Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

CIP- Efficient Method for Mining Frequent Itemsets from Data streams using Landmark Window Model

F. Ramesh Dhanaseelan
Department of Computer Applications, St. Xavier's Catholic College of Engineering, Chunkankadai - 03
Email: dhanaseelan@sxcce.edu.in
M. JeyaSutha
Department of Computer Applications, St. Xavier's Catholic College of Engineering, Chunkankadai - 03
Email: jayasuthaus@rediffmail.com

ABSTRACT

Continuous stream transactions like network monitoring, retail market data analysis and stock market prediction need the "frequent patterns" to be detected recurrently. Literature suggests that several pattern mining solutions are being developed over years. Still lot of challenges need to be addressed due to rapidness in generation of continuous, unbounded and ordered data real time. Hence extraction of frequent patterns from recent data will improve the analysis of stream data. In this article, a new landmark window model CIP (candidate indexing and pruning) is considered for mining the datasets. CIP allows us to mine over entire history of data streams, which improves the accuracy. This article also proposes the candidate indexed sub (CIS)-tree scheme to extract the essential information from each incoming transactions of data streams. Our proposal is compared with the existing "improved data stream mining" (ISDM) for maximal frequent itemsets algorithm. Extensive experimental analyses prove the superiority of the proposed CIP over the popular ISDM in terms of accuracy and time complexity for high-speed data stream. This article also covers up a case study where the proposed approach is applied for an application called "web prefetching".

Keywords -**Data streams, frequent itemsets, pruning, frequent patterns, web prefetching**

Date of Submission:19, Nov 2022 Date of Acceptance: 29, Dec 2022

I. INTRODUCTION

The boom in hardware technology in recent years has made the assessment of real time data continuously. Even though it sounds advantageous, it throws up several challenges ahead to the researchers, as the data may grow up to infinite and unbounded within no span of time (data stream). A lot of vital information needs to be processed and recovered rapidly in certain applications namely, manufacturing flow monitoring, sensor networks, stock exchange, and telecommunications. Data mining (DM) is a computational process of discovering this vital information in an understandable structure for further decision making.

An itemset can be defined as frequent itemset (FI), if it takes major portion of the dataset. Frequent itemsets (FIs) is a well acknowledged problem in DM as it is connected with many important tasks like associations [1-3] and clusters [4]. Basic DM task would be identifying set of items, products, symptoms and characteristics, which often co-occur in a huge database [5]. Recently, database and knowledge discovery communities have focused on new data model, where data arrive in the form of continuous data streams. Data streams [6-9] possess some computational characteristics, such as unknown or unbounded length, possibly very fast arrival rate, inability to backtrack over previously arrived data elements (only one sequential pass over the data is permitted), and a lack of system control over the order in which the data arrive. Hence, it will be impossible to analyze by capturing the important patterns and exceptions. Data stream is classified into offline streams and online streams. Offline streams are characterized by regular bulk arrivals. Online streams are characterized by real-time [10-11] updated data that come one by one in time. As the number of applications on mining data streams grows rapidly, there is an increasing need to perform association rule mining [12-13] on stream data.

There are three data stream processing models, namely landmark, sliding Windows and tilted/damped Windows. The landmark model mines all FIs over the entire history of data stream from a specific time point called landmark to present [14-15]. The sliding window model finds and maintains FIs in sliding windows. Only part of the data streams within the sliding window [16-20] is stored and processed at the time when the data flow in. The tilted/damped model mines FIs in data stream in which each transaction [21-24] has a weight and this weight decreases with age. Older transactions contribute less weight towards itemset frequencies [14], [25-29]. Each of these models has its own merits and demerits. However, recent literature suggests that relatively less contribution made in landmark data streaming.

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

Hence, we have decided to put effort in materializing a strong idea in landmark data streaming.

II. CANDIDATE INDEXING AND PRUNING

The proposed candidate indexing and pruning technique is devised to find the FIs from stream landmark window. This model is used to mine the most FIs [30-32] over the entire history of data streams. The frequent patterns are measured from the start of the stream upto the current moment. Our proposal mines the most FIs [33-35] irrespective of the nature of items (old or new).

The CIP algorithm has been proposed to improve the efficiency of mining FIs over the entire history of data streams when a user-specified 'θ' is given. For the efficient mining of FIs over stream landmark window, an efficient single-pass algorithm, called CIP is developed. It discovers the set of all FIs over data streams across the entire history. For constructing and mining the items a new structure generator is proposed and is called CIS-Tree. For maintaining the most FIs, a new structure called different traversal tree (DT*-Tree) is also proposed. Depending upon the 'θ', the infrequent items are removed from the DT*-Tree and placed in another structure called infrequent DT*-Tree, which is the same as DT*-Tree structure. For maintaining infrequent itemsets, the infrequent DT*-Tree is used.

The proposed algorithm consists of four steps.

