International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August, 2019

Mining Closed Item sets from Tuple-Evolving Data Streams

Bhargavi Peddireddy, Ch. Anuradha, P.S.R.Chandra Murthy

Abstract: ***Frequent Itemset Mining is playing major role in extracting useful knowledge from data streams that are exhibiting high data flow. Studies in data streams shows that every incoming data is considered as new tuple which is considered as revised tuple in some applications called as tuple evolving data streams. Extracting redundant less knowledge from such kind of application helps in better decision making with new challenges.One of the issue is, due to incoming revised tuple, some of the frequent itemsets may turn to infrequent or previously ignore itemsets may become frequent. Other issue is result of FIM may be huge and redundant results.In this paper, we address solution to the problem by finding closed itemsets from tuple revision data streams. We propose an efficient approach MCST that uses compressed SlideTree data structure to maintain stream data,proposeHIS hash tableto maintain itemsets, and CIS tables to maintain closed id sets to improve search performance of HIS.***

Keywords:Closed itemsets, data streams, SlideTree, tuple-evolving data streams.

I. INTRODUCTION

Data Mining is a process of identifying hidden knowledge as patterns which are useful for making decisions from the given data. One of the hidden knowledge is Frequent Patterns, mining such knowledge is called with a name Frequent Itemset Mining (FIM) [17]. Frequent itemsets are the combination of items whose occurrence in the database is important and reaches the given threshold. It has been extensively used in many applications where the data analysis is required. As a result, the outcome is huge list of frequent itemsets which can cause delay in decision making due to redundancy. Hence, lossless and condensed representation of frequent itemsets plays an important role in making decisions. From the literature, it is found that Maximal Itemsets and Closed Itemsets are the major condensed FI's. However Closed Itemset plays a major role since it doesn't exhibit redundancy in output.Closed Itemsets [24] (CI's) are the itemsets whose occurrence is not same as any one of the other itemset.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
Bhargavi Peddireddy*, Department of Computer Science and Engineering, ANUCET, Acharya Nagarjuna University, Guntur, AP, India
Ch. Anuradha, Assistant Professor, Department of Computer Science and Engineering, V.R.Siddhartha Engineering College, Vijayawada, AP,India
P.S.R.Chandra Murthy, Department of Computer Science and Engineering, ANUCET, Acharya Nagarjuna University, Guntur, AP, India

CI's able to derive other itemsets which are subset itemset occurrences. It is very essential in environment where the incoming flow rate is higher.

Due to the rapid growth of technology, high flow huge data will be arrived continuously as input to the database is known as Data Streams.In such situation, transactions are to be processed in a fast manner for answering a given query [5]. Since each data is considered as new transaction, Extending FIM to the data streams has been attracted[5, 15, 21, 23].Several models have been proposed for faster processing. Some consider unbounded streamingand others consider sliding window [26] such that they discover frequent itemsets in the corresponding sliding window. Sliding window methods have been used many applications, since it is proved that it is complete and efficient in deriving FI's.

In real time, some applications may consider some of the incoming transactions are the updated version of the previous transactions. In such environment, previously visited transaction and corresponding knowledge is to be updated. In sliding window based data streams, they do not maintain previous transaction. Hence, it is difficult to retrieve the previous transaction as well as updating knowledge. The following motivation example gives the importance of tuples that gets updated.

A. Motivation example:

Auction site application allows customers to bid the list of items in which they are interested from the auction list. It also gives provision that user can update their bid on any item at any time. Because of this provision, customers will be able to submit several bids on the same list. Hence these kind of bids considered as the updated version of the previous bids. Over the time, items in the auction expires due to its circumstances.Table 1 is considered as initial instance of auction site.It is recorded with 12 tuples, where each tuple represents a bid given by user id {UT1, UT2, … ,UTM}, and sliding window size |M| is 12. For easy processing, sliding widow is considered with windows whose size is 4. Therefore, it can be seen that has three windows from UT1-UT4, UT5-9, and UT10-UT14. Hence the previous tuple need to be removed as well as knowledge when tuple gets updated.Such kind of data streams are known as Tuple-Evolving Data Streams (TEDS) [13].

