Frequent Pattern Mining Over Data Stream Using Compact Sliding Window Tree & Sliding Window Model

Rahul Anil Ghatage

Student, Department of computer engineering, Imperial College of engineering & research, Pune, India

---***---

Abstract - *The frequent pattern mining in data streams is more challenging than mining the patterns in traditional databases since several requirements need to be additionally satisfied. In the sliding-window model of data streams, transactions both enter into and leave from the window at each sliding. The continuous, unbounded, and ordered sequence of data items generated at a rapid rate in a data stream so the database become very larger and frequent pattern mining methods have been faced problem that do not appropriately respond to the unbounded data. In this paper, we propose an efficient method to discover the set of latest frequent patterns from dynamic data stream using sliding window model and CSW (compact sliding window) tree. In addition, not only support conditions but also improved weight constraints are used to express the items and gain the resulting maximal frequent patterns more quickly. Extensive experiments report that this method has outstanding performance in terms of runtime, memory usage as compare to previous algorithms.*

Key Words: Data mining, Data stream, Tree, Frequent pattern.

1. INTRODUCTION

The data mining finding potentially useful and hidden information from large databases, and frequent pattern mining is one of the most interesting data mining fields which play an important role in extracting meaningful information. The frequent patterns mining is probably one of the most important concepts in data mining in which extracting informative and useful patterns in massive and complex datasets and patterns comprise sets of co-occurring attribute values, called item sets and different mining techniques and algorithms are used to find frequent patterns. The well-known frequent pattern mining algorithms are apriori algorithm which used by user to generate the candidate item set using breadth first search technique but it only useful for static database and this algorithm required repeatedly scan of database. The another important algorithm is FP- growth which use the depth first search method to find frequent pattern but it required two fixed database scan and it does not generate candidate patterns as comparison to apriori algorithm.

In data stream data is added at rapid rate and thus, they have continuous and unlimited features and their sizes are continuously increased according to the accumulation of transaction data, so the frequent patterns generated over data streams also become very large, which means spending a lot of time mining the patterns, and thereby it can violate the most important requirements for the data stream mining that is immediate processing. Data stream mining has to satisfy the requirements in which each data element required for data stream analysis has to be examined only once and all of the entered data elements have to be processed very quickly and the results of data stream analysis should be available instantly and their quality should also be acceptable whenever users want the results. The old frequent pattern mining techniques do not satisfy these requirements since they need to conduct multiple database scans to mine latest frequent patterns. Therefore, to resolve these problems apply mining approaches which support for single database scan to import the data and use effective tree structure to find latest frequent pattern over data streams effectively. The data streams are used in different industrial fields like network monitoring, sensor network analysis, cosmological application, and intrusion detection, environmental and weather data analysis.

2. RELATED WORK

Many previous research work studies contributed to the efficient mining of frequent item sets over data streams (Chang & Lee, 2003; Manku & Yu, 2005; Thomas, L.T., 2006) [14][15]. According to the stream data processing model (Zhu & Shasha, 2002), the research of mining frequent item sets in data streams can categorized into three categories[21]: landmark-window based mining (Li, 2004; Manku & Motwani, 2002), damped-window based mining concept is for finding frequent pattern (Chang & Lee, 2003; Giannella, 2003), and sliding-window based mining (Chang & Lee, 2004; Ahmed, Tanbeer, 2009)[5].

The most popular frequent pattern mining algorithms is apriori (Agrawal & Srikant, 1994) based on Breadth First Search finds frequent patterns over static databases and generate numerous candidate patterns in the process of actual frequent patterns[3] and FP-growth (Pei, Yin, & Mao, 2004) on the basis of Depth First Search which efficiently conduct mining work but it required two fixed database scans and does not generate candidate patterns[4], Based on this basic frequent pattern mining algorithms, a variety of pattern mining algorithms and techniques have been proposed, such as frequent pattern mining without the minimum support threshold specified by users (Chuang, Huang, & Chen, 2008; Li, 2008)[6], sequential frequent pattern mining (Chang, Wang, Luan, & Tang, 2009; Yun, Ryu, & Yoon, 2011)[2].