1. Construct the CIS-Tree (Candidate Indexed Sub-Tree)
2. Mining the CIS-Tree.
3. Mined frequent itemsets are placed in another tree called DT*-Tree.
4. The infrequent itemsets are placed in infrequent DT*-Tree.

The figure 1 shows the CIP mining steps to find the frequent itemsets from data streams using the concept of landmark windowing technique.

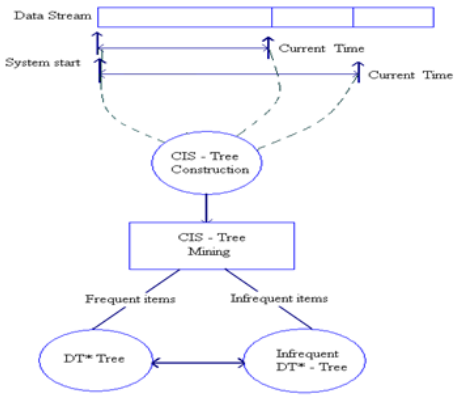

Figure 1 CIP mining steps

The CIS-Tree is used to mine the FIs. After mining, the FIs are placed in the DT*-Tree. The most FIs are placed at the top of the tree. To ensure the completeness of frequent patterns for stream data, it is necessary to store not only the information related to frequent items, but also infrequent ones. If the information about the currently infrequent items is not stored, such information would be lost. If these items become frequent later, it would be impossible to figure out their correct overall support and their connections with other items. However, it is unrealistic to hold all streaming data in the limited main memory. Thus, the patterns are divided into two categories: FIs and infrequent itemsets.

Two DT*-Trees are used in this model. One DT*-Tree maintains the FIs. The other DT*-Tree maintains the infrequent itemsets. Depending upon the value of 'θ', FIs are generated. The infrequent itemsets are removed from the DT*-Tree and placed in infrequent DT*-Tree. If the infrequent itemsets exceeds some limits, then it is deleted from the infrequent DT*-Tree.

CIS-Tree Construction

In the CIS-Tree construction process, there are five conditions. Depending upon the conditions, the tree is constructed.

Conditions

i. Read the items in the first transaction and insert into the center path. Increment the counter value by 1.

ii. Read the items in the second transaction. If the first item in the second transaction matches with the root, then follow the center path & left path and insert the transaction items. Increment the counter value for the corresponding items.

iii. If the first item in the second transaction does not match with the root, then follow the center path. If the first item in the second transaction matches with the center path, then insert the transaction items in the right path. Increment the counter value for the corresponding items.

iv. If the first item in the transaction does not match with root and center path, then follow the right path. If the first item in the transaction matches with the right path, then insert the transaction items in the right center path. Increment the counter value for the corresponding items.

v. If the first item in the transaction does not match with the root, center path and right path, then from the root, take the right path and insert the transaction items. Increment the counter value for the corresponding items.

The following transactions can be considered.

Table 1 Transaction table

Transaction ID	List of items
1	I1, I2, I5
2	I2, I4
3	I2, I3
4	I1, I2, I4
5	I1, I3

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

6	I2, I3
7	I1, I3
8	I1, I2, I3, I5
9	I1, I2, I3
10	I6, I7
11	I2, I4, I5
12	I6, I7
13	I4, I5

In the table 1, there are thirteen transactions and seven different items. The minimum support count is 2. The CIS-Tree is constructed as follows. First, the root of the tree is created. For example, the scan of the first transaction, “I1, I2, I5”, which contains three items, leads to the construction of the first branch of the tree with three nodes, <I1: 1>, <I2: 1>, <I5: 1>, where I1 is the root of the tree, I2 is linked to I1 and I5 is linked to I2. The second transaction is, “I2, I4”. The first item in the second transaction is different from the first item in the first transaction. Then search the center path and right path. If the item is found, then insert in the right direction. In the figure 2, I2 is found in the center path, therefore I4 is linked to I2 in right direction.

The third transaction is, “I2, I3”. The item I2 is found; therefore I3 is linked to I2 in right direction. The fourth transaction is, “I1, I2, I4”. The item I1 matches with the root and I2 matches with the second item, therefore increment the counter value 1 in I2, and I4 is linked to I2 in left direction. Similarly insert all the items in the tree. For transaction number ten, the items are “I6, I7” the first item I6. The item is searched with center path and right path of the tree. Here I6 is not found in the tree, therefore I6 is linked to the root in the right direction, i.e., I6 is linked to I1 in right direction and I7 is linked to I6 in center path or left path.

For transaction number thirteen, the items are “I4, I5”. The item I4 is searched with center path and right path. The item I4 is found; therefore I5 is linked to I4 in right direction. This is shown in figure 2.

