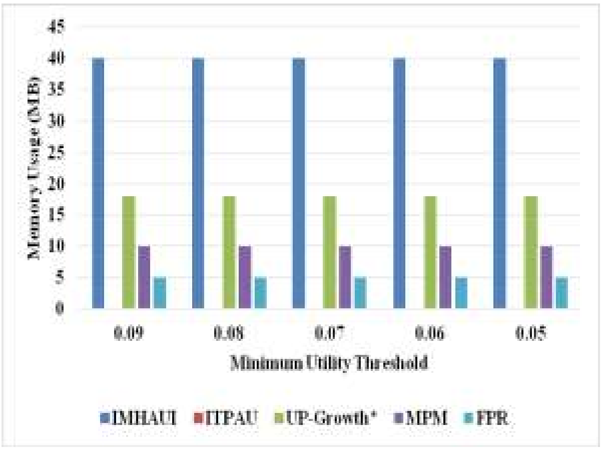

Figure 13. Connect Memory Usage

For large datasets, MPM demonstrates the less memory utilization appeared in Figure 12 to Figure 13. For the Accidents dataset, MPM uses stable memory space, paying little respect to the minimum support settings. In contrast, the space utilization of UP-Growth∗ is step by step greater than before. Particularly, its space utilization is strictly high when the minimum support is brought from 0.09 to 0.08. For the Connect dataset, results are in a comparable situation. The memory utilization of FPR is increasing when the minimum threshold increases from 0.05 to 0.06 for the rationale that the quantities of the retrieved results are bigger for the Connect dataset. The proposed technique ensures the steadiest memory utilization among the other techniques by considering all these issues.

5. Conclusion

This paper proposed the FPR method by utilizing a sliding window and the FP Tree approach to retrieve patterns successfully from the data stream. To improve the efficiency of FPR, the pattern mining process incorporated devised structures and a novel pruning strategy. Users can acquire the most important patterns that are valuable in disclosing disease with a retrieved set of symptoms from data through our method. Our technique compared with other utility pattern algorithms on several UCI datasets. We demonstrated that our technique has much better execution contrasted with others regarding runtime and memory use. Besides, we introduced our strategy's convenience by investigating critical examples mined from the coronary illness dataset. Besides, we introduced our strategy's convenience through the mining Heart Cleveland dataset for significant patterns mined.

5.1. Future Work

We can use other mining methods in conjunction with pattern mining draws near to extract patterns with prime quality. It is possible to find more reliable patterns by making use of clustering and classification techniques. In this manner, we are scheduled to carry out such examinations in our imminent works.

References

[1] G. Lee, U. Yun, and H. Ryang, “An uncertainty-based approach: Frequent itemset mining from uncertain data with different item importance,” *Knowledge-Based Syst.*, vol. 90, pp. 239–256, 2015.

[2] C. Loglisci, M. Ceci, A. Impedovo, and D. Malerba, “Mining microscopic and macroscopic changes in network data streams,” *Knowledge-Based Syst.*, vol. 161, pp. 294–312, 2018.

[3] T. Le, B. Vo, P. Fournier-Viger, M. Y. Lee, and S. W. Baik, “SPPC: a new tree structure for erasable mining patterns in data streams,” *Appl. Intell.*, vol. 49, no. 2, pp. 478–495, 2019.

[4] G. Lee and U. Yun, “A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives,” *Futur. Gener. Comput. Syst.*, vol. 68, pp. 89–110, Mar. 2017.

[5] P Srinivasa Rao, S Satyanarayana, “Privacy-Preserving Data Publishing Based On Sensitivity in Context of Big Data Using Hive,” Journal of Bigdata, Springer, Volume:5, Issue:20, ISSN: 2196-1115, July 2018.

[6] S. A. Moosavi, M. Jalali, N. Misaghian, S. Shamshirband, and M. H. Anisi, “Community detection in social networks using user frequent pattern mining,” *Knowl. Inf. Syst.*, vol. 51, no. 1, pp. 159–186, 2017.

