International Journal of Software Engineering and Its Applications
Vol. 9, No. 7 (2015), pp. 181-192
http://dx.doi.org/10.14257/ijseia.2015.9.7.19

Proposing a Algorithm for Finding Repetitive Patterns in Web Dataflow

Mohammad Rostami[1], Somayyeh Ehteshami[2], Fatemeh Yaghoobi[3], Farid Saghari[4] and Samaneh Dezhdar[5]

[1]Member of Young Researchers Club, Islamic Azad University, Dehaghan Branch, Isfahan, Iran
[2]Software Engineering Department, Shiraz, Iran
[3]Software Engineering Department, Isfahan, Iran
[4]Software Engineering Department, Tabriz, Iran
[5]Software Engineering Department, Arak, Iran
[1]mohammadrostami@dehaghan.ac.ir, [2]es_010111@yahoo.com, [3]fatemehyaghobi20@gmail.com, [4]saghari@modiranyar.com, [5]s.dezhdar@gmail.com

Abstract

Today, searching repetitive patterns on data flows is very important. By data flow we mean a type of data which is constantly produced in a very fast and unlimited manner. As a kind of these data we can name the report of clicks in computer networks. A repetitive pattern is a pattern which is available in a significant number of transactions. Finding repetitive patterns in data flows is a new and arguable issue in data mining as data is received in form of fast and continuous flow. Unlike static databases, flow mining faces a lot of problems including single review, requiring unlimited memory and high rate of input data.

A common way of searching repetitive patterns is the excess check of data which requires to be saved in memory. In addition, according to the features of data flows i.e. unlimited and fast production, it is not possible to save them in memory and hence techniques are needed which are able to process them online and find repetitive patterns. One of the most popular relative techniques is using sliding windows. It's advantage is reduction of the consumed memory and increase in search speed.

In this paper, a new vertical display and an algorithm based on pins, called DBP-BA, are proposed to find repetitive patters in data flows. Since this new display without any additional task has a compact form, the proposed algorithm has a better performance than similar ones in terms of consumed memory and processing time. On the other hand, experiments support this matter.

Keywords: *data flows, sliding windows, pin, repetitive data pen set*

1. Introduction

Among applications of data mining, one is to find important and frequent item sets in a huge amount of different data such as data streams. They are unlimited chains of data being created continuously at a very high speed. If an item set's number of occurrence exceeds a specific limit it is identified to be frequent. Frequent item sets mining was first introduced by Agrawal *et al.*, in 1993 [1]. Because this field is highly applied in trades, industries, and different sciences, Apriori basic algorithm was developed in 1994 [2]. By repeated data mining, this algorithm finds frequent item sets. Unfortunately, this is not applicable to data streams because they are generated unlimitedly at an extremely high speed, so they cannot be stored for repeated data mining and as a result such methods could not be used to find frequent item sets in data streams. Therefore, we need ways whereby frequent item sets could be found by one-time data scanning. This problem was solved through presenting a demonstration to store summarized received data [13-15]. This algorithm mines frequent item sets by keeping part of the latest received data and storing them as bit-based representation. In this new presentation, effects of transactions, where a data item is disappearing are removed from the stored summarized data leading to decreased memory use; besides, lower actions are performed to find a set of data items and consequently computation speed increases [3].

In recent years considering different applications of data streams, methods have been developed to extract these patterns at significant speed and accuracy [11]. Such methods should scan each data element just once and use low memory as well. Also, they should process newly generated elements very quickly, otherwise due to the high speed of data production input data processing would no more be possible and they would be lost [4].

Research Objective

This study aims at introducing a new pattern to mine data streams by using sliding window algorithm which is better than the previous methods in terms of efficiency and memory. This algorithm uses a dynamic data structure to decrease memory use.

2. Basic Definitions

Suppose I = {i1, i2, …, im} is the item set and DS = {*T*1, *T*2, …, *T*n} is a data stream. A T transaction is a set of items such that $T \subset I$ and every transaction corresponds to a unique transaction Tid (Transaction ID). Take pattern $X \subset I$ which is a set of I items; it is said that T transaction includes X if $X \subset T$. X support shown as supp(x) is the number of transactions including X. A set of data items repeated as much as a specific fraction of DS size (including s% of DS transactions) is called frequent [5, 6, 20].

Take sliding window W on DS data stream including |W| of the recent transaction; at transactions entry, the window moves forward, new transactions enter it through the front side and old ones exit through the end; this is called sliding [4,7-10]. If N is the number of the last input transaction then W window is defined as WN – w + 1 = {TN – w + 1, TN – w + 2, …, TN} where N – w +1 is the number of the current window. Considering this definition, a set of X data items is said to be frequent when supp(X) > s × |W|. By using this technique and with s threshold we can obtain the set of repeated data with least memory and computation without having to mine the entire input stream. Another thing to be mentioned is that instead of updating the window at every transaction entry, we can do this collectively, *i.e.*, storing input transactions in a buffer and updating the window when the buffer is full [10,11,16,17].