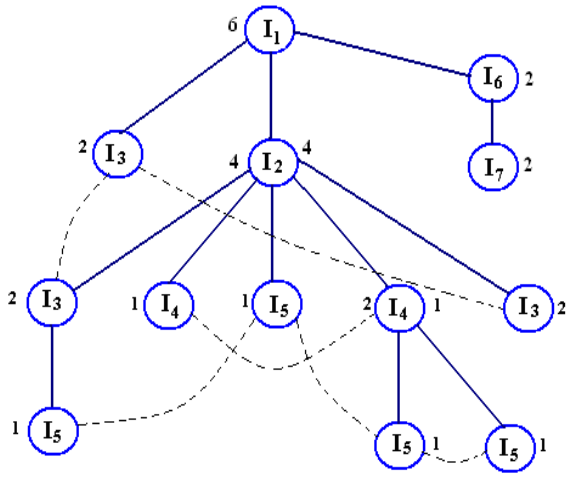

Figure 2 CIS-Tree generation

Algorithm: CIP

Input: Transaction database;
Min_sup- the minimum support count threshold.
Output: The complete set of frequent items.

The CIP mining steps is as follows.

1. Scan the transaction database and construct the CIS-Tree by calling the procedure Insert_items (T,Is).
2. The CIS-Tree is mined by calling the procedure CIS_tree_growth (T, min_sup) and the frequent itemsets are placed in DT*-Tree.
3. The infrequent itemsets are placed in infrequent DT*-Tree.

Read the items in the first transaction and insert into the center path. Increment the counter value by 1. Read the items in the next transactions and insert into left or center side of the tree or center or right side of the tree depending upon on the conditions. If the first item in the transaction matches with the root, then insert all items in the transactions in the left side of the tree and increment the counter value for the itemsets.

Algorithm :Insert_items(Tree T, Itemset Is)

```
if T is empty
for each item I of itemset Is
set item(T) = I
setitemcount(T) = 1
set T = Center (T)
else if item(T) is first item I0  of  itemset Is
Insert_left_or_center(T,Is)
else
Insert_center_or_right(T,Is)
```

If the first item in the transaction matches with the center path of the tree, then insert all the items in the right path of the tree and increment counter value for that itemset.

Insert_left_or_center(Tree T, itemset Is)

```
while item of Center(T) is next item of  Is
incrementitemcount of Center(T)
set T = Center(T)
set I = nextitem(Is)
if Center(T) is null
set item of Center(T) = nextitem(Is)
else
set item(Left(T)) = nextitem(Is)
while (nextitem(Is) is not null)
set item of Center(T) = nextitem(Is)
```

If the first item in the transaction does not match with root and center path, then follow the right path. If the first item in the transaction matches with the right path, then insert the transaction items in the right center path.

Insert_center_or_right(Tree T, itemset Is)

```
set found = Find_center_right(T, firstitem(Is))
```

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

```
if found is not null
set T = found then increment itemcount(T)
if item of Right(T) is nextitem(Is)
set T = Right(T)
if Center(T) is null
set item of Center(T) = nextitem(Is)
set T = Center(T)
else
set item of Left(T) = nextitem(Is)
set T = Left(T)
else
set T = Right(T)
set item(T) = nextitem(Is)
while (nextitem(Is) is not null)
set T = Center (T)
set item(T) = nextitem(Is)
else // if not found, go to right of the root//
set T = Right(T)
whileitemnext(Is) is not null
set item(T) = itemnext(Is)
set T = Center(T)
```

Increment the counter value for the corresponding items. If the first item in the transaction does not match with the root, center path and right path, then from the root, take the right path and insert the transaction items. Increment the counter value for the corresponding items.

The CIS-Tree mining process

In the CIS-Tree mining process, there are three conditions. Depending upon the conditions, the tree is mined.

Conditions

i. If the item is at the center path or at the left path, then consider all the branches in the left and center.

ii. If there is an item at the right hand side and root is not a starting point, then find the right starting node on which the item starts, from that, take one right path and corresponding center and left path.

iii. If the item is at the right hand side and root is the starting point, then skip the root node, proceed with right hand side and take the center and left path.

For the generation of frequent itemsets, first take item I5 from the figure 2, which is the last item in the tree. In the center and left side of the tree, I5 occurs two times. In the right side of the tree, I5 occurs two times. The paths formed by these branches are {I1, I2, I5: 1}, {I1, I2, I3, I5: 1}, {I2, I4, I5: 1} and {I4, I5: 1}. The frequent itemsets are {I1, I5} = 2, {I2, I5} = 3, {I4, I5} = 2, {I1, I2, I5} = 2, {I1, I2} = 2, {I3, I5} = 1 and {I5}=4. The itemsets that do not end with I5 are removed. Therefore remove the itemset {I1,I2}=2, because the itemset is end with item I2. The minimum support value taken is 2. Therefore, eliminate the set {I3,I5} because the count value of this set is 1.

