I.J. Image, Graphics and Signal Processing, **2022, 5, 72-83**
Published Online on October 8, 2022 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijigsp.2022.05.06

Sliding Window Based High Utility Item-Sets Mining over Data Stream Using Extended Global Utility Item-Sets Tree

P. Amaranatha Reddy
Research scholar, Department of CSE, UCE, JNTU Kakinada
Email: amaranatha.p@gmail.com

MHM Krishna Prasad
Professor, Department of CSE, UCE, JNTU Kakinada,
Email: krishnaprasad.mhm@gmail.com

Received: 12 February 2022; Accepted: 25 April 2022; Published: 08 October 2022

Abstract: High utility item-sets mining(HUIM)is a special topic in frequent item-sets mining(FIM). It gives better insights for business growth by focusing on the utility of items in a transaction. HUIM is evolving as a powerful research area due to its vast applications in many fields. Data stream processing, meanwhile, is an interesting and challenging problem since, processing very fast generating a huge amount of data with limited resources strongly demands high-performance algorithms. This paper presents an innovative idea to extract the high utility item-sets (HUIs) from the dynamic data stream by applying sliding window control. Even though certain algorithms exist to solve the same problem, they allow redundant processing or reprocessing of data. To overcome this, the proposed algorithm used a trie like structure called Extended Global Utility Item-sets tree (EGUI-tree), which is flexible to store and retrieve the mined information instead of reprocessing. An experimental study on real-world datasets proved that EGUI-tree algorithm is faster than the state-of-the-art algorithms.

Index Terms: Data mining, high utility item-sets mining, stream mining, sliding window.

1. Introduction

Data Mining techniques are useful to acquire enough knowledge to take certain decisions from the huge amounts of available data. Frequent item-sets mining (FIM) is one of the primitive data mining tasks [1, 2]. It extracts the item-sets having a frequency of occurrence more than user specified cut-off from the transactional database. Most of the basic FIM algorithms are designed to work on binary transactional databases. They assume, each item in a transaction is of one unit quantity and are equally profitable. However, in real life, each item generates different profit (also called external utility) and often their purchase quantity (also called internal utility) is not the same. For example, the quantity and profit generated by the items like rice packet and gold ring are for away from each other[3]. Hence, the same kind of perception about all the items may not produce valuable results. The Utility item-sets mining[4] came into the picture by addressing these issues. Even though utility item-sets are more useful compared with frequent item-sets, UIM is hard and intractable. The reason is FIM methods support anti-monotone property (or downward closure property) over frequency of item-sets which prune search space effectively whereas utility item-sets do not hold the property[5]. The property states that any superset of an infrequent item-set cannot be frequent[6].

Utility item-sets mining is created a space for new applications, those address the challenges in the areas of market basket analysis, web click stream analysis, wireless sensor networks, bio-medical data analysis, and stock market prediction[7,8]. It is also been combined with other mining techniques like sequential pattern mining, episode pattern mining, stream mining, top-k high utility item-sets mining, high utility item-sets mining, high utility rare pattern mining, high average utility item-sets mining[4, 9, 10, 11, 12]. This study aims to develop a high performance algorithm to retrieve high utility item-sets over data stream. It extracts item-sets having utility more than the user specified cutoff.

In a data stream, transactions come as a flow one after the other with high speed and the volume of transactions can be very large. Hence, it is no longer possible to store entire data into internal memory and process it because of limited resources. To address this, the continuously flowing data in the stream is need to cut into small chunks called batches. The set of batches used for processing is treated as a window. There are different kinds of window models available such as landmark window, sliding window and damped window [13]. The common activity in all the models

is the construction of the window by filling it with some set of batches. But, they differ by the way preference is given to the processed window result. The damped window model reduces the importance of results produced by the window as time passes. Secondly, the sliding window model considers only the current window result. Finally, the landmark window model uses the results of windows between the current point of time to the stipulated point of time.

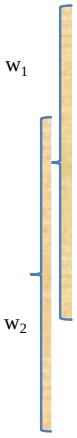
This work is motivated by a fixed-size sliding window model to process stream data. In this model, the size of the window is fixed by the user and cannot be changed in the middle of the process. Every time when a new batch is full with the recent data, it is added to the window by removing the oldest batch. This process is called window sliding. Let us assume, the window size is 'm' (i.e. each window posses 'm' number of batches.) and batch size is 'n' (i.e. each batch posses 'n' number of transactions.). It is it could be represented as W1 = {b1 ,b2 …bm} where, b1 = {t1, t2, ...tn }, b2 = {tn+1, tn+2, ...t2n}...etc. After a slide the newly generated window W2 = {b2 ,b3 …bm+1}. Whenever the new window is ready, It is sent as an input to the data mining algorithms. The underlying fact about processing only recent data is, It gives useful and appropriate knowledge to take decisions suitable to the current trend in the fields like marketing.

