ASEAN Journal of Science and Engineering

Journal homepage: http://ejournal.upi.edu/index.php/AJSE/

Frequent Items Mining on Data Streams using Matrix and Scan Reduced Indexing Algorithms

S. Vijayarani[1], *C. Sivamathi*[2, *], *R. Prassanalakshmi*[1]

[1]Department of Computer Science, Bharathiar University, Coimbatore, Tamilnadu, India
[2]Department of Computer Science, PSG College of Arts & Science, Coimbatore, Tamilnadu, India
Correspondence: E-mail: c.sivamathi@gmail.com

ABSTRACT

A data stream is used for handling dynamic databases, in which data can arrive continuously without limit. Association rule mining is a data mining technique, used to find the association between the data items in the databases. To generate association rules, frequent items are to be identified from the transactional database. Normally, in data mining, frequent-item-generation algorithms scan the database multiple times. But this is impossible in data streams because it handles dynamic databases. Hence, there is a need to develop a new algorithm, which reduces the number of database scans. In this work, two new algorithms named Scan-Reduced Indexing and Matrix algorithm are proposed for generating frequent itemsets in data streams. Performances of both algorithms are compared based on the execution time and the number of frequent items generated. Experimental results show that the performance of the Scan-Reduced Indexing algorithm is more efficient than that of the Matrix algorithm.

ARTICLE INFO

Article History:
Submitted/Received 20 Jan 2022
First revised 25 Feb 2022
Accepted 11 Apr 2022
First available online 12 Apr 2022
Publication date 01 Sep 2023

Keyword:
Association rules,
Data streams,
Database scans,
Frequent items,
Matrix algorithm,
Scan-reduced indexing algorithm.

1. INTRODUCTION

In recent years, advances in information technology have led to large flows of data. In many applications, these large volumes of data must be mined for retrieving unknown and interesting patterns. The process of arriving at continuous dynamic data is known as data streams (Chan, 1998). Traditional data mining algorithms cannot be applied directly in data streams, since those algorithms can handle only static databases. Hence, there is a need to develop new algorithms and techniques for data stream mining to handle the continuous arrival of data. Frequent pattern mining is a core data mining operation. Frequent pattern mining focuses on discovering frequently occurring patterns in a dataset. The patterns can be itemsets or sequences or even subtrees or subgraphs depending on the type of dataset. Frequent pattern mining has become a basic task for many other data mining techniques like association rule mining, classification, and clustering. In recent times, mining frequent patterns over data streams have attracted a lot of research interest.

The streaming algorithm is a method of managing the flow of data by examining the arriving items once and then discarding them (Basu, 1998). Datastream algorithms are responsible for managing continuously generated data, even if the volume of data is too large for memory. Multi-scan algorithms, which scan a database more than once, are not suitable for data streams. This is because of bounded memory, high-speed data arrival, and timely data processing. Hence, data streaming algorithms should be such that it applies only a one-scan technique. Frequent itemsets mining in data streams is an important technique. It has a wide range of emerging applications (Agrawal & Srikant, 1995; Anand *et al*., 1998) such as weblog and click-stream mining, network traffic analysis, trend analysis and fraud detection in telecommunications data, e-business, stock market analysis, and sensor networks. Hence, it is necessary to mine frequent patterns in data streams. The main objective of this work is to generate frequent items in data streams with a minimum number of scans. To conduct this study, two new algorithms are proposed. They are the Matrix algorithm and Scan Reduced Indexing Algorithm. These algorithms scan the database only once and hence they are highly suitable for mining data streams.

2. LITERATURE REVIEW

2.1. Related Works

Agrawal and Srikant were the first researchers to propose the Apriori algorithm. Two main processes are executed in the apriori algorithm: the first is candidate generation process, in which the support count of the corresponding items is calculated by scanning the transactional database and the second is large itemset generation, which is generated by pruning those candidate itemsets, which d have support count less than the minimum threshold. These processes are iteratively repeated until candidate itemsets or large itemsets become empty. Some researchers proposed the FP-growth method (Han *et al*., 2004) that mines all frequent itemsets without candidate generation. The FP-growth method follows the divide-and-conquer strategy to generate all frequent itemsets. The method uses a combination of the vertical and horizontal database format to store the dataset in the memory. FP-growth scans the database and generates all frequent items, reorders the items in descending order of their support. FP tree is constructed and items of transactions in the dataset are inserted into the FP-tree. From the FP-tree, frequent items, conditional pattern base, and conditional FP-tree of each frequent item are mined. Mining is performed recursively.

Many algorithms that have been proposed are only applicable to relational data. It is important to discover frequent patterns and association rules in XML data. Ding and Sundarraj retrieved frequent patterns and association rules in XML data. The challenge was the complexity of the structure in XML. The authors discussed the challenges and other important aspects of handling XML data. We insight into solutions and future research directions in association rule mining. Some researchers reported Equivalence CLAss Transformation (ECLAT) algorithm (2003). ECLAT uses the vertical-database representation and tidset-intersection method to determine the support of an itemset. The Apriori and FP-growth methods generate frequent itemsets from a set of transactions in horizontal database representation (tid- itemset), where tid is a distinct transaction identifier and itemset is the set of items that belongs to the transaction. Mining frequent itemsets can also be performed with a dataset presented in vertical data format (item: tidset). Some researchers discussed (Mittal *et al*., 2015) mining frequent itemset in the transactional database (2015). The objective of this comparative analysis was to reduce the number of scans and improve efficiency. The strength and weaknesses of Apriori, DHP, Partitioning, Sampling, DIC, H-mine, FP-growth, and Eclat algorithms were analyzed. Finally, the authors observed that the FP-growth algorithm worked better than any other algorithms.

2.2. Frequent Itemset Mining in Data Streams

Let D be a transaction dataset. Let t1, t2, t3….tn be a transaction. Thus, dataset D = {t1, t2, t3……tn}. A set of patterns in D are represented as F. Let c be the function to count the number of occurrences of F in D. It is defined as c: F X T -> N, where T is the set of transactions and N is the set of nonnegative integers. Let f and T be the parameters, such that f F, and T t. The counting function c(*f,*t) gives the occurrence of f in t. The support of a pattern f F in the dataset D is defined as in.

$$\text{Support (f)} = \sum_{k=0}^{d} \mathrm{I}\big(\mathrm{c}(\mathrm{f}, \mathrm{t}(\mathrm{k}))\big) \tag{1}$$

Here I am the indicator function. The patterns f, are said to be frequent patterns if Support (f) is greater than the minimum support threshold. Frequent pattern mining is a core data mining operation. Mining frequent patterns over data streams have attracted a lot of research interest. Frequent pattern mining in data streams poses more challenges due to high memory and computational costs. Frequent pattern mining focuses on discovering frequently occurring patterns in a dataset. Different types of datasets used for finding frequent patterns are transaction datasets, text datasets, XML datasets, and graph datasets. Based on the type of datasets, the patterns, which can be itemsets, sequences, subtrees, or subgraphs, are identified. In a data stream, transactions arrive continuously and the volume of transactions can be potentially infinite (Goulbourne *et al*., 2000; Griffin & Chen, 1998). A data stream D can be defined as a sequence of transactions [D = (t1, t2,… ti,….tj)] where ti is the ith arrived transaction. Traditionally, window models have been used to process and mine data streams. A window is defined as a subsequence of transactions arrived between the ith and jth transactions. It can be denoted as W[I,j] = (ti, ti+1,….., tj), i ≤ j. In this work also, the window model is applied, i.e., the continuous arrival of data is split into different partitions and each partition is termed a window. To process and mine data streams, often different window models are used.