Next take the item I4, the paths for I4 are {I1, I2, I4: 1},{I2, I4: 2} and {I4 :1}. The frequent itemsets are {I2, I4} = 3, {I1, I4} = 1 and {I4}=4. Eliminate {I1, I4} because the count value for this set is 1. Next take the item I3, the paths formed are {I1, I3: 2}, {I1, I2, I3: 2} and {I2, I3: 2}. The frequent itemsets are {I1, I3}= 4, {I2, I3}=4, {I1, I2, I3}=2 and {I3}=6. Next take the item I2. The paths are {I1, I2: 4} and {I2: 4}. The frequent itemsets are {I1, I2}=4 and {I2}=8. For the item I1 the frequent item is {I1}=6.

For I7, its path starts from the root. So skip the root and the path is {I6, I7: 2}. The frequent itemsets are {I6, I7}=2 and {I7}=2. For the item I6 the frequent item is {I6}=2. The final frequent 1-itemset are {I5}=4, {I4}=4, {I3}=6, {I2}=8, {I1}=6, {I6}=2 and {I7}=2. The final frequent 2-itemsets are {I1, I5} = 2, {I2, I5} = 3, {I4, I5} = 2, {I2, I4} = 3, {I1, I3}=4, {I2, I3}=4, {I1, I2}=4 and {I6, I7}=2. The final 3-frequent items are {I1, I2, I3}=2 and {I1, I2, I5}=2.

Algorithm : CIS_TREE_GROWTH(Tree T, Number min_sup)

```
For each item I of itemset
    For each item J of itemset from Inext to Ilast
if I is the first item I1 of itemset
count(I,J)=FindLCount(T,J,min_sup)
else
count(I,J)=FindRCount(T,J,min_sup)

FindLCount(Tree T, Item J, Number min_sup)
if (item(T) is J)
if (itemcount(T) is >= min_sup)
  count = itemcount(T)
else count =0
else if (Center(T) is not null)
set count = count + FindLCount(Center(T), J,min_sup)
else if  (Left(T) is not null)
set count = count + FindLCount(Left(T),J,min_sup)
return the value of count

FindRCount(Tree T, Item I, Item J, Number min_sup)
if item of T is I
if item of T's Right is J
set count = itemcount (T)
else if Left(T) is not null
set T=Left(T)
  set count = count + FindRCount(T,I,J,min_sup)
else if  Center(T) is not null
set T = Center(T)
```

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

set count = count + FindRCount(T,I,J,min_sup)
else if Right(T) is not null
set T = Right(T)
set count = count + FindRCount(T,I,J,min_sup)
return count

If the item is at the center path or at the left path, then consider all the branches in the left and center. Remove the items whose counter value is less than the minimum support. If the item value is greater than the minimum support, then take the counter value for those items and generate the frequent itemsets. If there is an item on the right hand side and root is not a starting point, then find the right starting node on which the item starts, from that, take one right path and corresponding center and left path. If the item is on the right hand side and root is the starting point, then skip the root node, proceed with right hand side and take the center and left path. Remove the items whose counter value is less than the minimum support. If the item value is greater than the minimum support, then take the counter value for those items and generate the frequent itemsets [36-37].

III. FIsIN DT* TREE

The pruned candidates are placed in another index structure, named DT*-Tree. This tree uses the concept of B-Tree [38] and T*-Tree [39]. The advantage of using this tree is, it will maintain the most frequent itemsets and the results will be analyzed in fast manner. The DT*-Tree consists of two types of nodes. One is outer level node and another one is the inner level node. For inserting an item, first check with the infrequent DT*-Tree whether the item is already available or not. If the item is present in the infrequent DT*-Tree, then for the item, increment the count value by 1.

If the item is not available in the infrequent DT*-Tree then insert the item in to the DT*-Tree and increment the count value by 1. The item is stored in the outer level node along with their count value. The item with the same count value is placed inside the inner level node. The growth is from top to bottom and pointer links every node. If the item is accessed for the second time, then the item is moved to the second outer level node. Now the growth is from bottom to top just like in B-Tree. Again when the same item is accessed, the item moves to the third outer level node. Similarly the item is moved and finally it takes the highest priority. So, now the root has the highest priority item.

CIP uses DT*-tree index structure to filter the unnecessary itemsets. When the CIS-Tree mined, the FIs are inserted into a DT*-tree. If the count value is greater than any other node count value, then it should be moved to the next higher level. Finally the count value, which is greater and equal to the 'θ', should be taken. The CIP prunes all the entries whose support is less than the 'θ'. After pruning, the frequent itemsets are placed in the DT*-Tree, which is used to maintain the most frequent itemsets. Figure 3 shows the frequent itemsets that are placed in the DT*-Tree.