Mining of frequent item sets over data stream using sliding window is one of the most important problems in stream data mining with broad applications because different approaches are used for static database and it required multiple scan of database. It is also a difficult challenging issue, such as unknown or unbound size, possibly a very fast arrival rate, inability to backtrack over previously arrived transactions, and a lack of system control over the order in which the data arrive, so new algorithm (Deypir, Sadreddini, 2012) as proposed by three phases like window initialization, sliding window in which involve the window creation, updating operation, pattern generation [10]. The sliding window is an interesting model to solve frequent pattern mining problem since it does not need to consider the entire history of received transactions and it can handle only a limited range of recent transactions. However, previous sliding window algorithms require a large amount of memory and processing time, new algorithm (Mhmood Deypir, 2013) based on a prefix tree data structure to find frequent item sets very quickly[20].

The new approach (Gangin Lee, Unil Yun, 2014) addresses a solution to find the latest patterns over data stream using weighted maximal frequent pattern using sliding window concept [1], they proposed approach in which within single scan we read the database and load the translation and find latest frequent patterns using sliding window model but the limitation of this model is it's tree structure because they used FP like tree which does not generate candidate patterns and it required much more time to perform insertion, deletion and searching operation to find them frequent patterns over data stream, so the new approach proposed (Amol Ghoting, 2012) is use to address the limitations of FP tree and use of B+ tree to increase the performance of the system and find out the frequent patterns quickly[8].

3. MOTIVATION

As per the brief literature survey the main limitation of existing system is tree structure which allow discover the frequent item sets without candidate item set generation and it is very difficult to implement due to complex data structure and it takes lots of time to build and it needs more memory for storing the data items and existing system use the formulas to calculate the maximal weight of each transaction involve the duplicate data items because of this reason it not accurately generate the maximal weight of each data item so due to this reason the main motivation is use the improved compact sliding window tree and weight conditions to find latest maximal weighted frequent pattern over data stream with in a single scan by analyzing each data item only once and the results of data stream analysis should be available instantly as well as their quality should also be maintained with limited memory usage and finally effectively compressing the generated frequent patterns over the dynamic data stream.

4. SLIDING WINDOW BASED WEIGHTED MAXIMAL FREQUENT PATTERN MINING OVER DATA STREAM

In this section, we first consider the preliminaries which focus on concepts which are used for proposed system and compact sliding window tree structure for storing the data items.

4.1 Preliminaries

The data stream (DS) can define as an infinite sequence of transactions DS= (T1, T2 ...Tn) where transaction T is set of items (I) which is defined as I= (i1, i2...in) each item has unique value and each item have weight (w) and set of weight is defined as W= (w1, w2... wn) and pattern is combination of one or more items available in transaction.

The data stream is divided into number of sliding window DS= (SW1, SW2 ... SWn) where each sliding window (SW) consist number of panes, SW= (P1, P2... Pi) where $1 \le i \le n$ and pane is set of transaction, P= (T1, T2... Tn). The weight of each item is calculated by following formula in which we consider the total count of item (O) divided by distinct item (D). The support of a pattern C in a W, denoted as PSUP is the number of transactions in SW. Therefore, a pattern is called frequent in W if its support is greater than an absolute minimal support threshold value δ. The average weight of pattern (AWP) is calculated by set of weight (WSET) divided by number of item (R) then finally we calculate weight support,
Weight support (WS) = PSUP*AWP

If this weight support (WS) is greater than or equal to threshold value, δ than it known as weighted frequent pattern.

4.2 Compact sliding window tree structure

In related work we already consider the limitations of tree structure of existing system that it does not generate candidate pattern so we required suitable tree structure to calculate the weighted maximal frequent pattern over sliding window based data stream. Here we use the CSW (compact sliding window) tree which based on B+, in this tree consist the root, internal node and leaves and when we compare this tree with existing system tree structure then it have several advantages like it store the data in leaf node so the searching of any data is very easy because all data is found in leaf node, high fan out, leaf nodes of this tree structure are linked together so full scan of all items completed within single linear pass.