UT1	a	c			
UT2	b	d	e		
UT3	a	c			
UT4	b	e			
UT5	b	e			
UT6	b	e			
UT7	a	b	c	d	e
UT8	a	c	f		
UT9	b	c	d	e	
UT10	a	c	d		
UT11	a	e			
UT12	a	b	c	e	

Table 1: Sample Auction Database [7]

The various instances of auction bids are projected in Figure 1. From the Figure 1(a), it can be seen that second instance does not have the transactions of 1,2,3 and 4 which are expired when window moves from W(UT1-12) to W(5-14). And other instances also visualized in the same.Traditional data stream FIM-techniques of closed itemsets are mentioned in Figure 1(a) for the consideration minimum support 40%. Second instance contains duplicate transactions UT8 and UT 10 tends to <ac>:5 is closed which is supposed to be infrequent. The reason for such invalid result is traditional data steams consider every incoming transaction as a new tuple.

Revision model consider transactions as new if they are not appeared, and consider as old and gets updated if it is already visited. Revision model for the same example is presented in Figure 1(b). It can been seen that UT8 and UT10 gets updated in second sliding window, then its minimum support count is to 4 (40% of 10), and itemset <ac> become infrequent with the support count 3.Duplication causes an itemset<*ce:4*> was reported as infrequent in the thirdslide, because minimum support count is 5 which is supposed to 4. From this example, it is observed that ignorance of tuple evolving can causes for such invalid knowledge. In addition to that, it is also observed that Frequent itemsets of slide 3are {a:6, b:7, c:5, e:6, ab:4, bc:4, be:5, ce:4, bce:4}, among that itemsets <bc:4>and <ce:4> are redundant. Hence it can be replaced with Closed Itemset<bce:4>.

The possible solutions are visualized for the tuple evolving data streams bellow.

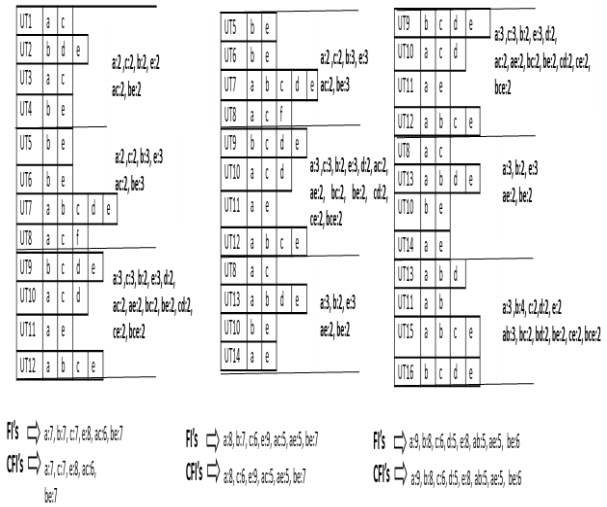


Figure 1.a: Naïve solution

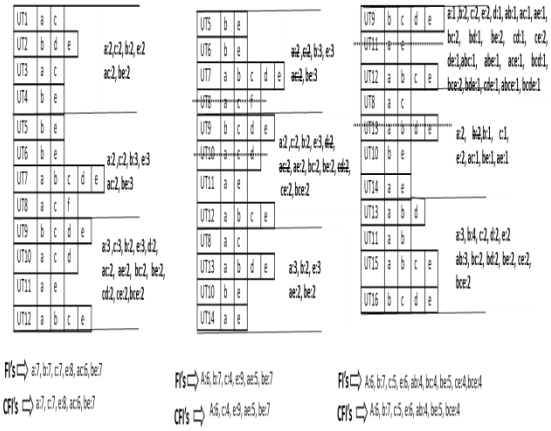


Figure 1.b: Revision Model-Remove and Update

Few studies from literature stating that efficient approaches are available on frequent itemset discovery from TEDS. However, those are also not exception redundancy in output. In this paper, we do investigation on closed itemset mining from data streams TEDS.

B. Contributions:

Here, we discuss the components towards the working principle of deriving closed itemsets from the tuple evolving datastreams. We use SlideTree [7] to maintain up-to-date transactions in compressed manner. We propose Update algorithm to handle when the tuples get updated due to incoming transactions. We also propose **MCST-M**ining **C**losed item**S**ets from **T**uple evolving streams. The proposed method uses hash tree for fast accessing and computation. In addition to that, one more ClosedItemset Set hash tree is proposed to reduce the search time during closure checking of closed itemsets.

The remaining paper is presented as follows, wefirst discuss the related work of data streams in very next section that is section 2. We present the formal notations and conditions that supports chosen model for dealing with tuple-evolving in section 3. We introduce the data structures in and the proposed approach for handling tuple-evaluation in section 4. We perform the experiment evaluation in section 5. And finally conclude this paper with a conclusion in section 6.