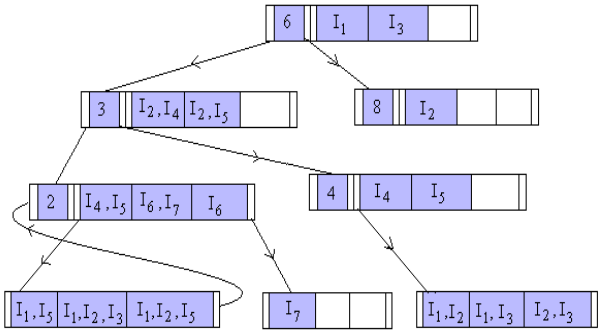

Figure 3: DT*-Tree for frequent itemsets

The infrequent itemsets are placed in infrequent DT*-Tree. This is shown in figure 4.

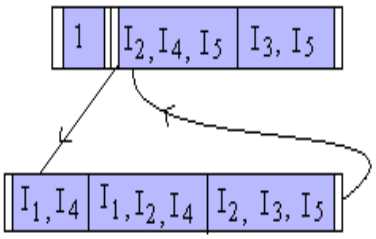

Figure 4: Infrequent DT*-Tree

To get the real useful FIs, the 'θ' should be adjusted. Let 'os' be the old support threshold, and 'F' be the set of all mined FIs. When the minimum support is changed to os1, there are two probabilities:

1. os1>os, i.e., some FIs may be not frequent.
2. os1<os, i.e., some infrequent itemsets may become frequent.

Set the new FIs as F1. DT*-Tree needs the first case. In the first case F1 will be got easily, i.e.,

$$F^1 = \{X \in F^1 / X.sup \geq os^1\}$$

IV. RESULTS AND DISCUSSIONS

All experiments were conducted in Intel® Core™2 Duo CPU, E7500 @2.93 GHz, 1.98 GB RAM and 250GB Hard Disk running on Windows XP. For the performance analysis, the datasets are taken from http://fimi.cs.helsinki.fi. The algorithms have been implemented in Java NetBeans version 6.8. The number of distinct items is 800; the maximum average number of items per transaction is 7. The parameter settings used in the experiments are shown in table 2.

Table 2: Parameter Setting

Parameter	Value
N	800
T	1,00,000

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

θ	2~10
A	5~7

Where N – Number of different items
T – Total Number of transactions
θ – Minimum Support value
A – Average transaction length

In the first experiment, the accuracy of the CIP algorithm is measured. Accuracy of an algorithm is defined as the fraction of reported frequent itemsets that are actually frequent. The accuracy of the algorithm is compared for number of transactions. The CIP algorithm is compared to the IDSM-MFI [40]. If the total number of transactions is 10,000, the CIP will generate the frequent itemsets with the accuracy of 98.7%. The IDSM-MFI generates the frequent itemsets with the accuracy of 97.8%. If the total number of transactions is 1,00,000 the CIP will generate the frequent itemsets with the accuracy of 97.4%. The IDSM-MFI generates the frequent itemsets with the accuracy of 96.8%. This is shown in the figure 5.

Figure 5 Accuracy of the algorithms for T

In the second experiment, the run time of mining from the CIP is measured. The runtime of mining from the CIP and the IDSM-MFI are compared. When the number of data sets increased, the run time of mining from the CIP slightly increased but the runtime of mining the IDSM-MFI is increased higher than the CIP algorithm. This is shown in the figure 6.

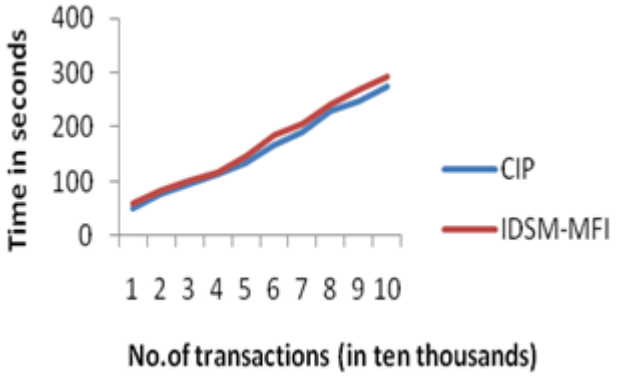

Figure 6 Execution time for T when A = 5

The third experiment is conducted for execution time for number of transactions when average length of the transaction is 7. When the average length of the transactions is increased, the total number of transactions will be decreased. If the total number of transactions is decreased, the run time will also be decreased. This is shown in figure 7.

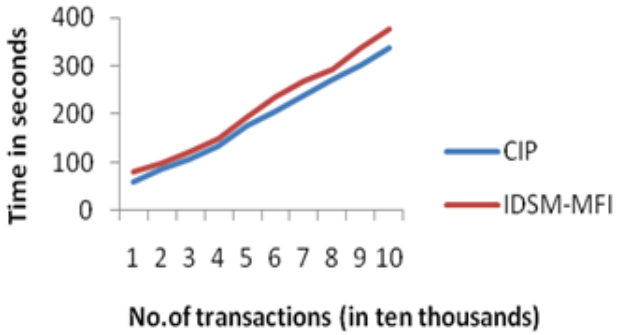

Figure 8Execution time for T when A = 7

In the fourth experiment, the effect of minimum support is tested. When the minimum support is increased, the runtime will be decreased. In the case of the CIP, it will maintain a constant level for the minimum support value of 7, 8, 9 and 10. This is shown in figure 8.