Item	Weight	Count	Link
B	0.7	2	
C	0.6	3	
E	0.5	1	
F	0.4	1	
A	0.3	1	
D	0.6	1	

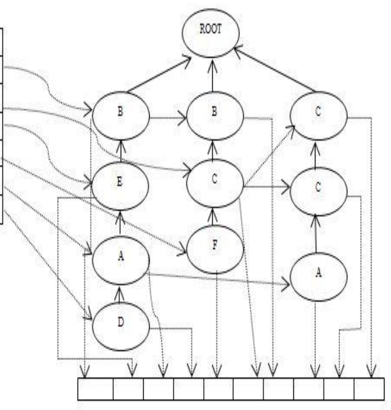

Fig. 1. Compact sliding window (CSW) tree structure

5. HIGH LEVEL DESIGN

5.1 Data independence and system architecture

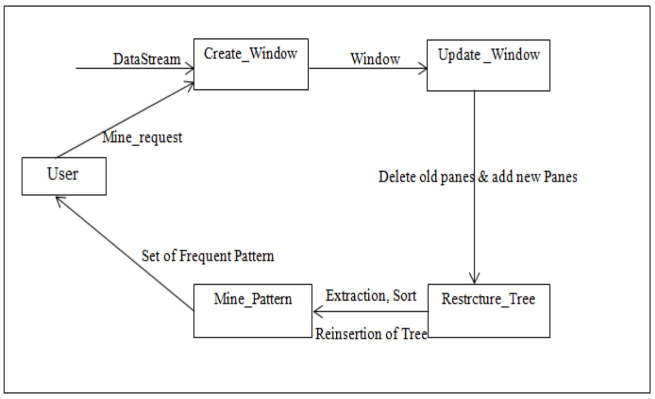

Fig. 2. Data flow architecture

In system architecture when user make the mine request then system first load the transactions from data stream within a single scan, after the successful database connection, the transactions are imported into compact sliding window tree with support value, the imported data pass as input to create window module which create the window and specify the each item count and it's weight, our system support for mining latest frequent pattern from data stream due to this reason, here we use sliding window in the next update window module which take the input from create window module and delete the old panes and add new panes to mine latest frequent patterns then updated window pass as input to restructure module which perform the extraction, sorting and reinsertion operation then pass the sorted, extracted data as input to final mine module which mine all the transactions over dynamic data streams and find the latest frequent patterns which are greater than threshold value and finally display the set of latest frequent patterns over the dynamic data stream.

5.2 Set Theory

Let S represent our proposed system
S = {I, O, F, Su, Fa, ϕ }
Where,
I is an Input I = {DS, Ws, Ps, δ}
Where, DS = Data Stream
Ws = Window Size
Ps = Pane Size
δ = Threshold value
O is an output = {P}
Where, P= set of frequent patterns
Success= Frequent pattern
F is a function = {LC, CW, UW, RES, MINE}
LC= LoadConnection (DS)
Input = DS (Data Stream)
Output = Imported Data
Failure = No Data imported
CW= CreateWindow (ID)
Input = Imported Data
Output = Window
Failure = No Window Created
UW = UdatedWindow (W)
Input = Window
Output = Updated Window
Failure = No Window Updated
RES = Restructure (W)
Input = Updated Window
Output = Sorted Window
Failure = No Sorted Data
MINE = Mining (Sorted Data)
Input = Sorted Data
Output = Frequent Pattern
Failure = No frequent Patterns
Fa is an failure = No frequent patterns
ϕ is constraints of the system
ϕ = {DS, Ps, Ws}

5.3 Mathematical model using Deterministic Finite Automata

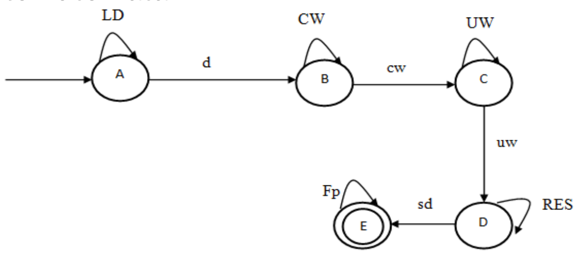

Fig. 3. Deterministic Finite Automata (DFA)

A deterministic finite automaton M is a 5-tuple, (Q, Σ, δ, q0, F) consisting as follow
A finite set of states (Q) = {A, B, C, D, E}
A finite set of input symbols called the alphabet (Σ) = {d, w, uw, sd, fp}
A transition function (δ: Q * Σ → Q) = {LD, CW, UW, RES, MINE}
A start state (q0 ϵ Q) = {q0}

A set of accept states (F) = {q4}

Where: d-document, w-window, uw-update window, sd-sorted data, fp-frequent data, LD-load data, CW-create window, UW- update window, RES- Restructure, MINE-Mine

Derivation (δ) is defined by following transition table

Table 1 - Transition Table

Alp. / States	D	W	uw	Sd	fp
A	B	Ø	Ø	Ø	Ø
B	Ø	C	Ø	Ø	Ø
C	Ø	Ø	D	Ø	Ø
D	Ø	Ø	Ø	E	Ø
E	Ø	Ø	Ø	Ø	Ø

5.4 Sliding window based frequent pattern mining algorithm

In this section we consider the four important algorithms which are used to perform create, update, restructure the sliding window and perform the mine operation.