There are three categories of stream data processing models: They are Landmark window model, the Damped window model, and the Sliding window model. A window, W, can be either time-based or count-based, and it can be either a landmark window or a sliding window. A window W is said to be time-based only if it consists of a sequence of fixed-length time units, i.e. a sequence of transactions per hour or sequence of transactions for two hours, etc. In this model, the number of transactions may vary within each time unit. A window W is said to be count-based if it consists of a sequence of batches, where each batch consists of an equal number of transactions. In this model, the number of transactions in each batch is the same. Similarly, a window can be a landmark window or a sliding window. The window that maintains all the transactions from a specified time in the past to the present moment is called the landmark window. For example, a landmark window model consists of a sequence of transactions from January 2017 to the present date. A landmark window does not distinguish recently arrived items from older ones. A sliding window model, on the other hand, considers the data from arrival to a certain limit in a time interval. For example, a sliding window model consists of a sequence of transactions in the past 12 months. This model, unlike any of the rest, gives more importance to recently arrived transactions.

2.3. Issues in Frequent Itemset Mining Over Data Streams:

The following are some of the issues in data streams (Ha & Park, 1998; Han *et al*., 2004; Han & Fu, 1999; Kaski *et al*., 1998).

1. Since the streaming data is passed only once, multiple scans of the database are not possible. Hence, frequent itemset mining algorithms in the data stream must be a single scan algorithm.
2. In a data stream, since data arrives continuously and the amount of data is also abundant, it is a challenge to keep the entire stream in the main memory or even in a secondary storage area.
3. Mining streams require fast, real-time processing to keep up with the high data arrival rate and mining results are expected to be available within a short response time.

3. METHODS

Considering the above factors, this work has proposed two new algorithms, namely the Scan-Reduced Indexing algorithm and Matrix algorithm to retrieve high frequent items from data streams with the minimum number of scans. Both algorithms use the window concept and scan the database only once. Hence, they are well suited for data streams.

3.1 Matrix Algorithm

The assumption is that the data set is divided into many windows and they are considered one by one. Each window has many transactions and each transaction has many items. In this algorithm, a matrix is created. Its size is l x m, where l represents the number of items and m represents the number of transactions; The items in each transaction are considered and placed in a matrix table i.e. matlm =1. Likewise, all the items in each transaction are placed in a matrix table. Here, the database is scanned only once. From this matrix table, the number of occurrences of each item is calculated to verify if the number of occurrences of an item is greater than the threshold. This gives the list of the one-item frequent set. With these itemsets, candidate generation, i.e. 2- itemsets, 3- itemsets, and n-itemsets are generated from this matrix table alone, without scanning the original database. This candidate generation process is repeated until no further generation is made. Finally, from the table, frequent itemsets are retrieved.

Example: Consider the given example. From window 1, T1, T2, T3 T4 and T5 are transactions ids and

1, 2, 3, 4, 5, 6, 7, 8, 9 are items. Sample database from Window1:

T1 - {1,3,5,7,9}

T2 – {2,4,5}
T3 – {3,7,9}
T4 – {1,2,3,4} T5 – {5,6,7}

Step 1 – Matrix table: In this step, a matrix table is constructed. Transactions T1, T2, T3, T4, and T5 are represented as columns, and items 1,2,3,4,5,6,7,8, and 9 are represented in rows. For example, the items in Transaction 1, T1, are 1,3,5,7,9. In the matrix **Table 1** is put in their respective location. Similarly, for T2, '1' is put in 2,4,5 locations. The same is repeated for all the transactions. The resultant matrix table is shown in **Table 1**.

Step 2: In this step, the number of counts of each item is retrieved from the table. It is then compared with minimum support (Min_Sup). For example, if Min Sup is 2, the items whose count is ≥ 2 are selected. From the above matrix, the items {1,2,3,4,5,7,9} are selected.

Step 3: In this step, the first item, 1, is compared with the remaining items {2,3,4,5,7,9} to generate 2- a candidate itemset. This is shown in the first column of **Table 2.** The transaction, in which it appeared, is also given. For example, 1,2 -> 4 in the table means that the itemset {1,2} appears in transaction t4. Similarly {1,3} -> 1,4 means that the itemset {1,3} appears in t1 and t4. Likewise, the table is constructed for the remaining items and the resultant table is shown in **Table 2**.

Table 1. Matrix table.

Transactions → **Items ↓**	**T1**	**T2**	**T3**	**T4**	**T5**	**counts**
1	1			1		2
2		1		1		2
3	1		1	1		3
4		1		1		2
5	1	1			1	3
6					1	1
7	1		1		1	3
9	1		1			2

Table 2. Candidate 2 itemsets.