3. A Review on Algorithms of Data Streams Mining

Mfi-Trans SW = Mining Frequent Item sets within a Transaction-sensitive Sliding Window;

This algorithm, introduced based on Apriori algorithm in 2009, mines a complete set of repeated items in the current window and uses bit-based representation for data items such that it considers a chain of bits for any data item. The existence and absence of a data item in a transaction is shown respectively by 1 and 0. To add a new transaction and delete an old one from the window, it uses bit-based left shift [8].

This method consists of three phases: the 1st one relates to windows initialization and obtaining the bit sequence of any data items; the length of each bit sequence equals the window length. In the next phase that is window sliding phase old transactions are deleted and new ones are added and for the sliding to be seen in the bit sequence of data items, the bit sequence is also left-shifted. Frequent patterns are discovered in the 3rd phase.

Although this algorithm tries to speed up mining by bit operations, it lacks good time efficiency because the same limitation of Apriori algorithm, *i.e.*, production and testing of candid items set exists here, that is to say when the number of items is high and we have a set of long repeated items there is a high volume of candid items in the said algorithm. On the other hand, since this algorithm keeps information both for repeat existence and lack, it requires high memory capacity, that is a fixed chain as long as the window length is considered for any data item disregarding how many repeats it had in the window.

Eclat Algorithm

This is a depth-first method to find repeated items, functioning based on database vertical view [18]. By database vertical view we mean for any item in the database a list of its transactions is kept (tid-list); the data item support equals the length of this list; then the support of any of data items set is found by intersecting on these lists and anassumed tree is made by a depth-first search whereby repeated data items are found.

LDS[1] Algorithm [19]

As Eclat algorithm reads the data one entire round and then performs mining operations, it may be used to mine data streams as well; however, its deficiency is that it is not suitable for dispersed data items. To solve this problem another method was introduced that uses a complementary list instead of keeping a usual list for any data items set, but these lists also have problems *i.e.*, they are not suitable for dense data sets. That is why Mr. Deypir introduced a new method called LDS which removed the two problems. Like Eclat method, a list of transaction number is kept for any data item in LDS method and if the length of this list is greater than half of the window length, its complementary list is kept instead of the usual one [19]. By complementary, we mean the list of transaction numbers in the window where they do not exist on the usual list of the respective data item; as a result, the shortest list is always kept for any data item and finally, mining operations are performed on the current window upon request of the user.

MFPN[2] Algorithm

Introduced by Jin *et al.*, this is among developed forms of Eclat algorithm [12]. MFPN is categorized in Pin algorithms, *i.e.*, it performs window upgrading based on a Pin. This algorithm is used to mine network data.

Here, instead of transactions numbering in the window they are numbered in each Pin and following the addition of the new Pin, operations to make transactions list as explained in Eclat method are performed for items in this Pin, not with numbers in the window but with numbers in any Pin.

Simultaneous with the new Pin preparation, the algorithm performs mining and finds the set of repeated data items; so upon request of the user the results are ready to be presented.

4. The Proposed Algorithm

The proposed algorithm uses the sliding window (SW) to mine data streams. Since in most applications of data streams recent data are of greater importance than old data a fraction of recent data is considered as a window and mining is performed on them. This window has a fixed size—equaling a fixed number of transactions—and following a new

[1]List-based Data Stream mining algorithm
[2]Mining Frequent Patterns form Network flows

transaction it is updated, but because transactions production rate is very high it has to be updated many times and with very high speed; this imposes overload and decreases system efficiency. To solve this problem, any window (or generally the input data) is divided into some sub-windows with fixed size called Pin; window updating is performed whenever new transactions appear within the size of the Pin. The updating process is performed following removal of the oldest Pin from the window and addition of the new one (Figure 1).