Several algorithms are available to solve the same problem in the same environment. However, still, there is a scope to improve the performance of algorithms. In the sliding window model, side by side windows has (window size - 1) common batches. Suppose, w1 having batches b1, b2, b3 then definitely w2 posses batches b2, b3, b4. It could be observed that both windows w1 and w2 having batches b2 , b3 as common. Most of the existing algorithms reprocess the common batches even though their processing is done in the previous window. It affects the performance of algorithm badly in terms of execution cost. This paper addresses the issue by introducing a trie like structure called Extended Global Utility Item-sets tree (EGUI-tree) for storing mined information of a window before it slides down. which saves the execution time by eliminating the redundant processing cost. Overall, the algorithm certainly proved its high performance to process the data stream. The remaining part of this paper is segregated into the following sections. Section 2, 3, 4 and 5 respectively present the problem statement and related work, proposed EGUI-tree algorithm and description, performance evaluation and finally the conclusion.

2. Problem Statement and Related Work

Preliminary definitions are placed at the beginning of this section. Let us consider, in a market basket databasei1, i2, …, in be the set of items and t1, t2, …, tm be the set of transactions denoted with 'I' and 'D' sequentially. Here, transaction is nothing but set of items purchased by the customer at a particular time[14]. Each item ij in a transaction tq is followed by it's internal utility IU(ij, tq)(in most of the cases it is purchase quantity). The external utility of each item EU(ii) (in most of the cases it is profit)is placed in a separate table. The sample data provided in Table 1, Table 2is used to demonstrate the preliminary definitions and related work. The minutil value is assumed as 150 throughout the paper.

Table 1. A sample database D.



BID	TID	Items in transaction with Internal Utility	TU
b_1	t_1	l(1), n(18), p(1)	27
	t_2	m(6), n(1), o(10), p(1), q(1)	67
	t_3	l(2), n(1), p(1)	13
b_2	t_4	o(1), p(1)	11
	t_5	n(4), p(2)	16
	t_6	m(1), q(1)	10
b_3	t_7	m(10), o(1), p(1)	101
	t_8	l(3), n(25), o(3), p(1), q(4)	59
	t_9	l(1), m(1), q(3)	15
b_4	t_{10}	m(6), n(2), o(2), p(2)	78
	t_{11}	l(4), n(2), o(5)	39
	t_{12}	n(3), o(2), q(3)	16

Table 2. External utility values.

Item	1	m	n	o	p	q
External Utility(EU)	3	9	1	5	6	1

Def. 1.Item utility in a transaction: The utility of an item i_j in a transaction $t_q \in D$ is represented as $U(i_j, t_q)$,and is calculated by,

$$U(i_j, t_q) = IU(i_j, t_q) \times EU(i_j) \quad (1)$$

For example, $U(l,\ t_1) = IU(l,\ t_1) \times EU(l) = 1 \times 3 = 3.$

Def. 2.Item-set utility in a transaction: The utility of an item-set α = {i_1, i_2, … i_s} in transaction t_q is represented as U(α, t_q),and is calculated by,

$$U(\alpha, t_q) = \sum_{j=1}^{s} U(\alpha, t_q) \tag{2}$$

For example, $U\left(\{l,n\},\ t_1\right) = U\left(l,\ t_1\right) + U\left(n,\ t_1\right) = 3 + 18 = 21.$

Def. 3.Item-set utility in database D: The utility of an item-set α= {i_1, i_2, … i_s} in database D is represented as U(α), and is calculated by,

$$U\left(\alpha\right) = \sum_{(X \subseteq t_q) \wedge (t_q \in D)} U\left(\alpha,\ t_q\right) \tag{3}$$

For example, $U\left(\{m,o\}\right) = U\left(\{m,o\}, T_2\right) + U\left(\{m,o\}, t_7\right) = 59 + 95 = 154.$

Def. 4.Transaction utility: The transaction utility of a transaction t_q is represented as TU(t_q), and is calculate by,

$$TU\left(t_q\right) = \sum_{j=1}^{n} U\left(i_j,\ t_q\right) \tag{4}$$

For example, $TU\left(t_2\right) = U\left(m,\ t_2\right) + U\left(n,\ t_2\right) + U\left(o,\ t_2\right) + U\left(p,\ t_2\right) + U\left(q,\ t_2\right) = 1 + 1 + 54 + 5 + 6 = 67.$