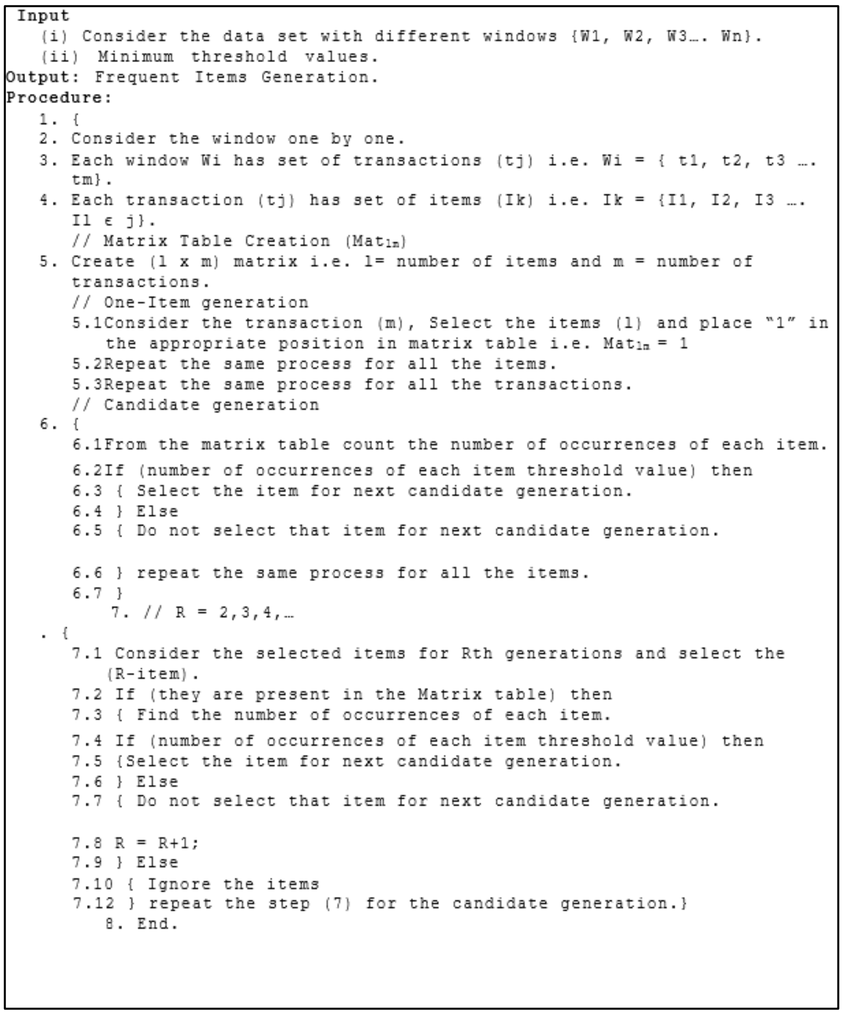
Item1	**Item2**	**Item3**	**Item4**	**Item5**	**Item7**	**Item9**
1,2 -> T_4	2,3 -> T_4	3,4 -> T_4	4,5 ->T_2	5,6 -> T_5	7,9 -> T_1, T_3	-
1,3 -> T_1,T_4	2,4 -> T_2, T_4	3,5 -> T_1	4,7 -> 0	5,7 -> T_1, T_5		
1,4 -> T_4	2,5 -> T_1	3,7 -> T_1,T_3	4,9 -> 0	5,9 -> T_1		
1,5 -> T_1	2,7 -> 0	3,9 -> T_1, T_3				
1,7 -> T_1	2,9 -> 0					
1,9 -> T_1						

Step 4: The number of occurrences of each item set is counted and the count value is verified with minimum support; i.e. the minimum threshold is 2. Hence, the itemsets whose support is greater than or equal to 2 are selected.
1,3 -> T1,T4
2,4 -> T2,T4
3,7 -> T1,T3
3,9 -> T1,T3
5,7 -> T1,T5
7,9 -> T1,T3

Step 5: The resultant candidate 2-itemsets are {1,3}, {2,4}, {3,7}, {3,9}, {5,7}, {7,9}. From this, candidates 3-itemsets are generated. Here, {1,3} itemset is combined with remaining itemset {2,4} {3,7} {3,9} {5,7} {7,9} and resultant frequent 3 itemsets are as follows: {1,3,2}, {1,3,4}, {1,3,7}, {1,3,9}, and {1,3,5}. Similarly, {2,4} is combined with remaining itemset {3,7}, {3,9}, {5,7}, {7,9} to form {2,4,3},{2,4,7}, {2,4,9}, {2,4,5}. The same combination is repeated for all the itemsets. The resultant 3 itemsets are {1,3,2}, (1,3,4), {1,3,7}, {1,3,9}, {1,3,5} {2,4,3}, {2,4,7}, {2,4,9}, {2,4,5}, {3,7,9}, {3,7,5}, {3,9,5}, and {5,7,9}. Step 6: From **Table 3**, it was found that the resultant 3 candidate itemsets are: {3,7,9}, since its support is 2. Step 7: From 3 candidate itemset {3,7,9}, further candidates cannot be generated. Hence the algorithm stops here and the resultant frequent itemsets are {1}, {2}, {3}, {4}, {5}, {7} {9}, {1,3}, {2,4}, {3,7}, {3,9},{5,7}, {7,9}, and {3,7,9}. This is frequent itemsets of window 1. Similarly, frequent itemsets for remaining windows are also generated. Matrix algorithm show in **Figure 1**.

Table 3. Candidate 3- itemsets.

{1,3}	{2,4}	{3,7}	{3,9}	{5,7}	{5,9}
{1,3,2}->T_4	{2,4,3}->T_4	{3,7,9}->T_1,T_3	{3,9,5}->T_1	{5,7,9}->T_1	--
{1,3,4}->T_4	{2,4,7}->0	{3,7,5}->T_1			
{1,3,7}->T_1	{2,4,9}->0				
{1,3,9}->T_1	{2,4,5}->T_2				
{1,3,5}->T_1					