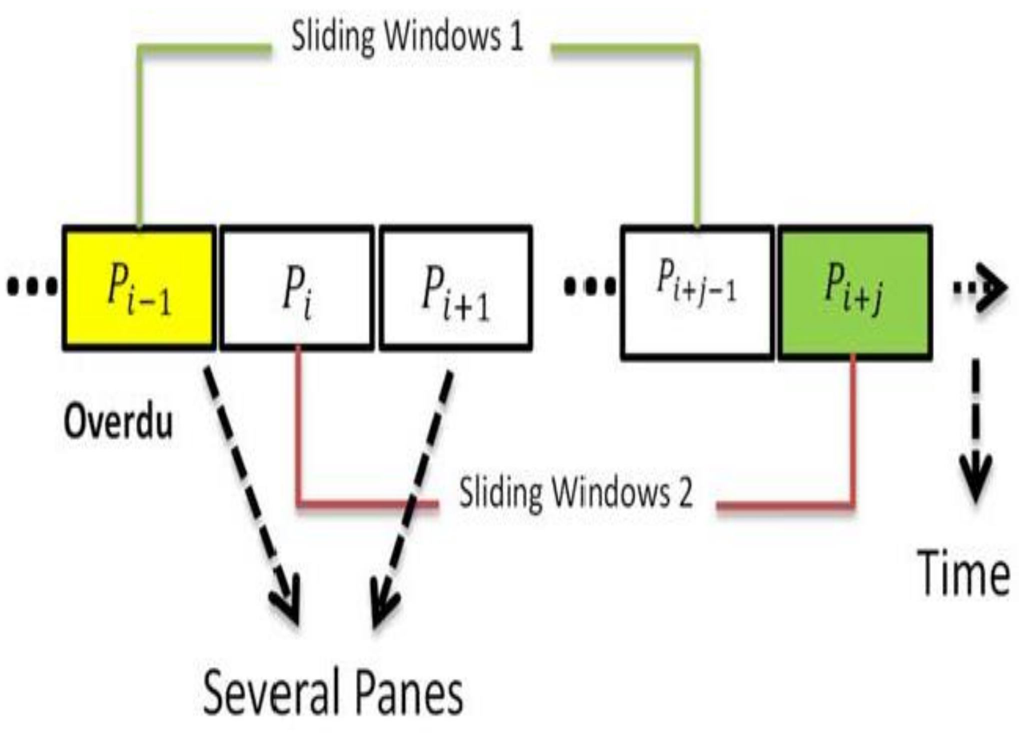

Figure 1. Window Updating by using Pin

The proposed algorithm consists of three phases:

1st phase: window initialization

This phase includes SW initialization and finding repeated data items. The process starts with monitoring the input stream and following every transaction entry its data item is obtained, the related dynamic vector list is made, and simultaneously whenever a transaction appears within the size of a Pin it [the Pin] is added to the window.

2nd phase: window sliding

Upon appearing a new Pin, *i.e.*, production of transactions with size of a Pin, the window has to be updated, that is a new Pin should be added to the window, the oldest Pin (the outdated one) be removed from the window, data items of the new Pin be added to those of the previous one, and results of Pins addition and removal be applied to the found data items.

3rd phase: production of repeated data item set

The set of repeated data items are kept in a tree and by depth-first searching the tree (as explained in Eclat method) the set of repeated data items is made.

5. DPB-BA Algorithm Pseudo Code

Data stream is the input of this algorithm and the output is the set of repeated data items. In the initialization phase, as long as the window is not full input transactions are added to the window as soon as they enter and simultaneously repeated data items are searched and stored.

```
Input: DataSet in Sliding window DB, and the Minimum support $\xi$
Output: The complete set of frequent p patterns

Method:

(1) Scan SW to find L[1] = {frequent 1-patterns} and collect Ti[1] =
    {Dynamic bit Array list}
(2) for(k=2;L[k-1]!=∅; k++){
(3)     for all p ∈ L[k-1] and q ∈ L[k-1]
(4)         if(p[1]=q[1], … , p[k-2]=q[k-2],p[k-1]<q[k-1]){
(5)             c = p[1], p[2],…, p[k-1],q[k-1]; //Candidate k-pattern
(6)             c.DBV = p.DBA and q.DBA;
(7)             if(c.support > |SW| × $\xi$) {
(8)                 L[k] = L[k] ∪ {c}
(9)                 Ti[K] = Ti[K] ∪ {c.DBA}
(10)            }
(11)        }
(12)    }
(13)    Delete Ti[k-1]
(14) }
(15) Results = L[1]∪ L[2]∪… ∪ L[k]
```

The Proposed Algorithm Pseudo Code

6. Results and Assessment

Data set T40I10D100K, artificially generated by IBM data generator software, was used to test. Considering similar distribution in artificial data, data set KOSARAK, related to web click, was also used.

Considering that the proposed method is a Pin one, it was compared with MFPN [10], LDS [9], and MFI-Trans SW [5] methods to examine its efficiency. In reality, a data stream is an unlimited sequence that could extend forever, but in the tests data streams are simulated by reading from data sets mentioned above and the window sliding process continues as far as they go on. Anyway, due to the high speed of data entry into data streams, any comparison between different methods of finding repeated patterns should take into consideration two traits: 1. memory used, 2. time elapsed to perform. This is why the amount of used memory and elapsed performance time have been examined in the tests. In the 1sttest, the amount of used memory to different sizes of the window was examined. In MFI-Trans SW, the used memory was obtained by summing up the length of bit strings of all the items because the length of a bit string for a data item in the window exactly equals the number of transactions in the window. The results are shown in Figures 2 & 3. As seen in these figures, memory use increases with the window size and this natural because greater volume of data has to be examined. However, it is generally observed that memory use of the introduced method is lower than that of the other ones because a different coding was used in this method such that extra data of no use were disregarded; besides, the coding of this method is of bit type which intrinsically uses lower memory.