II. RELATED WORK

In data streams, usually associated with characteristics size is unbounded and incoming tuple flow rate is high. Various models have been proposed to meet the data stream. Data stream FIM's uses theLand mark and sliding model are basic models for Itemset mining. One of the technique is *estWin* [16], maintains candidates in in prefix lattice based structure.Each node denotes an itemset and edge represents relation between them. Advantage of this method is that they visit lattice tree only when mining result is requested.Mozahariet. al. [6] has proposed delayed sliding window approach. In this approach, stream is divided into smaller window when the stream size is large. The major drawback of this approach is delay is one window whose speed is not same as incoming flow rate.

Every incoming tuple may not be new tuples, some may be revision which are not same as normal updates, and they are the corrections that invalidate old tuple or knowledge and incorporate new knowledge [14, 19].To improve the performance, Alexandru et al. [1] proposed the framework for managing the data that is storage-centric. ParisaHaghani et al. [23] proposed top-k query processing, and improved model ABS [10]. Clustered based Itemset evolving is proposed to derive itemsets from data streams.

Fideo framework is proposed by C Zhang et al. [13] to handle tuple revision model. The model uses *swTree*to maintain updated tuples information and cfTree to maintain candidate itemset information of a new slide. In addition, MVerifieris used to ensure that the output is complete. However, for each new transaction, huge storage is require for keeping old and new slide information, and more computation power is required to handle two tree data structures and MVerifier method.In [7], we present FIDE model to improve the performance of Fideo model in deriving frequent itemsets from tuple revision model. FIDE uses SlideTree and FIHashTable for keeping slide information and faster processing frequent itemsets. However, the output may contain large itemsets with redundancy. In [8], we present MFIDE model to avoid redundancy by deriving Maximal Frequent Itemsets from such data streams. However, such kind of knowledge is losing actual frequency.It has not been investigated in deriving frequent itemsets without redundancy and lossless.

Chi et al [12] has proposed Closed Enumerated Tree approach to mine CIM from a sliding window. In FIM approaches, FP-tree based Moment algorithm [12], CLOSET [18], and are the itemset mining techniques to find compact representation of itemsets. CLOSET is an augmentation of the FP-improvement. New moment algorithm [12], bit vector based approach for mining CIM from sliding window data streams. New CET a prefix tree based data structure is used to maintain itemset information. But, it consumes huge memory because of the non-closed itemset generation. Zhang et al. [27] proposed bit vector and diagraph based approach MFCIDS-BD to derive CIM. Diagraph is used to find closed itemsets in visiting tree in depth first manner. Yen et al. [26] proposed a method to derive CIM without looking at the previous closure information. It is under the performance of too many intersection operations. Chang et al. [11] proposed Subset-lattice, combinations of bit vector Tid's to derive CIM. But it is limited to single window. Lattice is to be reconstructed for every new window. Nori et al.[22]proposed TCET data structure based TMoment approach for all closed itemsets.Aliberti et al. [2] proposed EXPENDITE enumeration miner to improve the performance of Closed Itemset Mining. Ariyam Das et al. [4] Crucial Pattern Mining approach for deriving crucial patterns that are optimal closed itemsets. All the above techniques consider incoming transaction as new. These techniques do not maintain the expired slide information which helps us to update when old tuple gets updated.To the best of my knowledge, closed itemset mining has not been investigated. Hence, the remaining sections discuss the problem of mining closed itemsets from data streams when tuple gets updated.

III. PROBLEM DEFINITION

Here, we define problem statement of Closed Itemset mining from the data streams and notations of Table 2. , are associated with sliding windows, under the consideration of*revision model*(figure 1), where the tuples in the past slides get updated and the windows are cutinto smaller slides.

A. Problem statement:

Let us consider SW is a window (Sl_1, .., Sl_n) is a set of slides, Sl_i is a slide contains set of transactions (UT_1, …, UT_m), UT_i is a transaction contains a set of items, and I is an item ϵ *Items*.δ is the given minimum threshold. Support (I/W) is the occurrence count of an itemset I in window W. An itemset I is said to be frequent FI if its $Support(I) \geq \delta$. An itemset is said to be closed itemset CFI iff there is no itemset X such that X$\supseteq$Iand Support(I)=Support(X).W_{cur}= {Sl_1, Sl_2, ..,Sl_i }is the current window, when a transaction arrives or window arrives, the old window expires according to the sliding window size as W_{old}={Sl_2, Sl_3, ..., Sl_{i+1}}. When a transaction UT_jwhich is an update version of old one arrives, then corresponding Slide transaction is to be updated. The goal of Closed Itemset Mining is to derive all the closed itemsets from the Tuple Evolving Data streams.