The following steps are considered for latest frequent pattern mining LFPM algorithm.

1) At the initial level CSW tree, order, prefix and set of frequent patterns are empty.
2) If there are transactions in data stream then import the data.
3) If CSW tree is empty call create window algorithm.
4) Else call update window algorithm.
5) Call restructure algorithm.
6) If the mine request comes from user then call mine algorithm.

1) Create Window Algorithm

The create window algorithm is use to inserting data items into a compact sliding window tree. The window size indicates the number of panes and the pane size represent the number of transactions. Transactions are inserted in tree, T according to their incoming order and create window algorithm is use to computes support descending order regarding the items composing tree.

```
Algorithm (Create Window)
Input - Data stream, DS, Window size, Ws, Pane size, Ps
Output - created window, w
Variables - CSWT- Tree, Tr - New transaction, W - Weight;
Method –
Begin
Window size, Ws = ø, current pane size, Ps = ø;
if there are new transactions inserted into DS
for each new transaction, Tr in DS, do
if (Order != ø)
insert items of Tr into CSWT according to Order;
else insert items of Tr into T in their incoming order;
update supports of the inserted items;
update W as weights of Tr;
end for
End
```

2) Update Window Algorithm

This algorithm updates the sliding windows as per the flow of data stream and delete the old panes and decrease the node support, node with 0 support are deleted directly and insert the new panes.

```
Algorithm (Update Window)
Input - Created window, w
Output - Updated window, uw
Method -
Begin
for each path, pi in the old pane, do find tail node ti for pi
for each node, nk in pi, do
calculate nk support;
if nk.support = 0, do
delete nk;
shift pane counters of all remaining tail nodes in CSWT to left by one;
for each transaction, Tr of the new pane in DS,
do
insert Tr into T according to Order;
set tail node information for Tr;
calculate the Order support descending order for the items included in the current CSWT;
End
```

3) Restructure Algorithm

The restructure algorithm is use to perform the extraction, sorting and reinsertion operations and nodes with 0 support are deleted.

```
Algorithm (Restructure)
Input - Updated window, uw
Output - Restructured sorted data,
Variables – Pr – Path, CSWT- Compact sliding window tree
Method -
Begin
Sort Order in their support descending order;
For each path, Pr in CSWT, do
if Pr is not sorted
extract Pr from CSWT and decrease support of Pr by last nodes support in Pr;
if there are nodes with 0 support in T delete the corresponding node;
sort Pr depending on Order
reinsert the sorted Pr into CSWT
end for
End
```

4) Mine Algorithm

The mine algorithm is use to perform mining operation, when mining request is received from user then first delete the some data items using maximum weight Mw value, if tree as single path then mine algorithm check whether it is really weighted maximal frequent pattern to combine T's all items with the prefix. If the weight support is greater than the threshold value then combine all frequent patterns into set of frequent patterns, P and if multiple paths are available in the tree then call the algorithm recursively.

```
Algorithm (Mine)
Input - Restructured sorted data, sd
Output - Set of frequent patterns, P
Method -
Begin
for each data items, dk in T header table, do
prefix ← dk
calculate Mw which is the maximum weight among the items weights in CSWT;
if prefix.support * Mw < δ, do
check the new data item in header table; else
construct conditional CSWT tree;
if CSWT has single-path,
do
pattern P = prefix ∪ all items in CSWT;
if WS(pattern) ≥ δ, do
P = P ∪ pattern;
else
for each items, im in the header table of CSWT, do
prefix = prefix ∪ im;
Mine(T, prefix, P);
end for
End
```

6. PERFORMANCE EVALUATION

In this section, we evaluate performance of the proposed LFPM algorithm through extensive experiments. The target algorithm is WMFP-SW. The algorithm is written in java and used Core 2 duo, 2 GB ram and window 7 OS. The Real data sets used in run time and memory usage experiments are Accidents, Pumsb, Retail, and Mushroom, where they can be obtained at http://fimi.cs.helsinki.fi/data. The Accidents dataset consists of anonymous traffic accident data and pumsb dataset includes census data. They have a dense feature. Retail dataset is sparse and contains basket datasets in a retail supermarket store. Mushroom dataset with a dense nature contains hypothetical sample data corresponding to mushrooms. In performance evaluation we consider the run time and memory usage for different threshold values. The tables 3 represent the datasets details which are used in experimental work.