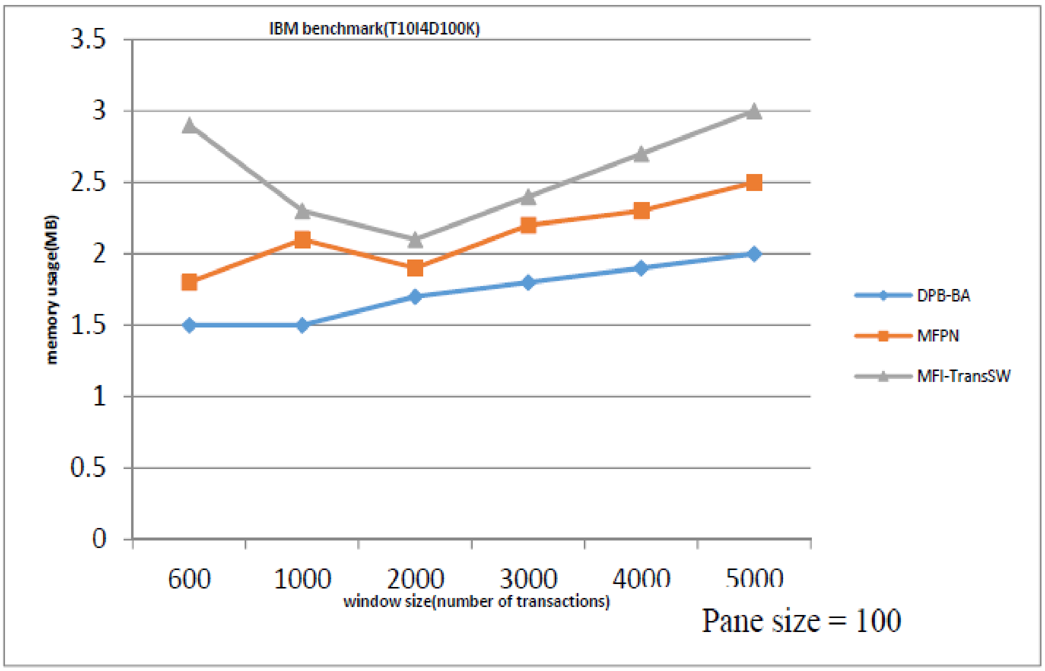

Figure 2. Comparison of Memory Use on IBM Data

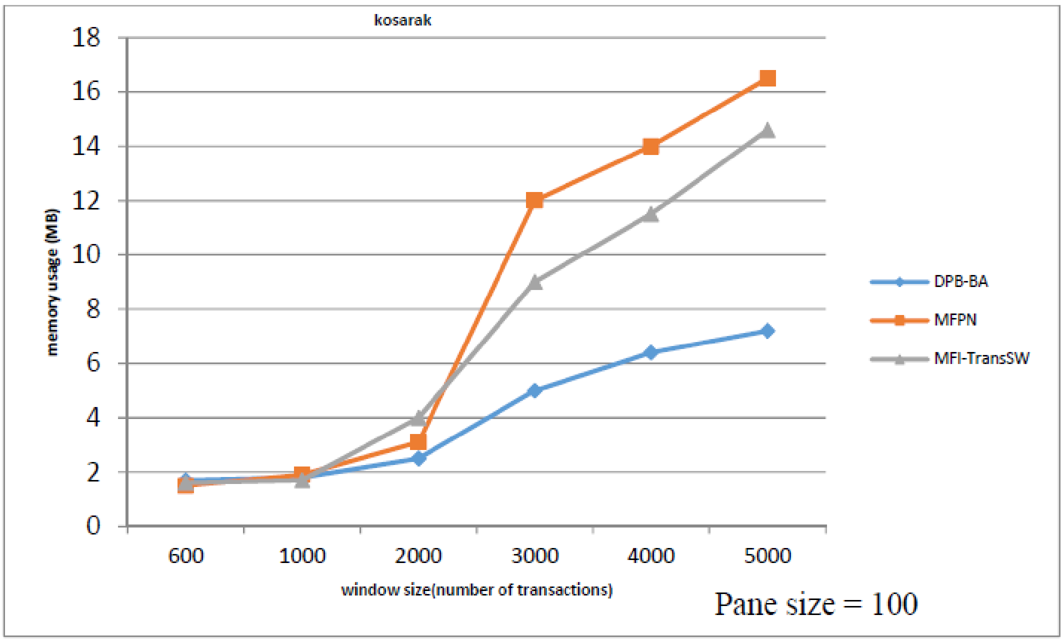

Figure 3. Comparison of Memory Use on KOSARAK Data

DPB-BA method uses a dynamic Pin bit list for every data item and because a dynamic Pin list—also efficient as to memory—has been used the introduced method functions better than the other ones in terms of memory use. In the next test, performance time of the proposed method was compared to that of MFPN and Mfi-Trans SW methods and to window with different sizes.

The proposed method is faster than the other ones and as the window size increases the efficiency of the method is more obvious. This method is efficient because items are represented efficiently. The bigger is the size of the window, the more frequent item sets are detected and kept. Because an efficient representation, consuming the lowest memory, is used to keep any frequent item set in this method lower computation is performed to find frequent item sets, but there are much more computation in the other methods, requiring greater performance time. Comparison of the effect of window size on performance time is shown in Figures 4 & 5.