Symbol	Meaning
W, W_{old}, W_{cur}	Window, old Window, CurrentWindow
$\|W\|$	Length of window (total number of transactions)
Sl_1, .., Sl_n	Slide
Sl'_1, … , Sl'_n	Updated Slides
Sl_{n+1}	New transactions in new slide
I	Itemset
$Support(I/W)$	Occurrence of an itemset I in window W
δ	User Minimum threshold
FI	Frequent Itemset
CFI	Closed Frequent Itemset

Table 2: Terminology

IV. DERIVING CLOSED ITEMSETS FROM TUPLE EVOLVED DATA STREAMS

In this section, we discuss the methodology of closed itemset mining from data streams. Basically it is comprised of the following steps

Step Step 1: Build compressed tree such that entire steam fit into a tree in a format that allow us to retrieve specific transaction from the tree. Build SlideTree

Step Step 2: Closed itemset mining algorithm to derive closed itemsets from streams for each incoming transaction.

A. SlideTree [7]: Sliding window Tree

In this section, we discuss the tree which is adopted from [7] that is *SlideTree*. It is used to maintain all the transactions of current sliding window.

For each new transaction, path is created with a sequence nodes in a tree. The tuple id, slide id of each transaction are maintained in a table, and a reference pointer to the last node of path is maintained. The node structure is visualized in Figure 2. Each node consists of four fields, info-label of node, sup- occurrence count, PID-parent ID, adj-adjacent node address. This kind of structure easily update tree when sliding window moves from one to other. The corresponding transactions in the expired transactions can be removed.

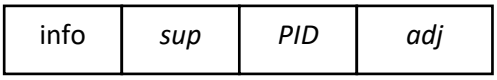


Figure 2 : Node structure

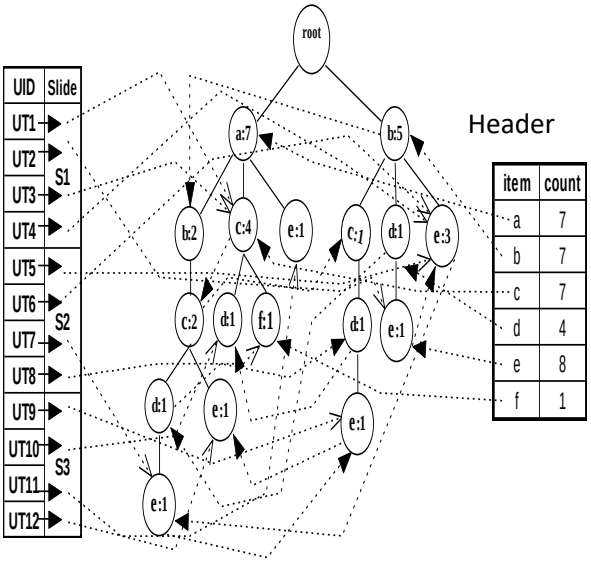


Figure 3: SlideTree of W(Sl1, ..., Sl12)

SlideTreerepresentation of the window $W(Sl_1, .. ,Sl_{12})$ is visualized in Figure3.It is partitioned into three parts:User Transaction id (UTi) with slide information, SlideTree and Header.This kind of pointers are used to retrieve easily when it is required.The Header Table is used to record lable with its support count. SlideTree is used to record each transaction as a path into the tree. And each node is associated with the structure that contains PID, sup and adj pointers that are used to find the support count of each itemset easily.

SlideTree representation of the window $W(Sl_5, .. ,Sl_{14})$ that is $W_{cur}$is visualized in Figure4. Hence, sliding window moves from W_{old} to W_{cur}, where $W_{old}$is $Sl_1 – Sl_4$, and obsolete slides are from Sl_1-Sl_4. The obsoleted transactions are removed and visualized with dotted lines.

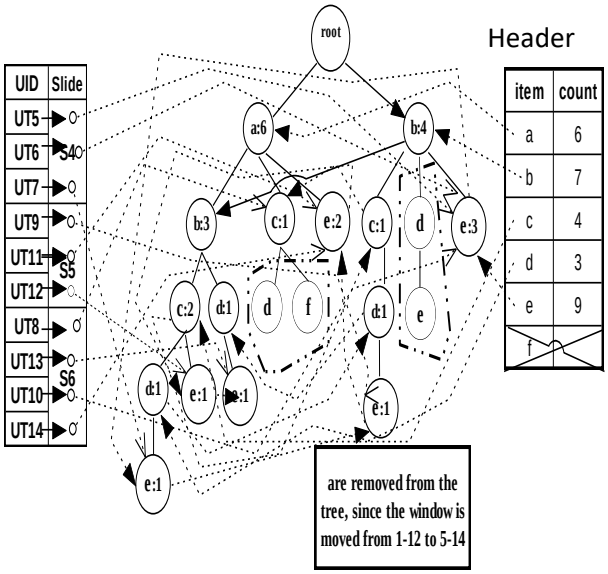


Figure 4: SlideTree of Wcur(Sl5, ..., Sl14) [7]