Def. 5. Minimum threshold utility: It is represented as minutil, and used to extract high utility item-sets. Its value ranges from 0 to 100.Let δ be the percentage value fixed by the user then the minutil is calculated as follows.

$$minutil = \left(\sum_{t_q \subseteq D} TU\left(t_q\right)\right) \times \delta / 100 \tag{5}$$

For example, suppose δ is set to 47%. For the first 9 transactions minutil is calculated as $\left(27 + 67 + 13 + 11 + 16 + 10 + 101 + 59 + 15\right) \times 0.47 = 150.$

Def. 6.Transaction weighted utility of an item-set: The transaction weighted utility of an item-set X is represented as TWU(X), and is calculated by,

$$TWU(X) = \sum_{(X \subseteq t_q) \wedge (t_q \in D)} TU\left(t_q\right) \tag{6}$$

For example from the first 9 transactions, $TWU\left(l\right) = TU\left(t_1\right) + TU\left(t_3\right) + TU\left(t_8\right) + TU\left(t_9\right) = 27 + 13 + 59 + 15 = 114.$

Def. 7. High utility item-set: An item-set is treated as a high utility item-set, if it is having utility greater than or equal to minutil in the database[15].For example, MO is a high utility item-set as it's utility in database is 154 which is greater than minutil 150.

2.1 Problem Statement:

Let us consider, D is a transactional data stream. The sliding window algorithm catches each arriving transaction and added to a batch. Once the batch is full, it is appended to the window. If the window is already full, the oldest batch is dropped to accommodate the new batch and release the recent window. The batch and window are represented as b ={t1, t2, t3, ... tm}, w ={b1, b2, ... bm } sequentially. Here, 'm' is the batch size and 'n' is window size. These 'm' and 'n' values are set by a data analyst. For example, in table 1, 'm' and 'n' values are taken as 3. Hence, the first three transactions go to b1, the next three go to b2 and continue further likewise and fully loaded batches added to the window. Therefore, the current window w1 contains b1, b2, and b3 batches. After some time whenever b4 is full with the transactions, a new window w2 is released with batches b2, b3, and b4. The objective of this work is to retrieve all HUIs from the transactions of a very recent window[16].

2.2 Related work:

Real-time data streams like supermarket transactions contain a huge number of transactions and again there could be many items in each transaction. Pattern mining algorithms takes many steps to process such data. These reasons badly influence the performance of algorithms with respect to execution time. The following two parameters need to take care to get the best performance from the pattern mining algorithms.

First one is, minimize the number of database scans as well as candidate item-sets generation. Apriori based algorithms[17] generates 'k+1' length candidate item-sets at each level by combining 'k' length frequent item-sets of the previous level. A new database scanning is required at each level to filter frequent item-sets from the candidate item-

sets. Tree-based algorithms somehow short out this problem by limiting database scans to two. But, they may not support dynamic databases as the nodes of the tree follow some order in the construction which should not be disturbed. The second one is the search space pruning. Traditional HUIM algorithms use TWU to prune the search space, but in dynamic environments like data streams, it is helpless.

In the earlier stage of HUIM using the sliding window control, SHUPM [16] and HUPMS[18] have been proposed. SHUPM employs the construction and update of global utility item-sets. However, this algorithm performs the redundant scanning of global lists for every window slide to produce extended lists. On the other hand, HUPMS reduces the reprocessing cost by using a tree structure in order to store the processed data results. The major obstacle to the performance here is the construction of prefix trees for mining high utility item-sets after each window slide.

SHUPM and HUPMS are state-of-the-art HUIM algorithms to process stream data by using sliding window control. However, these two algorithms have certain performance bottlenecks. Therefore, a new algorithm is proposed in this paper called EHUI-tree algorithm, which proved to be outperform than the sate-of-the-art algorithms.

3. Proposed Method

This section contains a key idea of the paper. The main procedure (algorithm 1) takes as input a transaction database with utility values, minutil, batch size, and window size. The overall process is divided into two stages. In the first stage, only the first window w1 is scanned and following are the sequence of activities performed. i. calculate the TWU of each item. ii. build the global utility 1-item sets. In the second stage, scanning of data in each recent window is performed starting from first window w1 and the following are sequence of activities performed. i. reorganize the transactions. ii. update the global utility 1-item sets. iii. filter the eligible global utility item-sets for extension iv. extend the eligible global utility item-sets v. build/update the EGUI-tree vi. mine high utility item-sets from the EGUI-tree. vii. renew the result for each recent window release. These steps are elaborated in detail as follows.