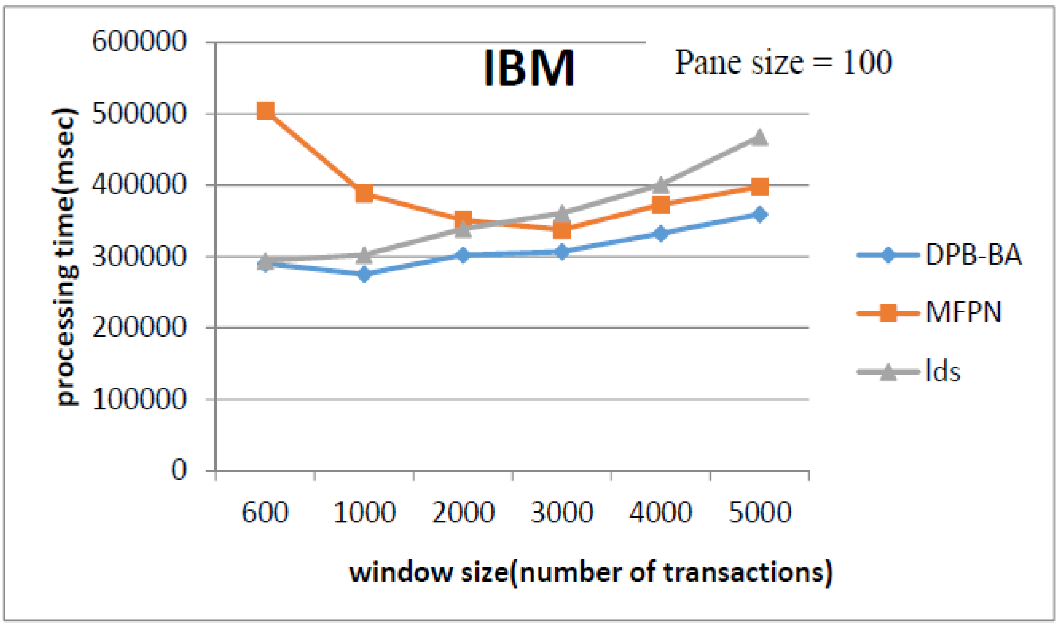

Figure 4. Comparison of the Effect of Window Size on Performance Time by using IBM Data

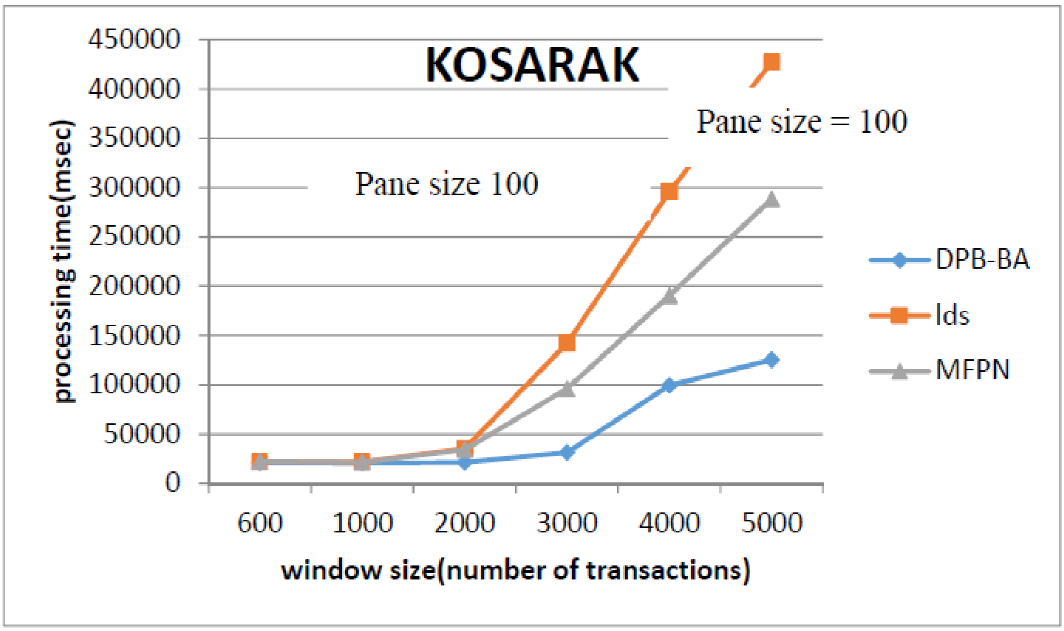

Figure 5. Comparison of the Effect of Window Size on Performance Time by using KOSARAK Data

The higher is the rate of minimum support, the lower repeated data are detected and as a result, as shown in Figures 6 & 7, in the next phases of the algorithm lower data should be examined to build the tree in the 3rd phase and so time use of the algorithm decreases. Moreover, considering the presentation of any data items set, lower computation is required leading to decreased time use.