B. Mining Closed itemSets from Tuple evolving streams- MCST:

In this section, we present the methodology and working principle of the proposed approach for solving closed itemset mining with an example. Basically MCST adopts the set closure property in deriving closed itemsets. In pursuing closure property, it uses IHT hash table with the itemsets as keys and its occurrence as value. Basically IHT records the occurrence count of all itemsets, and these entries are used to verify the holding item is closed or not. To avoid computation for the all entries, the proposed CIS maintains closed itemset set for each product/item as shown in fig. CIS. This CIS gives an idea of which entries should be used for closure checking. Thus way it reduces computation.

Update algorithm is used to construct SlideTree for each new transaction, update tree when tuples get revised. UpdateHT method is invoked to remove the information of itemsets that are presented in obsolete slide when sliding window moves to new.

Itemset Hash Table (IHT)	
Key	Count

Fig 5(a) IHT Table

Closed Itemset Set (CIS)	
Key	Set of Closed Keys

Fig 5(b) CIS Table

It initiates Update SLTree method to insert new path into tree when a new transaction arrives. Possible combinations are generated from transaction and MCST is initiated to find closed itemsets from the transaction visited so far.

MCST takes the set of itemsets as input, identify the closed itemsets from the associated CIS table where the given input is to be verified in IHT table. Firstly, for a given input itemset, set is derived from the union of CIS of each component of itemset. If the holding itemset is closed itemset then its subsets are removed from the associated CIS table. The holding itemset is added to its associated CIS table.

After closure verification, closed itemsets are derived from CIS table whose support same or higher than δ.

Algorithm 1: Update
Input: SLTree, Lattice-Tree, W_{old}, W_{cur}, δ- minimum threshold from (0 to 1)
Output:SLTree, IHT, CIS, CI-Closed Itemsets

// For handling obsolete window
$Sl_i \leftarrow W_{old}$-W_{cur}
for each T_i from Sl_i
brach*b* from SLTreeCand←PossComb(*b*)
UpdateHT(Cand, IHT, CIS)
remove branch *b* from SLTree
end for
for each $T_i \in Sl_{n+1}$// *For handling* Sl_{n+1}*newtransactions*
Update SLTree// *insert each transaction into SLTree.*
Cand←PossComb(T_i)
Call Algorithm MCST with inputs: *Cand* and*δ// call closed itemset mining algorithm*
End for

Algorithm 2:**MCST**-Mining Closed itemsets from Tuple evolving streams
Input: IS-set of itemsets,δ- minimum threshold from (0 to 1) are formal parameters. IHT –Itemset Hash Table, CIS-Closed Itemset Set
Output:CI, IHT, CIS

IS = $IS_1, IS_2, \ldots, IS_i$
for each *ItS* from IS_i
CIS(*ItS*)=CIS(ItS_1)∪ CIS(ItS_2) ∪, …., CIS(ItS_i)
for each *is* ϵ CIS (IS_i)
if (ItS⊃ is || ItS ⊂ *is*)
if (HIS(ItS).Count==HIS(*is*).Count)
then replace CIS(ItS_i).*is* with ItS
else CIS(ItS_i).add(ItS)
end if
end for
For finding closed itemsets
Keys ← CIS
SCK {} / set name SCK, initially empty
for each *k* ϵ Keys
SCK.add(*k*)
end for
for each *sk* ϵ SCK
if HIS(*sk*).Count < δ
then remove it from SCK
end for

UpdateHT
Input:Cand, IHT, CIS

for each *CS* ϵ Cands
CS =$CS_1, CS_2, \ldots, CS_i$
CIS(CS)=CIS(CS_1)∪ CIS(CS_2) ∪, …., CIS(CS_i)
for each *ci* ϵ CIS (CS_i)
if (*ci*⊂*CS*)
if (HIS(ci).Count==HIS(*CS*).Count)
then replace CIS(CS_i).*ci* with *CS*
elseif(*ci*⊃CS)
if (HIS(ci).Count==HIS(*CS*).Count)
then remove *ci* from CIS(CS_i)
end if
end if
end for