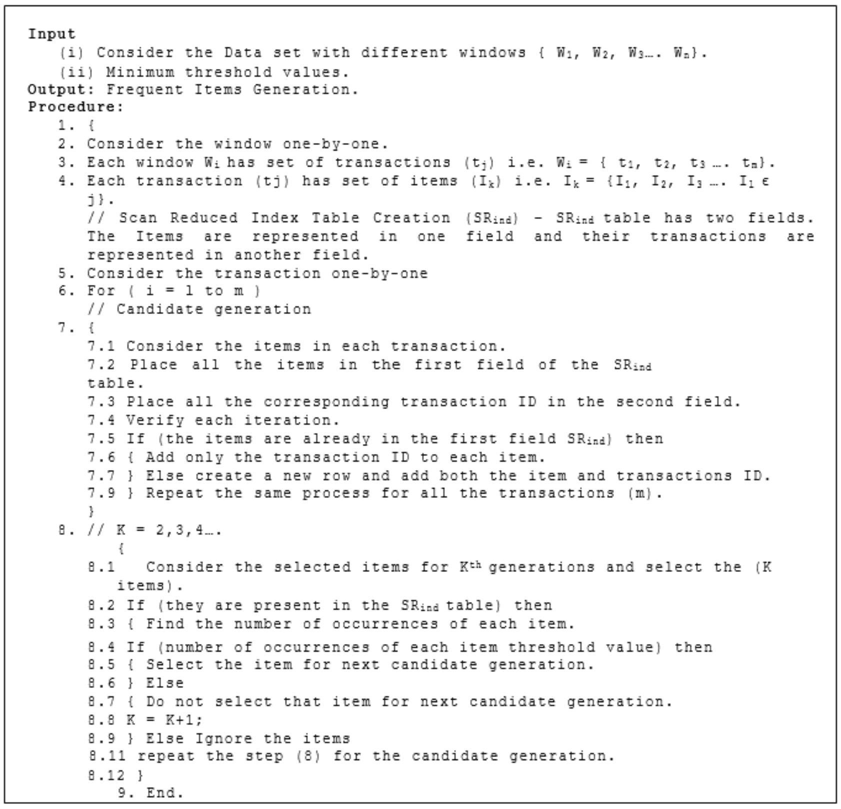


```
 Input
   (i) Consider the data set with different windows {W1, W2, W3…. Wn}.
   (ii) Minimum threshold values.
Output: Frequent Items Generation.
Procedure:
   1. {
   2. Consider the window one by one.
   3. Each window Wi has set of transactions (tj) i.e. Wi = { t1, t2, t3 ….
      tm}.
   4. Each transaction (tj) has set of items (Ik) i.e. Ik = {I1, I2, I3 ….
      Il ϵ j}.
      // Matrix Table Creation (Mat_lm)
   5. Create (l x m) matrix i.e. l= number of items and m = number of
      transactions.
      // One-Item generation
      5.1Consider the transaction (m), Select the items (l) and place "1" in
         the appropriate position in matrix table i.e. Mat_lm = 1
      5.2Repeat the same process for all the items.
      5.3Repeat the same process for all the transactions.
      // Candidate generation
   6. {
      6.1From the matrix table count the number of occurrences of each item.
      6.2If (number of occurrences of each item threshold value) then
      6.3 { Select the item for next candidate generation.
      6.4 } Else
      6.5 { Do not select that item for next candidate generation.

      6.6 } repeat the same process for all the items.
      6.7 }
         7. // R = 2,3,4,…
   . {
      7.1 Consider the selected items for Rth generations and select the
         (R-item).
      7.2 If (they are present in the Matrix table) then
      7.3 { Find the number of occurrences of each item.
      7.4 If (number of occurrences of each item threshold value) then
      7.5 {Select the item for next candidate generation.
      7.6 } Else
      7.7 { Do not select that item for next candidate generation.

      7.8 R = R+1;
      7.9 } Else
      7.10 { Ignore the items
      7.12 } repeat the step (7) for the candidate generation.}
         8. End.
```

Figure 1. Matrix algorithm.

3.2. Scan-Reduced Indexing Algorithm

In the Scan–Reduced Indexing algorithm (see **Figure 2**), an index table having two fields is generated. The first field represents the item and the second field represents the transaction id. All of the items in each transaction are placed in the first field of the index table, and their corresponding transaction ids are placed in the second field. Then, in the next transaction, the algorithm verifies whether the items are already present in the index table or not. If the item is already present in the index table, it only stores the current transaction id in the second field along with the previous one. And, in case it does not, a new entry for that item is made.



```
Input
   (i) Consider the Data set with different windows { $W_1$, $W_2$, $W_3$…. $W_n$}.
   (ii) Minimum threshold values.
Output: Frequent Items Generation.
Procedure:
   1. {
   2. Consider the window one-by-one.
   3. Each window $W_i$ has set of transactions ($t_j$) i.e. $W_i$ = { $t_1$, $t_2$, $t_3$ …. $t_n$}.
   4. Each transaction (tj) has set of items ($I_k$) i.e. $I_k$ = {$I_1$, $I_2$, $I_3$ …. $I_l$ ϵ
      j}.
      // Scan Reduced Index Table Creation ($SR_{ind}$) - $SR_{ind}$ table has two fields.
      The Items are represented in one field and their transactions are
      represented in another field.
   5. Consider the transaction one-by-one
   6. For ( i = 1 to m )
      // Candidate generation
   7. {
      7.1 Consider the items in each transaction.
      7.2 Place all the items in the first field of the $SR_{ind}$
      table.
      7.3 Place all the corresponding transaction ID in the second field.
      7.4 Verify each iteration.
      7.5 If (the items are already in the first field $SR_{ind}$) then
      7.6 { Add only the transaction ID to each item.
      7.7 } Else create a new row and add both the item and transactions ID.
      7.9 } Repeat the same process for all the transactions (m).
      }
   8. // K = 2,3,4….
         {
      8.1   Consider the selected items for $K^{th}$ generations and select the (K
         items).
      8.2 If (they are present in the $SR_{ind}$ table) then
      8.3 { Find the number of occurrences of each item.
      8.4 If (number of occurrences of each item threshold value) then
      8.5 { Select the item for next candidate generation.
      8.6 } Else
      8.7 { Do not select that item for next candidate generation.
      8.8 K = K+1;
      8.9 } Else Ignore the items
      8.11 repeat the step (8) for the candidate generation.
      8.12 }
         9. End.
```

Figure 2. Pseudo code for scan - reduced indexing algorithm.

This is repeated for all the transactions. After creating the index table, the number of occurrences of each item is identified, and then the algorithm verifies if the number of occurrences is greater than threshold or not. If it is greater than the threshold, it is considered for the next candidate generations. Otherwise, it is ignored. This process is repeated for all the candidate generations. *Example:* Consider the following example from window 1. From window 1, T1, T2, T3 T4 and T5 are transactions ids and 1, 2, 3, 4, 5, 6, 7, 8, 9 are items. Sample database from Window1:

t1 - {1,3,5,7,9}
t2 – {2,4,5}
t3 – {3,7,9}
t4 – {1,2,3,4}
t5 – {5,6,7}

Step 1**:** Consider transaction 1. The items of t1 are 1,3,5,7,9. In the index table, create a row for each item, and in the second field enter the transaction id as shown in **Table 4**. The above **Table 4**, is the Index table, which updates the information after reading the first transaction. Now, the second transaction t2=2,4,5 is considered. The first item is 2. The algorithm verifies if 2 is already present in the index table. Since 2 is not present, a new row is created and 2 is placed under the 'Items' column and 2 is placed in the 'Transaction Id' column. Then, the algorithm verifies if 4 is present in the index table. Since 4 is not present here, a new row is created and 4 is placed under the 'Items' field and 2 is placed in the 'Transaction Id' column. The next item in t2 is 5, which is already present in the index table. So, in item 5, the transaction id 2 is added. The updated index table is given in **Table 5**.

Table 4. Index table.

Items	Transaction ids
1	1
3	1
5	1
7	1
9	1

Table 5. Final index table.

Items	Transction Ids
1	**1**
3	**1**
5	**1,2**
7	**1**
9	**1**
2	**2**
4	**2**

Now, consider t3, the items are {3,7,9}. The updated index table is given in **Table 6.**

Table 6. Updated table.

Items	Transaction Ids
1	**1**
3	**1,3**
5	**1,2**
7	**1,3**
9	**1,3**
2	**2**
4	**2**

Now, consider t4 = {1,2,3,4}. The updated table is given in **Table 7**.