Performance time of DPB-BA algorithm is lower than that of the other two ones, but as you see generally as the rate of minimum support increases performance time decreases in all the algorithms.

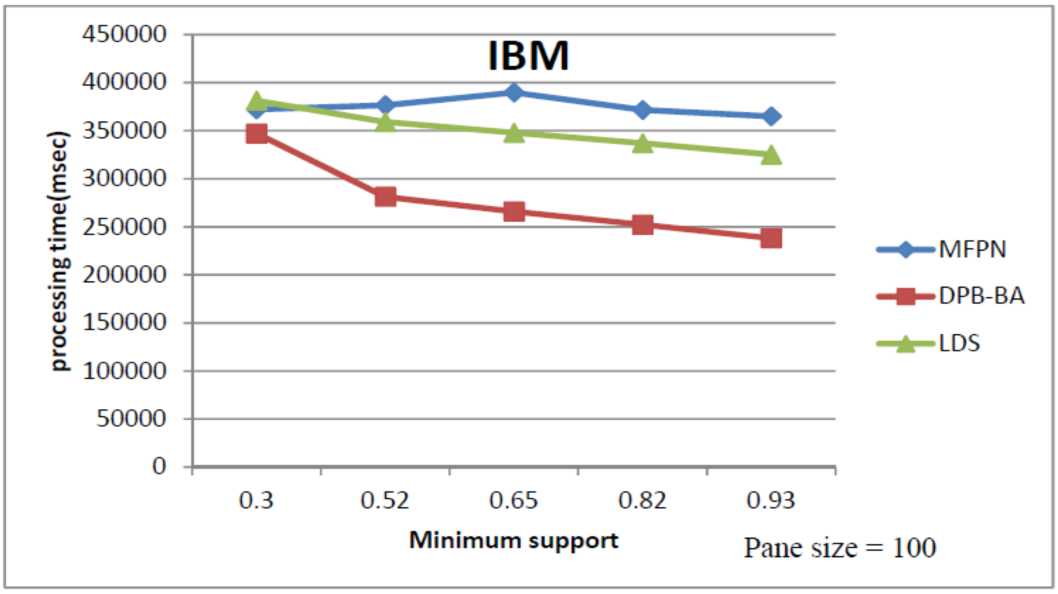

Figure 6. The Effect of Minimum Support Rate on Algorithm Performance Time with IBM Data

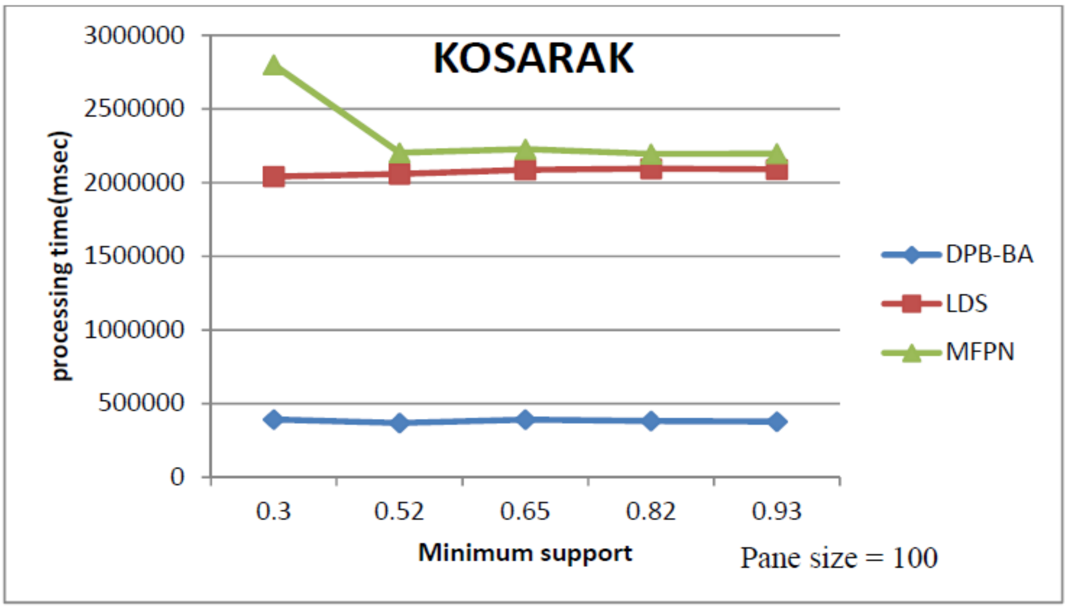

Figure 7. The Effect of Minimum Support Rate on Algorithm Performance Time with KOSARAK Data

As the window size increases performance time changes, but because the coding presented by the proposed algorithm is compressed—refraining from keeping extra items and performing no extra processing for this—its performance time is generally shorter than the other algorithms. As the Pin size increases necessary computation increases as well, because the number of transactions to be added to the window increases. This is shown in Figures 8 & 9.