Table 3 - Datasets Details

Datasets	No of Transactions	Size(M)
Retail	88,162	3.97
Pumsb	49,046	15.9
Mushroom	8124	0.83
Accidents	340,183	33.8

The following table show the overall pane size and window size use for the datasets.

Table 4 - Window and pane size used for datasets

Datasets	Pane Size	Window Size				
		W_1	W_2	W_3	W_4	W_5
Retail	10	2	3	4	5	6
Pumsb	5	2	4	6	8	6
Mushroom	1	2	4	6	8	6
Accidents	50	2	3	4	5	6

6.1 RUNTIME ANALYSIS

In these runtime experiments, weight ranges for each used datasets are set randomly. Figure shows a runtime result of the Retail, Accidents dataset. The LFPM algorithm presents the fastest runtime performance in all cases. The algorithm finds the latest frequent patterns which weight value greater than threshold value. When we compare our run time with WMFP-SW then our algorithm required less time as compare to current algorithm in all cases. As per the threshold value we compare our algorithm with existing algorithm and in all cases it gives better performance for runtime.

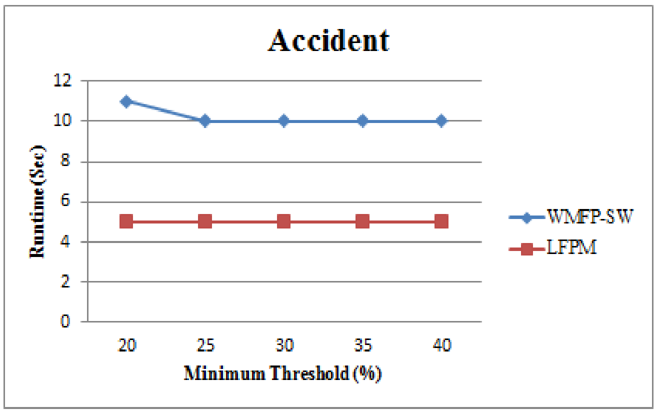

Fig. 6. Accidents Dataset (W2)

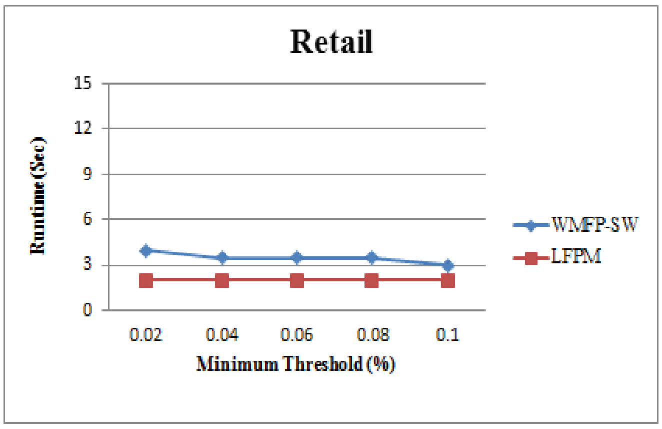

Fig. 7. Retail Dataset (W5)

6.2 MEMORY USAGE ANALYSIS

The fig. 8 and fig. 9 are present the results of memory usage tests for the real and synthetic datasets, where weight ranges for the datasets are set in common with the runtime experiments. LFPM performs mining operations using an data structure, compact sliding window tree (CSW tree) because of this reason it provide outstanding performance and when we compare the memory usage with WMFP-SW algorithm then it consumes very less memory. We consider the different threshold value and for every threshold value it consumes very less memory as compare to existing system.