C. Running example for MCST:

Here, we discuss the working principle of MCST with an example. Consider Table 1 as an example, before UT1, SLTree, IHT, and CIS is set to empty. The first transaction UT1 {a,c} and its possible itemsets are Cand {a, c, ac}. Consider itemset {a}, since IHT is empty, IHT puts entry to (a,1). CIS(a) is ϕ, add a to the CIS(a). For input itemset {c} then CIS(b) is {b}. For example, input itemset is <ac>, IHT(ac) is set to 1, computation of set (ac)=CIS(a) ∪CIS(b)={a,b}, these itemsets are the possible itemsets that are to be used in closure verification. Therefore, IHT(ac).count = IHT(a).count and ac ⊃a, Hence<ab> is closed. Therefore, <ab> is added to CIS(a). The same procedure is applied for all the itemsets and transactions. The result of slide1 is visualized in Figure 6.a.

The input contains a change in window means that window moves from slide 1-3 to slide 2-4, Update algorithm initiates the procedure for the removal of obsolete transactions and UpdateHT method. The result of this operation is visualized in Figure 4 and Figure7. It puts new transactions into SLTree, then invoke MCST algorithm with the argument Cand is possible itemsets that are generated from new transaction. The same procedure is applied for all the transactions and the result is visualized in Figure 6.b.

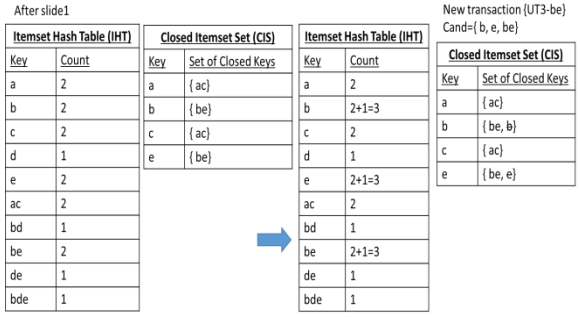


Figure 6. a: MCST –against Table 1

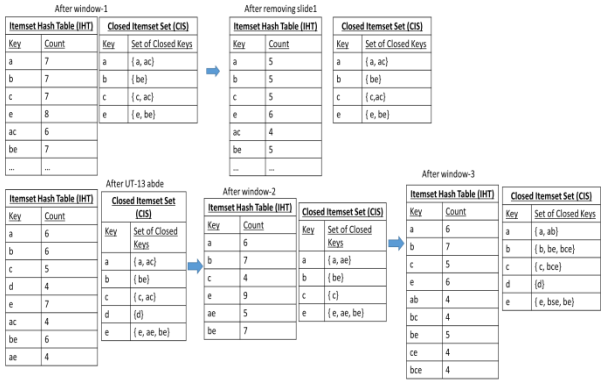


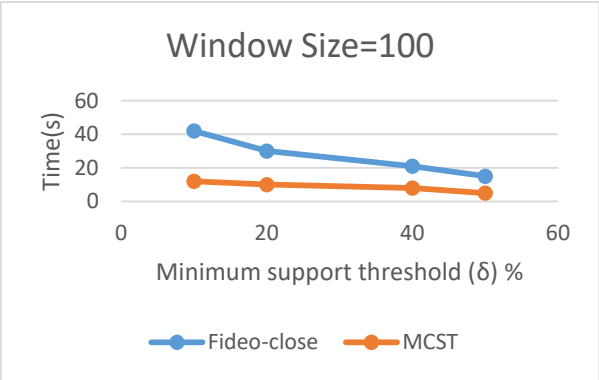
Figure 6. b: MCST(UpdateHT method) –against Table 1

V. EXPERIMENT ANALYSIS

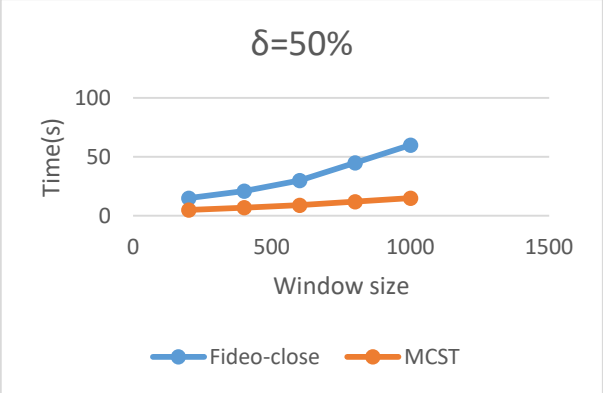
This section is to provide experimental evidence to show the performance of the proposed approach in deriving CIS's. We consider the standard datasets that are show in table for experimental result. The datasets chess and Connect doesn't have old tuples. For experimental analysis, some of the tuples are consider as updated tuples. For comparison, we consider naive approach in Fideo environment that is **Fideo-closed** which sgenerates and verifies closure property with each frequent itemset, and the proposed framework **MCST**.