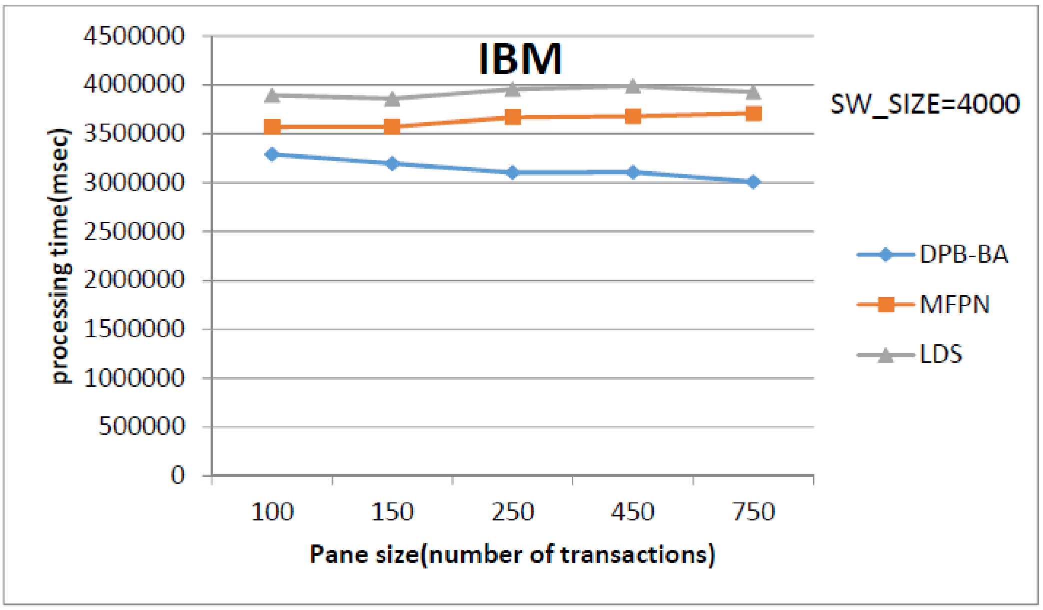

Figure 8. Comparison of Performance Time among Different Pins by using IBM Data

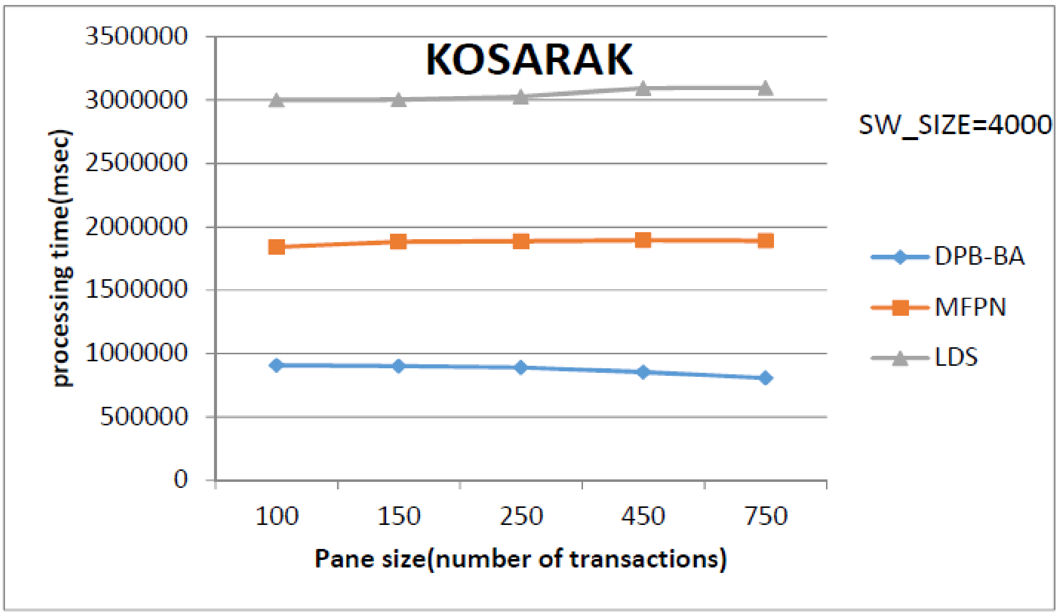

Figure 9. Comparison of Performance Time among Different Pins by using KOSARAK Data

7. Conclusion

Data streams are a new pattern of data recently noticed a lot. Because we deal with huge volume of data they cannot be stored in memory, so we need methods to immediately process them and reveal their embedded knowledge; this is a big challenge. Finding frequent item sets is an issue of attention for many scientists. This has been highly discussed in relation to static databases but still to be argued with respect to data streams. Today, data generation rate has increased with achieved progresses, hence previous algorithms of data mining are no more usable because they use a lot of memory, so they cannot process input data and as a result considerable amount of input data are lost. It is noticed that the presented algorithms are not adequately capable in reality and give approximate results, so not accurate enough in many cases. Therefore, we need algorithms being accurate in this regard. In this research, we studied the presented algorithms to mine repeated patterns in data streams. They were classified as to different aspects, one of which was the time to perform data mining that could be done at any moment or be postponed by when the user requested to resume; and naturally the former is faster.