Table 7. Updated table after t4.

Items	Transaction Ids
1	**1,4**
3	**1,3,4**
5	**1,2**
7	**1,3**
9	**1,3**
2	**2,4**
4	**2,4**

Now, consider t5={5,6,7}. The updated table is given in **Table 8**.

Table 8. Updated table after t5.

Items	Transaction Ids
1	**1,4**
3	**1,3,4**
5	**1,2,5**
7	**1,3,5**
9	**1,3**
2	**2,4**
4	**2,4**
6	**5**

Step 2: The number of occurrences of each item in the final updated index table is counted. In the above example, the count for item 1 is 2, i.e. item 1 is found in t1 and t4. Likewise, the count for item 3 is 3; the count for item 5 is 3; the count for item 7 is 3; the count for item 9 is 2; the count for item 2 is 2; the count for item 4 is 2 and the count for item 6 is 1. The items whose support is greater than or equal to 2 are selected. Thus, the items selected for next candidate generation are {1,3,5,7,9,2,4}

Step 3: Candidates are generated from the index table, by identifying the common transaction ids of the two items. For example, from the index table given below, the candidate generation, i.e. two itemsets (1,3) and their occurrences are obtained. In the same way, all the possible 2-itemsets and their occurrences are identified. This is shown in **Table 9**. From the **Table 10**, the number of occurrences of each 2-itemsets is counted. The resultant itemsets whose threshold value greater than or equal to 2 are: {1,3} {3,7} {3,9} {5,7} {7,9} {2,4}. Step 4: Now, 3-itemsets are generated from the resultant itemsets and it is shown in **Table 11**. From **Table 11**, it was found that the resultant 3 candidate itemsets are: {3,7,9} since its support is 2.

Step 5: From 3 candidate itemset {3,7,9} additional candidates cannot be generated. Hence, the algorithm stops here and the resultant frequent itemsets are { {1}, {2}, {3}, {4}, {5}, {7} {9}, {1,3}, {2,4}, {3,7}, {3,9},{5,7}, {7,9}, {3,7,9} }. This is frequent itemsets of window 1. Similarly, frequent itemsets for remaining windows are also generated.

Table 9. Itemsets generation.

Items	Transaction Ids
1	1,4
3	1,3,4

Table 10. Candidate Generation – 2-itemsets.

Comparison of item 1 with other items	Item3 with other items	Item5 with other items	Item7 with other items	Item9 with other items	Item 2 with other items	Item 4 with other items
1,3 -> T_1,T_4	3,5 -> T_1	5,7 -> T_1, T_5	7,9 -> T_1, T_3	9,2 -> -	2,4 -> T_2, T_4	--
1,5 -> T_1	3,7 -> T_1, T_3	5,9 -> T_1	7,2 -> -	9,4-> -		
1,7 -> T_1	3,9 -> T_1, T_3	5,2 -> T2,	7,4 ->--			
1,9 -> T_1	3,2 -> T_4	5,4 -> T_2				
1,2 -> T_4	3,4 -> T_4					
1,4 -> T_4						

Table 11. Candidate 3- itemsets.

{1,3}	**{3,7}**	**{3,9}**	**{5,7}**	**{7,9}**	**{2,4}**
{1,3,7}-> T_1	{3,7,9}-> T_1, T_3	{3,9,5}-> T_1	{5,7,9}-> T_1	{7,9,2}->-	--
{1,3,9}- > T_1	{3,7,5}-> T_1	{3,9,2}->-	{5,7,2}->-	{7,9,4}->-	
{1,3,5}-> T_1	{3,7,2}->-	{3,9,4}->-	{5,7,4}->-		
{1,3,2}-> T_4	{3,7,4}->-				
{1,3,4}-> T_4					

4. RESULTS AND DISCUSSION

The proposed algorithms, Matrix Algorithm and Scan-Reduced Indexing Algorithm are implemented in Java with MySQL. The connect data set from the UCI repository was used for experimentation. It consists of 67,558 instances and 48 attributes. In this work, five windows W1, W2, W3, W4, and W5 are created with the size of 1K, 2K, 5K, and 10K. Different threshold values are applied for analyzing the results. The performance is the analysis based on the number of items generated and the execution time. **Table 12** shows the items generated in the matrix algorithm at various thresholds in each window. **Figure 3** shows the graphical representation of the same.

Table 12. Frequent item generation for matrix algorithm.

Window Size	**Threshold (σ) - %**	**1000 Ds**	**2000 Ds**	**5000 Ds**	**10000 Ds**
		Items			
W1	20	83782	89768	90878	98768
	40	87686	86445	82019	90890
	60	83738	89078	87898	93787
W2	20	87483	89372	89281	94892
	40	88374	90078	90189	95922
	60	83721	91083	92837	99282
W3	20	90271	87989	94827	91837
	40	83726	88291	92837	90187
	60	92817	90921	89173	94288
W4	20	81928	89189	88198	92018
	40	90283	90182	99282	97982
	60	82717	93849	94872	93879
W5	20	91871	97928	98272	92873
	40	89173	91891	93872	89278
	60	89837	88972	91981	93874