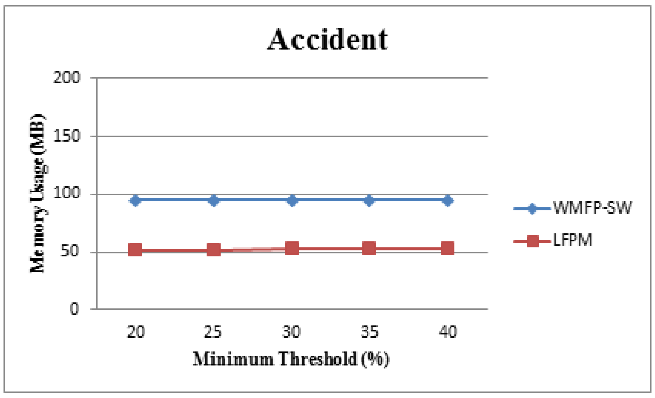

Fig. 8. Accident Dataset (W2)

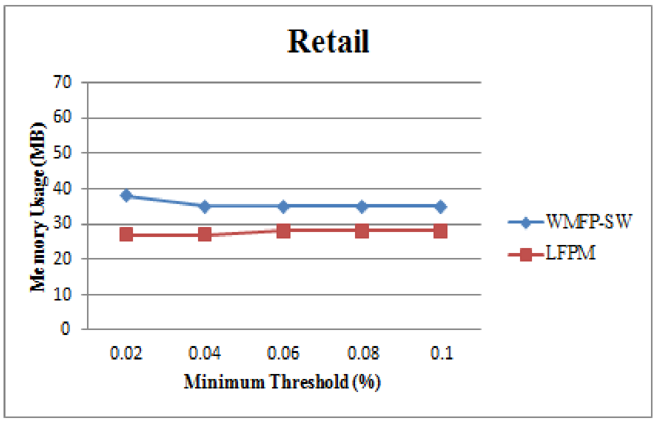

Fig. 9. Retail Dataset (W5)

6.3 SCALABILITY RESULTS

The fig. 10 represent the T10I4DxK datasets in terms of runtime, where the weights values between 0.5 and 0.8, their threshold value is fixed as 0.1%, and number of transactions are increases from 100 to 1000 K. In this figure graph slop is sharply increase after the number of transaction 200k, which means that their scalability results by the increasing transactions are unfavorable and the runtime of LFPM stably increases according to the growth of the transactions, so the algorithm LFPM support for scalability and when we compare the experimental result with existing algorithm WMFP-SW then it gives outstanding runtime results.

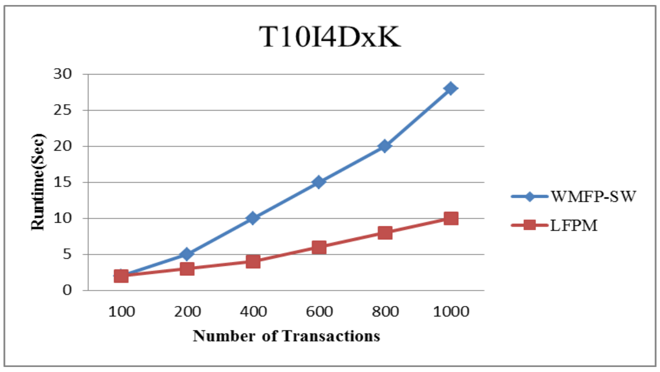

Fig. 10. Runtime scalability of T10I4DxK (d = 0.1%)

7. CONCLUSION

Our approach mainly focus on finding frequent pattern within single database scan over dynamic data stream using effective compact sliding window tree structure and improved weight conditions which avoid the duplicate items in transactions and provide accurate frequent patterns, additionally it extract mining results regarding the latest data over data streams, and can gain the resulting patterns more quickly, so this approach is very useful to find frequent pattern over wide range of data stream like retail, accident, marketing and network related data stream. The extensive experiments presented in this paper showed that the proposed algorithm could mine latest frequent patterns more effectively than the previous algorithms and it gives outstanding performance in terms of runtime, memory usage. The suggested techniques and strategies can be also applicable in other mining areas such as closed frequent pattern mining, high utility pattern mining.

8. REFERENCES

[1] Gangin Lee, Unil Yun , Keun Ho Ryu, "Sliding window based weighted maximal frequent pattern mining over data streams", Expert Systems with Applications, vol. 41, pp. 694–708, 2014.