Data set	no.ofitems	No.ofTransactions	Density
Chess [2]	50	3196	0.69
Connect [2]	46	67557	0.78

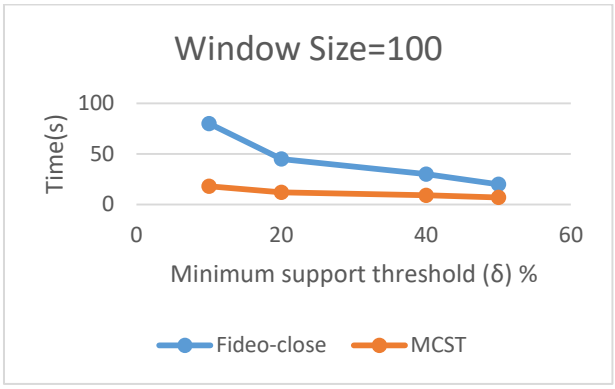
Table 3: Standard Datasets



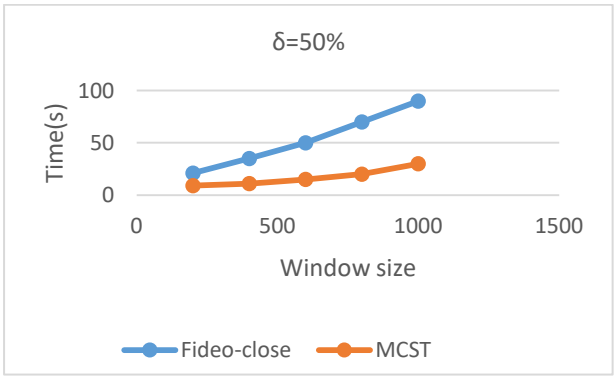
(a) Chess dataset, δ vs Time, window size=100



(b) Chess dataset, window size vs Time, δ=50%



(c) connect δ vs Time, window size=100



(d) Chess dataset, window size vs Time, δ=50%

Figure7: Execution time w.r.t threshold and window size

A. Efficiency:

Here, we discuss the performance of MCST against chess and connect dataset. Figure 7 depicts the running time of MCST and Fideo-close approach against chess and connect dataset. Figure 7 (a) shows the performance of MCST and Fideo-close against minimum threshold and running time on chess dataset for the window size is 100. It shows that the running time goes down when minimum threshold increases. The reason for such change is more number of itemsets are generated when δ is less. MCST shows better performance than naïve, because of CIS set that avoids unnecessary closure checking. Figure 7(b) represents running time of approaches against the window size, and observed that running time goes up when window size increases. The reason for such change is more number of transactions in a window generates huge combinations. MCST shows good performance because of IHS and CIS hash tables.

Figure 7(c) shows the comparison of MCST and Fideo-close w.r.t minimum threshold and running time when window size is 100.In connect dataset, each transaction is recorded with on average 78% of items, then the possibility of itemsets are more. Hence, computation time for CIS increases when threshold decreases. The characteristics of MCST framework takes less running time than other models. Figure 7(d) shows comparison w.r.t window size and running time. MCST shows good performance because of IHS and CIS hash tables.

VI. CONCLUSION

We propose MCST approach for mining closed itemsets from the data streams, in which some of the tuples gets updated. Unlike to traditional data streaming techniques, MCST removes tuples that gets updated from the SlideTree as well as when sliding windows moves on incoming transactions. MCST doesn't need to visit the entire previous slide to update knowledge. MCST improves the performance of searching hash tree, by limiting closure checking with only its related itemsets with the help of CIS set.It does not require to compare all the itemsets.Experiments shows that it outperforms all the approaches.