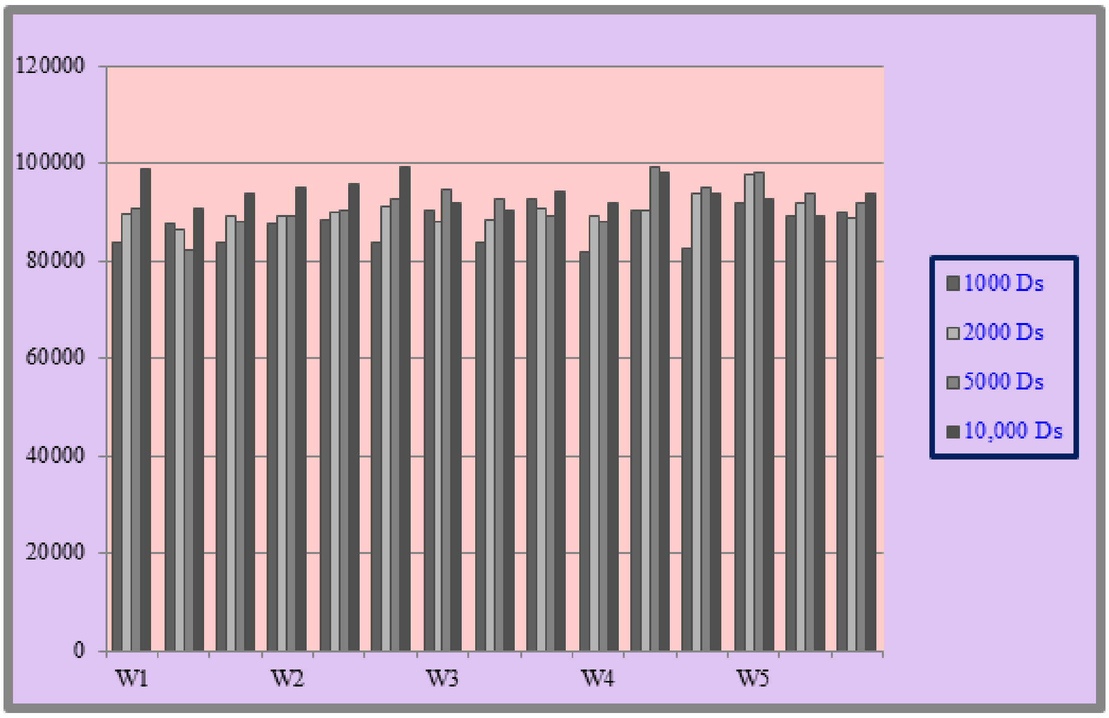


Figure 3. Matrix algorithm for item generations.

Table 13 shows the execution time of the matrix algorithm at various thresholds in each window. **Figure 4** shows the graphical representation of the same. **Table 14** shows the items generated in the SRI algorithm at various thresholds in each window. **Figure 5** shows the graphical representation of the same. **Table 15** shows the execution time of the SRI algorithm and **Figure 6** shows the chart representation of the same.

Table 13. Execution time for matrix algorithm.

Window Size	Threshold (σ) - %	1000 Ds	2000 Ds	5000 Ds	10,000 Ds
		Time (ms)			
W1	20	650	657	655	699
	40	651	650	654	708
	60	653	651	659	719
W2	20	657	659	650	755
	40	654	659	654	790
	60	655	658	659	699
W3	20	655	665	657	701
	40	656	645	645	755
	60	652	658	656	743
W4	20	640	648	659	798
	40	643	655	660	698
	60	650	659	678	712
W5	20	641	660	690	799
	40	666	678	704	795
	60	659	680	714	784

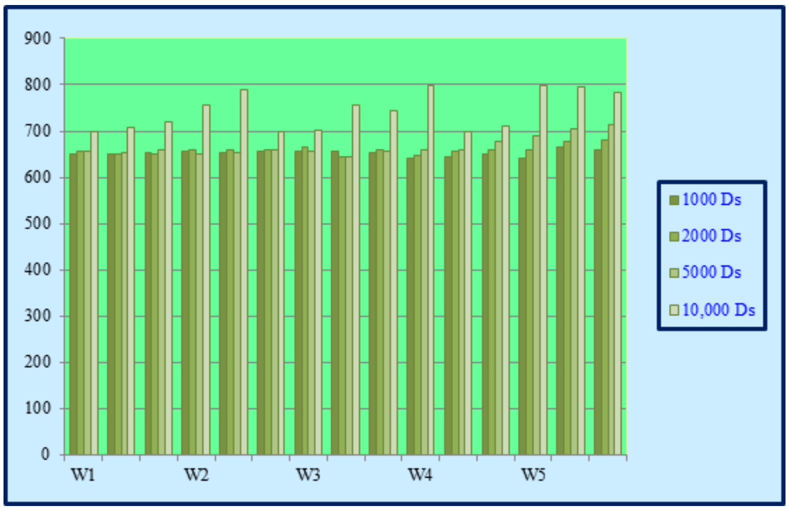


Figure 4. Execution time for matrix algorithm.

Table 14. SRI algorithm for frequent item generation.

Window Size	Threshold (σ) - %	1000 Ds	2000 Ds	5000 Ds	10,000 Ds
		Items			
W1	20	98738	109721	150442	153783
	40	108937	120991	160820	155026
	60	119889	101891	130810	188374
W2	20	99821	129391	182098	197348
	40	109283	148857	182378	199019
	60	129872	194887	176476	183741
W3	20	99271	160938	192083	109834
	40	109890	139874	98783	177913
	60	189828	199830	99804	187408
W4	20	102887	97321	183207	99132
	40	138727	178321	98301	198372
	60	176819	134771	149747	190731
W5	20	130027	143098	199789	176437
	40	147589	185994	152372	134355
	60	159576	199381	158387	198074

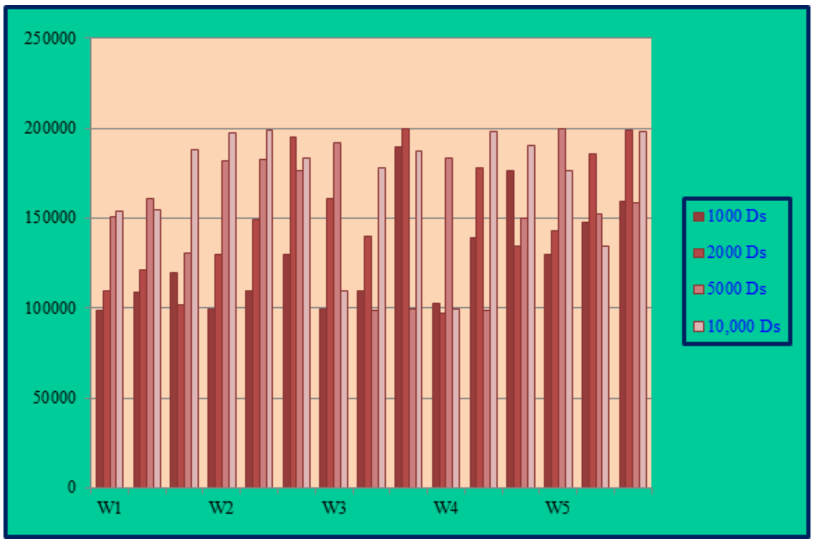


Figure 5. Frequent item generations for sri algorithm.

DOI: https://doi.org/10.17509/ajse.v3i2.45345
p- ISSN 2776-6098 e- ISSN 2776-5938

Table 15. Execution time for sri algorithm.