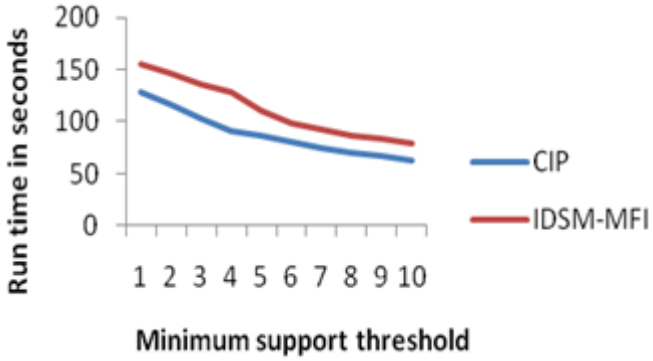

Figure 8 Execution time with**θ**

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

The experimental results show that the CIP algorithm outperforms the existing IDSM-MFI. The CIP algorithm is an adaptive approximate algorithm for finding frequent itemsets over the entire history of data stream. The CIP is used to find the most frequently used itemsets in the entire history of database.

VI CASE STUDY

For illustrating the effectiveness of the CIP tree, a web prefetching application is described. Since the bandwidth of the network is limited, web page access with no latency is impossible. In order to solve this problem, researchers have come with solution called web prefetching. The concept is based on the history of access pattern.

The most expected web page to be accessed next by the user is estimated by the previous history of access patterns. In order to achieve this, the access patterns are identified from the web access log. These access patterns are grouped and frequency of every unique access pattern is computed. The most frequent access pattern is the most expected access pattern. The access patterns are ordered according to their frequencies. Based on the current access sequence of the user, the most expected access pattern is identified from the list by matching the prefix.

From the matched entry the next expected web page to be accessed is fetched from the web server and stored in the local machine. Now, when the user accesses that web page, the page will be displayed immediately by accessing the local file. This process eliminates the latency. CIP tree is the suitable data structure used to find the most FI. We have taken real data from world cup football 1998 web access log(http://ita.ee.lbl.gov). The summary of the access log is given in table 3.

Table 3 World cup 1998 Web Access Log

World cup 1998 Web Access Log	
From : 01/Jun/1998:22:00:01 To : 02/Jun/1998:02:57:15	
Description Count	Count
Total URLS	936071
Access Pattern(Itemset)	12929
Unique Itemset	10970
Itemset with Frequency Greater than 1	357

The CIP tree is constructed by taking all 936071 URLS present in the access log for the period of 4hrs 57mins 14secs. Total of 12929 users accessed these URLS in different sequences called access pattern, among these, 10970 access patterns are unique. 357 access patterns are having frequency greater than 1. The unique URLS present in the frequent itemsets are given in table 4, and are mapped with the symbols Ui as mentioned in the table 4.

Table 4: URLS present in the frequent ite>

URL ID	URL
U1	/images/home_tool.gif
U2	/images/home_sponsor.gif
U3	/images/home_intro.anim.gif
U4	/images/home_fr_phrase.gif
U5	/images/home_fr_button.gif
U6	/images/home_eng_phrase.gif
U7	/images/home_eng_button.gif
U8	/images/home_bg_stars.gif
U9	/images/home_logo.gif
U10	/english/playing/download/images/big.bird.gif

The top 5 frequent itemsets are shown in table 5. 357 users had accessed the web site having the same access pattern, this forms the most frequent itemset with eight items {U1,U2,U3,U4,U5,U6,U7,U8}.

Table 5: Top 5 frequent itemsets

URL	**Access Pattern**	**Frequency**	**Number of URLs**
U1 U2 U3 U4 U5 U6 U7 U8	/images/home_tool.gif, /images/home_sponsor.gif, /images/home_intro.anim.gif, /images/home_fr_phrase.gif, /images/home_fr_button.gif, /images/home_eng_phrase.gif, /images/home_eng_button.gif, /images/home_bg_stars.gif,	357	8
U1 U2 U9 U3 U4 U5 U6 U7 U8	/images/home_tool.gif, /images/home_sponsor.gif, /images/home_logo.gif, /images/home_intro.anim.gif, /images/home_fr_phrase.gif, /images/home_fr_button.gif, /images/home_eng_phrase.gif, /images/home_eng_button.gif, /images/home_bg_stars.gif,	278	9
U3	/images/home_intro.anim.gif,	85	1
U10	/english/playing/download/images/big.bird.gif,	73	1
U8	/images/home_bg_stars.gif,	44	1

278 users had accessed the web site having the same access pattern, this forms the second most frequent

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

itemset with nine items { U1,U2,U9,U3,U4,U5,U6,U7,U8}. 85 users had accessed the web site having the same access pattern, this forms the third most frequent itemset with only one item{ U3}.