REFERENCES

1. A. Moga, I. Botan, and N. Tatbul. Upstream: storage-centric load management for streaming applications with update semantics. VLDB Journal, 20(6):867–892, 2011.
2. Aliberti, G, Colantonio, A, Di Pietro, R, Mariani, R. EXPEDITE: EXPressclosEDITemset Enumeration. Expert Systems with Applications, 2015, 42(8):3933–3944.
3. Anamika Gupta, VasudhaBhatnagar, Naveen Kumar:Mining Closed Itemsets in Data Stream using Formal Concept Analysis. DaWaK 2010.
4. Ariyam Das, Carlo Zaniolo Fast Lossless Frequent Itemset Mining in Data Streams using Crucial Pattterns.SIAM-2016.
5. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In PODS, pages 1–16, 2002.
6. B. Mozafari, H. Thakkar, and C. Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In ICDE, pages 179–188, 2008.
7. BhargaviPeddireddy, Ch. Anuradha, P.S.R. Chandra Murthy: An Approach for Mining Mining Frequent Itemsets from Tuple-Evolving Data Streams, in IJITEE, Vol-8, Issue-7, May-2019.
8. BhargaviPeddireddy, Ch. Anuradha, P.S.R. Chandra Murthy: Mining Maximal Frequent Itemsets from Tuple-Evolving Data Streams, in IJRTE, Vol-8, Issue-1, may-2019.
9. C. Zhang, F. Masseglia, and X. Zhang. Modeling and clustering users with evolving profiles in usage streams. In TIME, pages 133–140, 2012.
10. C. Zhang, F. Masseglia, and Y. Lechevallier. Abs: The anti-bouncing model for usage data streams. In IEEE ICDM, pages 1169–1174, 2010.
11. Chang, Y-I., Li, C-E. andPeng, W-H. (2012) 'An efficient subset-lattice algorithm for miningclosed frequent itemsets in data streams', in Proceedings of Conference on TechnologiesandApplications of Artificial Intelligence.
12. Chi, Y., Wang, H., Yu, P.S. and Muntz, R.R. (2006) 'Catch the moment: maintaining closed frequent itemsets over a data stream sliding window', Knowledge and Information Systems, Vol. 10, No. 3, pp.265–294.
13. Chongsheng Zhang, Yuan Hao, MirjanaMazuran, Carlo Zaniolo, Hamid Mousavi, and FlorentMasseglia. Mining frequent itemsets over tuple-evolving data streams. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, pages 267–274. ACM, 2013
14. E. Ryvkina, A. Maskey, M. Cherniack, and S. B. Zdonik. Revision processing in a stream processing engine: A high-level design. In ICDE'06, pages 141–144, 2006.
15. H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, V. Russo, and C. Zaniolo. Smm: A data stream management system for knowledge discovery. In ICDE, pages 757–768, 2011.
16. J. H. Chang and W. S. Lee. stWin: adaptively monitoring the recent change of frequent itemsets over online data streams. In ACM CIKM, pages 536–539, 2003.
17. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In SIGMOD Conference, pages 1–12, 2000.
18. Jian J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In SIGMOD Int'l Workshop on Data Mining and Knowledge Discovery, May 2000.
19. L. Golab and M. T. Ozsu. Update-pattern-aware modeling and processing of continuous queries. In SIGMOD Conference, pages 658–669, 2005.
20. Liu, X., Guan, J. and Hu, P. (2009) 'Mining frequent closed itemsets from a landmark window overonline data streams', Journal of Computers and Mathematics withApplications, Vol. 57,No. 6, pp.927–936.
21. M. JeyaSutha; F. Ramesh Dhanaseelan: Mining Frequent, Maximal and Closed Frequent Itemsets over Data Stream- a review. International Journal of Data Analysis Techniques and Strategies, Volume 9 Issue-1, Jan 2017.
22. Nori, F., Deypir, M. and Sadreddini, M.H. (2013) 'A sliding window based algorithm for frequentclosed itemset mining over data stream', The Journal of Systems and Software, Vol. 86, No. 3,pp.615–623.
23. P. Haghani, S. Michel, and K. Aberer. Evaluating top-k queries over incomplete data streams. In CIKM, pages 877–886, 2009.
24. Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules. In: Proc. Intern. Conf. Database Theory, Jerusalem, Israel, 10-12 January, 1999:398–416.
25. Uno, T, Kiyomi, M, Arimura, H. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. Proc. ICDM'04 Workshop on Frequent Itemset Mining Implementations, CEUR, 2004.
26. Yen, S-J., Wu, C-W., Lee, Y-S., Tseng, V.S. and Hsieh, C-H. (2011) 'A fast algorithm for miningfrequent closed itemsets over stream sliding window', in Proceedings of IEEE InternationalConference on Fuzzy Systems, June 27–30, Taipei, Taiwan.
27. Zhang, G., Lei, J. and Wu, X. (2010) 'Mining frequent closed itemsets over data stream based on bit vector and digraph', in Proceedings of 2nd International Conference on Future Computer and Communication, Vol. 2, pp.241–246.