[7] C. Sweetlin Hemalatha, V. Vaidehi, and R. Lakshmi, “Minimal, infrequent pattern-based approach for mining outliers in data streams,” *Expert Syst. Appl.*, vol. 42, no. 4, pp., 1998–2012, 2015.

[8] U. Yun and G. Lee, “Sliding Window-based Weighted Erasable Stream Pattern Mining for Stream Data Applications,” *Futur. Gener. Comput. Syst.*, 2016.

[9] S. Ren, B. Liao, W. Zhu, Z. Li, W. Liu, and K. Li, “The Gradual Resampling Ensemble for mining imbalanced data streams with concept drift,” *Neurocomputing*, vol. 286, pp.

150–166, 2018.

[10] P Srinivasa Rao, MHM Krishna Prasad, K Thammi Reddy, "A Novel Approach For Identification Of Hadoop Cloud Temporal Patterns Using Map Reduce" Published In IJITCS (MECS) Vol. 6, No. 4, Pp:37-42, March 2014.

[11] J. Wang, C. Liu, X. Fu, X. Luo, and X. Li, "A three-phase approach to differentially private crucial patterns mining over data streams," *Comput. Secure.*, vol. 82, pp. 30–48, 2019.

[12] J. Liu, Z. Chang, C. K. S. Leung, R. C. W. Wong, Y. Xu, and R. Zhao, "Efficient mining of extraordinary patterns by pruning and predicting," *Expert Syst. Appl.*, vol. 125, pp. 55–68, 2019.

[13] H. Ryang and U. Yun, "Top-k high utility pattern mining with effective threshold raising strategies," *Knowledge-Based Syst.*, vol. 76, pp. 109–126, 2015.

[14] X. Wu, D. Theodoratos, and T. Sellis, "From Homomorphisms to Embeddings: A Novel Approach for Mining Embedded Patterns from Large Tree Data," *Big Data Res.*, vol. 14, pp. 37–53, 2018.

[15] P Srinivasa Rao, MHM Krishna Prasad, K Thammi Reddy, "An Efficient Semantic Ranked Keyword Search Of Big Data Using Map Reduce," IJDTA, Vol.8, No.6, Pp.47-56,2015.

[16] A. Y. Rodríguez-González, F. Lezama, C. A. Iglesias-Alvarez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and E. M. de Cote, "Closed frequent similar pattern mining: Reducing the number of frequent similar patterns without information loss," *Expert Syst. Appl.*, vol. 96, pp. 271–283, 2018.

[17] B. Qin, Y. Xia, S. Wang, and X. Du, "A novel Bayesian classification for uncertain data," *Knowledge-Based Syst.*, vol. 24, no. 8, pp. 1151–1158, 2011.

[18] L. Bustio-Martínez, M. Letras-Luna, R. Cumplido, R. Hernández-León, C. Feregrino-Uribe, and J. M. Bande-Serrano, "Using hashing and lexicographic order for Frequent Itemsets Mining on data streams," *J. Parallel Distrib. Comput.*, vol. 125, pp. 58–71, 2019.

[19] X. Li, Y. Wang, and D. Li, "Medical Data Stream Distribution Pattern Association Rule Mining Algorithm Based on Density Estimation," *IEEE Access*, vol. 7, pp. 141319–141329, 2019.

[20] P. S. Latha Kalyampudi, P. Srinivasa Rao, and D. Swapna, "An Efficient Digit Recognition System with improved Preprocessing Technique," Springer Nature Singapore, ICICCT 2019 – System Reliability, Quality Control, Safety, Maintenance and Management, pp. 312–321, 2019

[21] L. Bustio-Martínez, A. Muñoz-Briseño, R. Cumplido, R. Hernández-León, and C. Feregrino-Uribe, "A novel multi-core algorithm for frequent itemsets mining in data streams," *Pattern Recognit. Lett.*, vol. 125, pp. 241–248, 2019.