Window Size	Threshold (σ) - %	1000 Ds	2000 Ds	5000 Ds	10,000 Ds
		Time (ms)			
W1	20	499	501	532	566
	40	493	517	534	578
	60	501	521	544	545
W2	20	514	503	529	567
	40	513	516	519	598
	60	517	515	550	599
W3	20	499	490	527	555
	40	498	498	532	590
	60	505	504	533	581
W4	20	498	513	520	553
	40	497	521	529	567
	60	501	520	550	599
W5	20	495	532	548	573
	40	499	512	599	596
	60	504	519	575	599

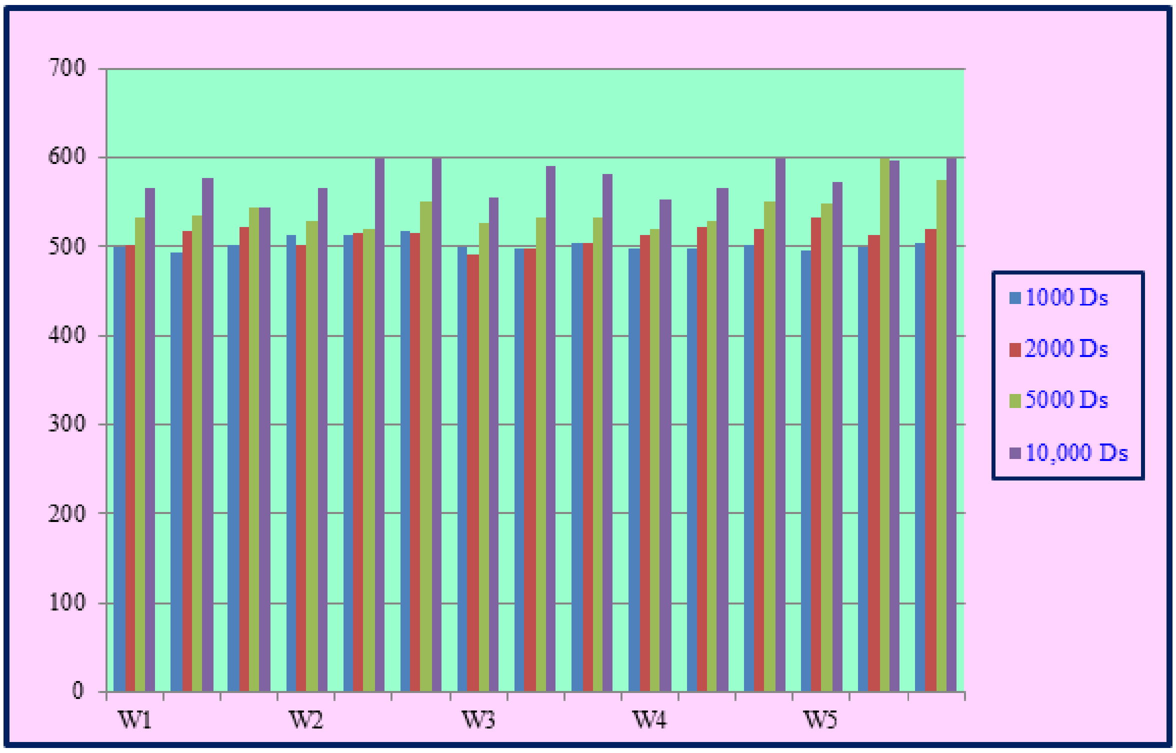


Figure 6. Execution time for sri algorithm.

Now both the algorithms are compared. **Table 15** shows the comparison of the execution time of the SRI and Matrix algorithm. Similarly, table 16 shows the number of items retrieved in both algorithms. **Figures 7** and **8** show a graphical representation of the results. From the results, it was found that the SRI algorithm was more efficient than the Matrix algorithm.

Table 16. Comparison of execution time of ma and sri algorithm.

Window size	Threshold	1000 Ds		2000 Ds		5000 Ds		10,000 Ds	
		MA algorithm	SRI algorithm	MA algorithm	SRI algorithm	MA algorithm	SRI algorithm	MA algorithm	SRI algorithm
W1	20	650	499	657	501	655	532	699	566
	40	651	493	650	517	654	534	708	578
	60	653	501	651	521	659	544	719	545
W2	20	657	514	659	503	650	529	755	567
	40	654	513	659	516	654	519	790	598
	60	655	517	658	515	659	550	699	599
W3	20	655	499	665	490	657	527	701	555
	40	656	498	645	498	645	532	755	590
	60	652	505	658	504	656	533	743	581
W4	20	640	498	648	513	659	520	798	553
	40	643	497	655	521	660	529	698	567
	60	650	501	659	520	678	550	712	599
W5	20	641	495	660	532	690	548	799	573
	40	666	499	678	512	704	599	795	596
	60	659	504	680	519	714	575	784	599

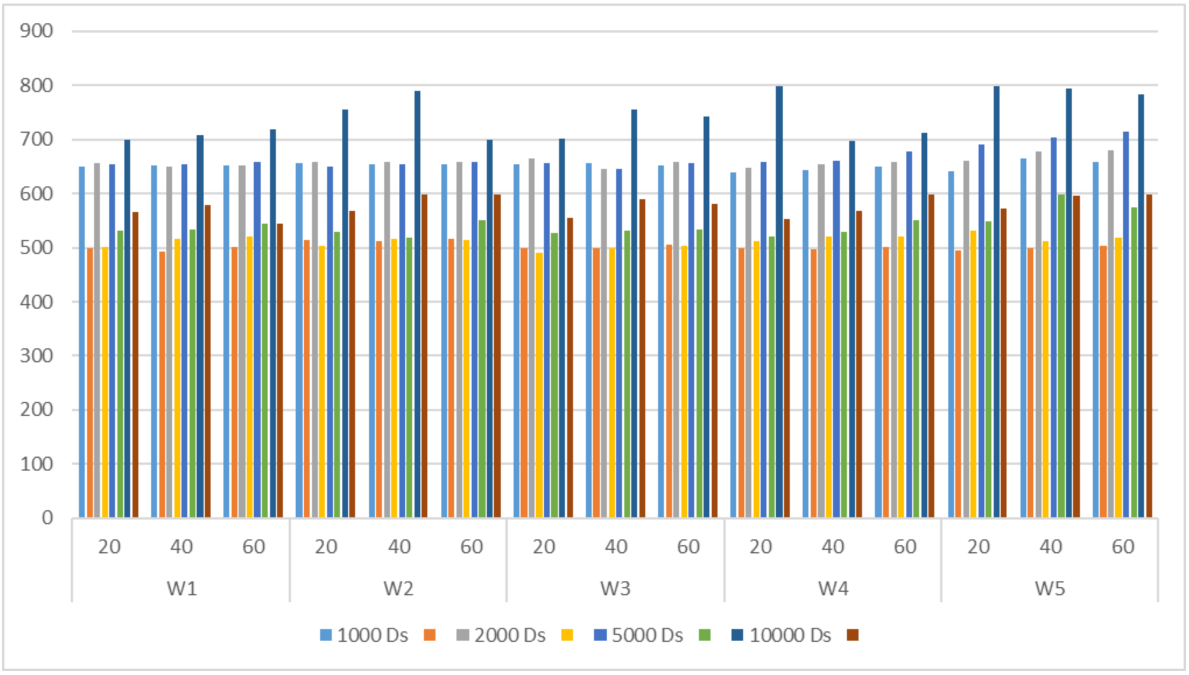


Figure 7. Comparison of execution time in MA and SRI algorithm.