Another classification related to the window used for them which in turn were divided in two subclasses. This research presented a new method to mine repeated patterns in data streams. It uses the sliding window pattern and remarkably decreases the performance time and memory use of the algorithm by a new presentation. This algorithm is among those which perform data mining upon request of the user. It was seen that the introduced method is superior to the other ones in performance time and memory use.

References

[1] R. Agrawal, T. Imielinski and A. N. Swami, "Mining association rules between sets of items in large databases", Proc. ACM SIGMOD Conf. Manag. Data, **(1993)**, pp. 207-216.
[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", Proc. VLDB Int. Conf. Very Large Databases, **(1994)**, pp. 487-499.
[3] S. Pramod and O. P. Vyas, "Survey on Frequent Item set Mining Algorithms", Int. J. Comput. Appl., vol. 1, no. 15, **(2010)**, pp. 86-91.

[4] C.-H. Lee, C. Lin and M. Chen, “Sliding window filtering: an efficient method for incremental mining on a time-variant database”, Inf. Syst., vol. 30, no. 3, **(2005)** May, pp. 227-244.
[5] H. Li, S. Lee and M. Shan, “An efficient algorithm for mining frequent itemsets over the entire history of data streams”, Proc. First Int. …, **(2004)**.
[6] M. Zaki and K. Gouda, “Fast vertical mining using diffsets”, Conf. Knowl. Discov. data Min., **(2003)**, pp. 326-335.
[7] J. H. Chang and W. S. Lee, “estWin: Online data stream mining of recent frequent itemsets by sliding window method”, J. Inf. Sci., vol. 31, **(2005)**, pp. 76-90.
[8] H.-F. Li and S.-Y. Lee, “Mining frequent itemsets over data streams using efficient window sliding techniques”, Expert Syst. Appl., vol. 36, no. 2, **(2009)** March, pp. 1466-1477.
[9] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong and Y.-K. Lee, “Sliding window-based frequent pattern mining over data streams”, Inf. Sci. (Ny)., vol. 179, no. 22, **(2009)** November, pp. 3843-3865.
[10] H. Liu, S. Lin, J. Qiao, G. Yu and K. Lu, “An Efficient Frequent Pattern Mining Algorithm for Data Stream”, 2008 Int. Conf. Intell. Comput. Technol. Autom., **(2008)** October, pp. 757-761.
[11] C. Leung, F. Jiang and Y. Hayduk, “A landmark-model based system for mining frequent patterns from uncertain data streams”, Proceedings of the 15th Symposium on... “IDEAS”, no. i, **(2011)**, pp. 249-250.
[12] X. Li and Z.-H. Deng, “Mining frequent patterns from network flows for monitoring network”, Expert Syst. Appl., vol. 37, no. 12, **(2010)** December, pp. 8850-8860.
[13] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J. L. Larriba-Pey, “Dynamic adaptive data structures for monitoring data streams”, Data Knowl. Eng., vol. 66, no. 1, pp. 92-115, **(2008)** July.
[14] S. Bashir and A. Baig, “Ramp: Fast Frequent Itemset Mining with Efficient Bit-Vector Projection Technique”, arXiv Prepr. arXiv0904.3316, **(2009)**, pp. 1-37.
[15] A. Bifet, “Adaptive learning and mining for data streams and frequent patterns”, ACM SIGKDD Explor. Newsl., vol. 11, no. 1, **(2009)** November, pp. 55.
[16] A. Yarlagadda, J. V. R. Murthy and M. H. M. K. Prasad, “Estimating Correlation Dimension Using Multi Layered Grid and Damped Window Model Over Data Streams”, Procedia Technol., vol. 10, **(2013)**, pp. 797–804.
[17] D. Basin, F. Klaedtke and E. Zălinescu, “Greedily computing associative aggregations on sliding windows”, Inf. Process. Lett., vol. 115, no. 2, **(2015)** February, pp. 186-192.
[18] M. J. Zaki, S. Parthasarathy, M. Ogihara and W. Li, “New Algorithms for Fast Discovery of Association Rules”, Knowl. Discov. Data Min., **(1997)**, pp. 283-286.
[19] M. Deypir, “An Efficient Sliding Window Based Algorithm for Adaptive Frequent Itemset Mining over Data Streams”, J. Inf. …, vol. 1020, **(2013)**, pp. 1001–1020.
[20] M. Deypir and M. H. Sadreddini, “A dynamic layout of sliding window for frequent itemset mining over data streams”, J. Syst. Softw., vol. 85, no. 3, **(2012)** March, pp. 746-759.