V. CONCLUSION

Mining data streams is a very demanding and challenging area of research. Real time data streams are very rapid, making the traditional methods fall short. Existing schemes do not use the past history, which might be essential at any point of time. This situation can be handled by the landmark data mining schemes. But, literature suggests that, very feeble amount of research has been devoted. In this article, we have proposed a new CIP based landmark scheme. It uses a user-specified 'θ' for the efficient single-pass algorithm to mine the FIs over data stream. It does not discard the history of the data streams, which is its main strength compared to other existing schemes. For constructing and mining the items, a new structure generator called 'CIS-Tree', is also proposed. This model is suitable for certain applications, where people need the most FIs. The efficacy of the proposal is compared and contrasted with the existing IDSM-MFI. Examination is performed over 1,00,000 transactions. The proposal could be able to achieve 97.4% accuracy, compared to 96.8% of the IDSM. The proposed test is also tested by varying values of 'θ' for its run time effectiveness. Moreover, a real time application, called prefetching is implemented, using the data collected from world cup 1998. The proposed approach has successfully outperformed its counterpart in every aspect. This opens up a new area of research in landmark data mining.

REFERENCES

[1] Agrawal R, Srikant R (1994) Fast Algorithms for Mining Association Rules. In Proc. of VLDB, pp 487- 499

[2] Agrawal R, Srikant R (1995) Mining Sequential Patterns. In Proc. of IDCE, pp 3-14

[3] Liu B, Hsu W, Ma.Y (1998) Integrating Classification and Association Rule Mining. In Proc. of KDD

[4] Wang H, Yang J, Wang W, Yu PS (2002) Clustering by Pattern Similarity in Large Datasets. In Proc. of SIGMOD, pp 394-405

[5] Vimal Kumar D, Tamilarasi A (2013) An effective approach to mine relational patterns and its extensive analysis on multi-relational databases Int. J. of Data Mining, Modelling and Management, Vol.5, No.3, pp.277 - 297

[6] Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. Proceedings of PODS, pp 1-16

[7] Graham Cormode, Muthukrishnan S (2005) What's Hot and What's Not: Tracking Most Frequent Items Dynamically. ACM Transactions on Database Systems 30:249-278

[8] Golab L, Ozsu MT (2003) Issues in data stream management. SIGMOD 32: 5-14

[9] Jun Tan, Yingyong BU and Haiming Zhao (2010) Efficient Single-pass Frequent Itemsets Mining over Data Streams. Seventh IEEE International Conference on Fuzzy Systems and Knowledge Discovery, pp 1438-1431

[10] Chang, Lee, Zhou (2003) Finding Recent Frequent Itemsets Adaptively over online Data Streams. ACM SIGKDD International Conference on knowledge Discovery and Data Mining, pp 487-492

[11] Lukasz Golab, Theodore Johnson, and VladislavShkapenyuk (2012) Scalable Scheduling of Update in Streaming Data Warehouses. IEEE Transactions on Knowledge and Data Engineering 24:1095-1105

[12] Nan Jiang, Le Gruenwald (2006) Research issues in Data Stream Association Rule Mining. SIGMOID Record, 35:1

[13] Sotiris Kotsiantis, DimitrisKanellopoulos (2006) Association Rules Mining: A Recent Overview. GESTS International Transactions on Computer Science and Engineering, 32: 91-82

[14] Li H, Lee S, Shan M(2004) An Efficient Algorithm for Mining Frequent Itemsets over Entire History of Data Streams. In Proc. of First International Workshop on Knowledge Discovery in Data Streams

[15] Wang J, Han J, Pei J (2003) CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In Proc. of KDD, pp 236-245

[16] Chang, Lee (2005) A sliding window method for finding recently frequent itemsets over online data streams. Journal of Information Science and Engineering pp 76-90

[17] Chi Y, Wang H, Yu PS, Muntz RR (2004) Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. In Proc. of ICDM, pp 59-66

[18] Chih-hsiang Lin, Ding-ying Chiu, Yi-hung Wu (2005) Mining frequent itemsets from data streams with a time sensitive sliding window. SIAM International Conference on Data Mining, pp 486- 491

[19] Dawar S, Sharma V, Goyal V, (2017) Mining top-k high-utility itemsets from a data stream under sliding window model, Applied Intelligence, 47(4), pp 1240–1255

[20] Chang Y-I, Li C-E, Chou T-J, Yen C-Y (2018) A weight-order-based lattice algorithm for mining maximal weighted frequent patterns over a data stream sliding window, 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, Japan, 13-17 April 2018