Table 17. Comparison of No. of Items Generated in MA and SRI Algorithm.

Window size	Threshold	1000 Ds		2000 Ds		5000 Ds		10000 Ds	
		MA algorithm	SRI algorithm	MA algorithm	SRI algorithm	MA algorithm	SRI algorithm	MA algorithm	SRI algorithm
W1	20	83782	98738	89768	109721	90878	150442	98768	153783
	40	87686	108937	86445	120991	82019	160820	90890	155026
	60	83738	119889	89078	101891	87898	130810	93787	188374
W2	20	87483	99821	89372	129391	89281	182098	94892	197348
	40	88374	109283	90078	148857	90189	182378	95922	199019
	60	83721	129872	91083	194887	92837	176476	99282	183741
W3	20	90271	99271	87989	160938	94827	192083	91837	109834
	40	83726	109890	88291	139874	92837	98783	90187	177913
	60	92817	189828	90921	199830	89173	99804	94288	187408
W4	20	81928	102887	89189	97321	88198	183207	92018	99132
	40	90283	138727	90182	178321	99282	98301	97982	198372
	60	82717	176819	93849	134771	94872	149747	93879	190731
W5	20	91871	130027	97928	143098	98272	199789	92873	176437
	40	89173	147589	91891	185994	93872	152372	89278	134355
	60	89837	159576	88972	199381	91981	158387	93874	198074

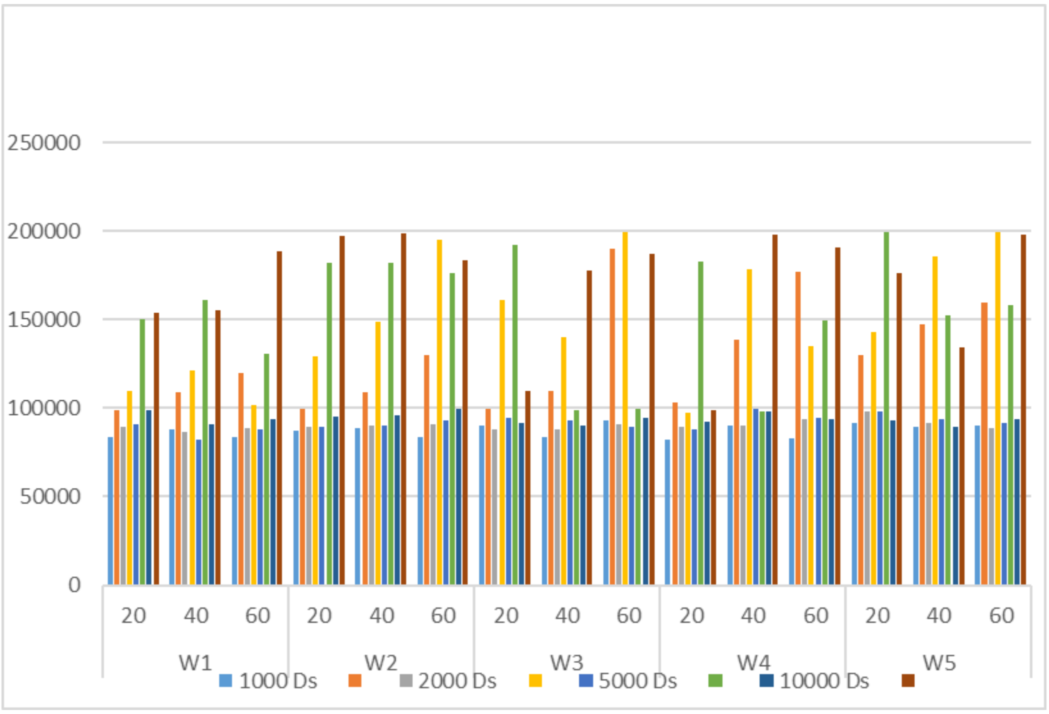


Figure 8. Comparison of the number of items generated in MA and SRI algorithm.

5. CONCLUSION

Association rule mining is the most essential data mining technique to discover hidden patterns from large volumes of data. This research mainly focused on finding an increased number of frequent itemsets in data streams. This research has proposed two algorithms—

MA and SRI algorithm—to find frequent items from data streams. Both of them have used two novel data structures, Matrix structure and Index list, to store the transaction details. This structure can be used for further processing so that database does need not to be scanned again and again. The algorithms scan database only once, and hence, both algorithms are suitable for mining frequent items in data streams. The performance of these algorithms has been assessed based on the number of items generated and the execution time, by using different threshold values for five windows. The result shows that the proposed SRI algorithm performs better than the MA algorithm, and, since the SRI algorithm works faster than the MA algorithm, it generates more frequent items than the MA algorithm.

6. AUTHORS' NOTE

The authors declare that there is no conflict of interest regarding the publication of this article. The authors confirmed that the paper was free of plagiarism.

7. REFERENCES

Agrawal, R., and Srikant, R. (1994, September). Fast algorithms for mining association rules. *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, *1215*, 487-499.

Anand, S. S., Patrick, A. R., Hughes, J. G., and Bell, D. A. (1998). A data mining methodology for cross-sales. *Knowledge-Based Systems*, *10*(7), 449-461.

Basu, A. (1998). Perspectives on operations research in data and knowledge management. *European Journal of Operational Research*, *111*(1), 1-14.

Chan, C. C. (1998). A rough set approach to attribute generalization in data mining. *Information Sciences*, *107*(1-4), 169-176.

Goulbourne, G., Coenen, F., and Leng, P. (2000). Algorithms for computing association rules using a partial- support tree. *Knowledge-BasedSystems*, *13*(2-3), 141-149.

Griffin, G., and Chen, Z. (1998). Rough set extension of Tcl for data mining. *Knowledge-Based Systems*, *11*(3-4), 249-253.

Ha, S. H., and Park, S. C. (1998). Application of data mining tools to hotel data mart on the Intranet for database marketing. *Expert Systems with Applications*, *15*(1), 1-31.

Han, J., and Fu, Y. (1999). Mining multiple-level association rules in large databases. *IEEE Transactions on Knowledge and Data Engineering*, *11*(5), 798-805.

Han, J., Pei, J., Yin, Y., and Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, *8*(1), 53-87.

Kaski, S., Honkela, T., Lagus, K., and Kohonen, T. (1998). WEBSOM–self-organizing maps of document collections. *Neurocomputing*, *21*(1-3), 101-117.

Mittal, A., Nagar, A., Gupta, K., and Nahar, R. (2015). Comparative study of various Frequent Pattern Mining algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, *4*(4), 550-553.