[22] M. Kontakt, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Efficient and flexible algorithms for monitoring distance-based outliers over data streams," *Inf. Syst.*, vol. 55, pp. 37–53, 2016.

[23] Y. Chen, P. Yuan, M. Qiu, and D. Pi, "An indoor trajectory frequent pattern mining algorithm based on vague grid sequence," *Expert Syst. Appl.*, vol. 118, pp. 614–624, 2019.

[24] M. M. Rahman, C. F. Ahmed, and C. K. S. Leung, "Mining weighted frequent sequences in uncertain databases," *Inf. Sci. (NY).*, vol. 479, pp. 76–100, 2019.

[25] P Srinivasa Rao, MHM Krishna Prasad, K Thammi Reddy, "A Novel And Efficient Method For Protecting Internet Usage From Unauthorized Access Using Map Reduce" Published In IJITCS (MECS) Vol. 5, No.3, Pp:49-55, February 2013.

[26] F. Liu, Q. Wang, and X. Wang, "Improved algorithm for parallel mining collaborative frequent itemsets in multiple data streams," *Cluster Comput.*, pp. 1–9, 2018.

[27] U. Yun, D. Kim, E. Yoon, and H. Fujita, "Damped window-based high average utility pattern mining over data streams," *Knowledge-Based Syst.*, vol. 144, pp. 188–205, 2018.

[28] U. Yun, D. Kim, H. Ryang, G. Lee, and K. M. Lee, "Mining recent high average utility patterns based on sliding window from stream data," *J. Intell. Fuzzy Syst.*, vol. 30, no. 6, pp. 3605–3617, 2016.

[29] H. Li, N. Zhang, J. Zhu, Y. Wang, and H. Cao, "Probabilistic frequent itemset mining over uncertain data streams," *Expert Syst. Appl.*, vol. 112, pp. 274–287, 2018.

[30] S.Vidya Sagar Appaji, P. V. Lakshmi, P. Srinivasa Rao, "Maximizing Joint Probability in Visual Question Answering Models," International Journal of Advanced Science and Technology Vol. 29, No. 3, pp. 3914 – 3923,2020.

[31] Z. H. Deng, "DiffNodesets: An efficient structure for fast mining frequent itemsets," *Appl. Soft Comput. J.*, vol. 41, pp. 214–223, 2016.

[32] U. Yun, G. Lee, and E. Yoon, "Efficient High Utility Pattern Mining for Establishing Manufacturing Plans with Sliding Window Control," *IEEE Trans. Ind. Electron.*, vol. 64, no. 9, pp., 7239–7249, 2017.

[33] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Min. Knowl. Discov.*, vol. 8, no. 1, pp. 53–87, 2004.

[34] A. Bhandari, A. Gupta, and D. Das, "Improvised apriori algorithm using frequent pattern tree for real-time applications in data mining," *Procedia Comput. Sci.*, vol. 46, no. Icict 2014, pp. 644–651, 2015.

[35] M. Narvekar and S. F. Syed, "An optimized algorithm for association rule mining using FP tree," Procedia Comput. Sci., vol. 45, no. C, pp. 101–110, 2015.

[36] Indu Chhabra and Gunmala Suri, "Knowledge Discovery for Scalable Data Mining," EAI Endorsed Transactions on Scalable Information Systems, vol. 6, issue. 21, 2019.

[37] Jianguo Jiang, Jiuming Chen, Kim-Kwang Choo, Chao Liu, Kunying Liu, and Min Yu, "A Visualization Scheme for Network Forensics Based on Attribute Oriented Induction Based Frequent Item Mining and HyperGraph", ICDF2C 2017, LNICST 216, pp. 130–143, 2018.

[38] C. Oswald, S. Srinidhi, K. Sri Vishnu, T.V. Vishal and B. Sivaselvan, "Hash based Frequent Pattern Mining approach to Text Compression", COMPSE, 2017.