Volume: 14 Issue: 04 Pages: 5563-5571 (2023) ISSN: 0975-0290

[21] Kuen-Fang Jea, Chao-Wei Li, Tsui-ping Chang (2008) An efficient approximate approach to mining frequent itemsets over high speed transactional data streams. IEEE Eight International Conference on Intelligent Systems Design and Applications, pp 275-280
[22] Bo Li (2009) Finding Frequent Itemsets from Uncertain Transaction Streams. IEEE International Conference on Artificial Intelligence and Computational Intelligence, pp 331-335
[23] Li, A., Xu, W., Liu, Z. et al(2021). Improved incremental local outlier detection for data streams based on the landmark window model. KnowlInfSyst 63, 2129–2155.
[24] Kolomvatsos K and Anagnostopoulos C (2021), "Landmark based Outliers Detection in Pervasive Applications," 2021 12th International Conference on Information and Communication Systems (ICICS), 2021, pp. 201-206.
[25] Lee D, Lee W(2005) Finding Maximal Frequent Itemsets over Online Data Streams Adaptively. In Proc. of ICDM, pp 1550-1505
[26] Chang JH, Lee WS (2003) estWin: Adaptively Monitoring the Recent Change of Frequent Itemsets over Online Data Streams. In Proc. of CIKM, pp 536-539
[27] Chernoff H (1952) A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. The Annals of Mathematical Statistics 23:493-507
[28] Yu J, Chong Z, Lu H, Zhou A (2004) False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In Proc. of VLDB, pp 204-215
[29] Chang JH, Lee WS (2003) Finding Recent Frequent Itemsets Adaptively over online Data Streams. In Proc. of KDD, pp 753-762
[30] Giannella C, Han J, Pei J, Yan X, Yu PS (2003) Mining Frequent Patterns in Data Streams at Multiple Time Granularities. H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.) Next Generation Data Mining
[31] Chen Y, Dong G, Han J, Wah B.W, Wang J (2002) Multidimensional Regression Analysis of Time- Series Data Streams. In Proc. of VLDB, pp 323-334
[32] Gouda K, Zaki M (2001) Efficiently Mining Maximal Frequent Itemsets. In Proc. of ICDM
[33] ToonCalders, NeleDexters, Bart Goethals (2008) Mining Frequent Itemsets in a Stream. Seventh IEEE International Conference on Data Mining, pp 83-92
[34] RenJiadong, He Huiling, XuLina, Hu Changzhen (2009) DSMFI-Miner : An Algorithm for Mining Maximal Frequent Itemsets on Data Streams. IEEE Second International Workshop on Computer Science and Engineering, pp 139-143
[35] Alfredo Cuzzocrea, Fan Jiang, Wookey Lee, Carson K.Leung (2014) Efficient frequent Itemset Mining from Dense Data Streams. APWeb, Springer, (LNCS 8709), pp 593-601
[36] Luigi Troiano, G. Scibelli (2013) A time-efficient breadth-first level-wise lattice-traversal algorithm to discover rare itemsets. Data Min. Knowl. Disc., Springer 27:1-35
[37] Luigi Troiano, GiacomoScibelli (2014) Mining frequent itemsets in data streams within a time horizon. Data & Knowledge Engineering, Elsevier, 89:21-37
[38] Hongjun Lu, YuetYeung Ng, ZenpingTian (2000) T-Tree or B-tree: main memory database index structure revisited. 11th IEEE Australasian database conference, pp 65-73
[39] Kong Rim Choi, Kyung-Chang Kim (1996) T*-tree: a main memory database index structure for real time applications. IEEE workshop on real time computing systems and applications, 81-88
[40] Yinmin Mao, Hong Li, Lumin Yang, Zhigang Chen, Lixin Liu (2009) A Mining Maximal FrequentItemsets over the Entire History of Data Streams. Proceeding of the First IEEE International Workshop on Database Technology and Applications, pp 413-419

Biographies and Photographs

F.RameshDhanaseelan received Master degree in Computer Science from Bharathidasan University in 1992, M.Tech in CSE from Pondicherry University in 1998 and Ph.D in CSE from Alagappa University in 2009. He is in teaching profession for the past 27 Years. Currently, he is working as Professor in St. Xavier's Catholic College of Engineering, Tamil Nadu, India. His areas of interest are Big Data and Machine Learning.

M Jeyasutha received Master degree in Computer Applications from Mother Teresa Women‟s University, Kodaikanal, India in 2003. She obtained M.Phil. degree in Computer Science from Madurai Kamaraj University, India in 2005 and PhD degree in Computer Science from Bharathiyar University, Coimbatore, India in 2018. She is in teaching profession since 2004. Currently, she is working as an Assistant Professor in St. Xavier's Catholic College of Engineering, Tamil Nadu, India. Her current research interest includes Stream mining, Bigdata mining and Machine learning. She is a member of IET.