[2] Chen, Y., Bie, R., & Xu, C., "A new approach for maximal frequent sequential patterns mining over data streams", International Journal of Digital Content Technology & its Application, Vol. 5(6), pp. 104–112, 2011.

[3] Agrawal, R., & Srikant, R., "Fast algorithms for mining association rules", in proceedings of the 20th international conference on very large databases, pp. 487–499, 1994.

[4] Jiawei Han, Jian Pei, Yiwen Yin, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, Vol. 8, pp. 53-87, 2004.

[5] Ahmed, C. F., Tanbeer, S. K., Jeong, B. S., "An efficient algorithm for sliding window-based weighted frequent pattern mining over data streams", IEICE Transactions, Vol. 92-D (7), pp. 1369–1381, 2009.

[6] Li, H., "A sliding window method for finding Top-k path traversal patterns over streaming Web click-sequences", Expert Systems with Applications, Vol. 36 (3), pp. 4382 - 4386, 2008.

[7] Chen, Y., Bie, R., & Xu, C., "A new approach for maximal frequent sequential patterns mining over data streams", International Journal of Digital Content Technology and its Applications, Vol. 5 (6), pp. 104–112, 2011.

[8] Amol Ghoting, Gregory Buehrer, Srinivasan Parthasarathy, "Cache-conscious Frequent Pattern Mining on a Modern Processor", Springer VLDB Journal, vol. 16, Issue 1, pp. 77-96, 2006.

[9] Chen, H., Shu, L., Xia, J., & Deng, Q., "Mining frequent patterns in a varying-size sliding window of online transactional data streams", Information Sciences, vol. 215, pp. 15–36, 2012.

[10] Deypir, M., Sadreddini, M. H., & Hashemi, S., "Towards a variable size sliding window model for frequent item set mining over data streams", Computers & industrial engineering, Vol. 63(1), pp. 161–172, 2012.

[11] Farzanyar, Z., Kangavari, M. R., & Cercone, "Max-FISM: Mining (recently) maximal frequent item sets over data streams using the sliding window model", Computers & Mathematics with Applications, Vol. 64(6), pp. 1706–1718, 2012.

[12] Zhang, X., & Zhang, Y., "Sliding-window Top-k pattern mining on uncertain streams", Journal of Computational Information Systems, vol. 7(3), pp. 984–992, 2011.

[13] Yuh-Jiuan Tsay, Tain-Jung Hsu, Jing-Rung Yu, "FIUT: A new method for mining frequent item sets", Information Sciences, vol. 179, pp 1724–1737, 2009.

[14] Thomas, L.T., Valluri, S.R., & Karlapalem, K., "MARGIN: Maximal frequent subgraph Mining", In Proceedings of the 6th IEEE international conference on data mining, pp. 1097–1101, 2006.

[15] Gouda, Zaki, M. J., "GenMax: An efficient algorithm for mining maximal frequent item sets", Data Mining and Knowledge Discovery, vol. 11(3), pp. 223–242, 2005.

[16] Luo, C., & Chung, S. M., "A scalable algorithm for mining maximal frequent sequences using a sample", Knowledge and Information Systems, vol. 15(2), pp. 149–179, 2008.

[17] Tanbeer, S. K., Ahmed, C. F., Jeong, B. S., & Lee, Y. K., "Efficient single-pass frequent pattern mining using a prefix-tree", Information Sciences, vol. 179(5), pp. 559–583, 2009.

[18] Yang, C., Li, Y., Zhang, C., & Hu, Y., "A novel algorithm of mining maximal frequent pattern based on projection sum tree", Fuzzy Systems and Knowledge Discovery, vol. 1, pp. 458–462, 2007.

[19] Zhi-Hong Deng, "Fast mining Top-Rank-k frequent patterns by using Node-lists", Expert Systems with Applications, Vol. 4, pp. 1763–1768, 2014.

[20] Mhmood Deypir, Mohammad Hadi Sadreddini, "An Efficient Sliding Window Based Algorithm for Adaptive Frequent Item set Mining over Data Streams", Journal of Information Science and Engineering, Vol. 29, pp. 1001-1020, 2013.

[21] Yunyue Zhu, Dennis Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time", VLDB, Vol. 15, 2002.

[22] Caiyan Dai, Ling Chen, "An Algorithm for Mining Frequent Closed Item sets in Data Stream", Physics Procedia, Vol. 24, PP. 1722 – 1728, 2012