Algorithm 1: The EGUI-Tree algorithm
Input-D: a stream of transactions
minutil: a user specified utility cutoff
m: batch size
n: window size
output- H: high utility item-sets
//initially T number, B number, W number =0
1. for each Ti in D
2. fill Bj with T
3. if Bj is full, add it to Wk
4. if Wk is full
5. if k equal to 1
6. calculate TWU of each item
7. order items according to ascending of TWU values
8. construct global utility 1-item sets structure
9. end if
10. reorganize the transactions
11. update the global utility 1-item sets
12. mine_HUI(build_EGUI-tree (extend(1-item sets, minutil)))
13. remove B having least j value in W, 1-item sets and EGUI-tree
14. increment the k value
15. end if
16. increment j value
17. end if
18. increment i value
19. end for

Algorithm 2: extend procedure
input- x: item-set
minutil: a user specified utility cutoff
output- EL: extended list of item-sets

```
1.if  TU(x)+RU(x) is greater than or equal to minutil
2.for each item y in after x
3.   if TU(x)+TU(y) + RU(y) is greater than minutil
4.        if compare (x, y) not null
5.             update_EGUI-tree(xy, TU(xy), RU(xy)
6.               end if

7.          end if
8.          extend(xy, minutil)
9.     end for
10.end if
```

Algorithm 3: The compare procedure
input: X, Y: global utility lists
output: EL {BID: batch id, TU(XY): transaction utility, RU(XY): remaining utility}: extended list of item-sets

```
1. for each batch in X
2.       for each batch in Y
3.   if x.bid is equal to y.bid
4.        for each transaction t in bid
5.             if x.tid is equal to y.tid
6.                TU(xy)←TU(x)+TU(y)
7.                RU(XY)←RU(Y);
8.             end if
9.        end for
10. end if
11.      end for
12. end for
```

Algorithm 4: build_EGUI-tree procedure
input- Ex: extended item
output- updated EGUI-tree

```
1. search for Ex in EGUI-tree using BFS
2. if node not found
3.   create a new node
4. end if
5. update branch utility
6. end
```

Algorithm 5: mine_HUI procedure
input: EGUI-tree: Extended Global Utility Item-sets tree
output: HUI: list of high utility item-sets

```
1. traverse each node of the tree by comparing with minutil
2. if node utility is greater than or equal to minutil
3.   print node item and utility
4. end if
5.end
```

Once the first window w1 is completely filled with batches, the analysis part starts with scanning of transactions of w1. The following step (section 3.1.1) perform in parallel by synchronizing with the scanning. Thereafter next step (section 3.1.2) will perform.

3.1.1. Calculate TWU of items.

Here, the TWU values of each item present in w1 were calculated and they are used in the entire mining process. For example, in the table 1 the first window i.e. w1contains b1, b2, b3batchesand TWU values of each item is shown in Fig.1.

l:114		
BID	TID	TWU
b_1:40	t_1	27
	t_3	13
b_3:74	t_8	59
	t_9	15

m:193		
BID	TID	TWU
b_1:67	t_2	67
b_2:10	t_6	10
b_3:116	t_7	101
	t_9	15

n:182		
BID	TID	TWU
b_1:107	t_1	27
	t_2	67
	t_3	13
b_2:16	t_5	16
b_3:59	t_8	59

o:238		
BID	TID	TWU
b_1:67	t_2	67
b_2:11	t_4	11
b_3:160	t_7	101
	t_8	59

p:294		
BID	TID	TWU
b_1:107	t_1	27
	t_2	67
	t_3	13
b_2:27	t_4	11
	t_5	16
b_3:160	t_7	101
	t_8	59

q:151		
BID	TID	TWU
b_1:67	t_2	67
b_2:10	t_6	10
b_3:74	t_8	59
	t_9	15

Fig.1. TWU values of items

3.1.2.Build global utility 1-item sets.

Global utility item-sets are the structures specially designed for each item to store TU and RU values along with TID and BID. Initially, global utility 1-item sets were built for each item in w1 and TU, RU initialize with '0'. These items need to be arranged in the increasing order of their TWU values[16,19, 20]. The order of items available in w1 is l, q, n, m, o, p. Since, the TWU of l < q < n < m < o < p. For example, the global utility 1-item sets for items of w1 given in Fig.1 is shown in Fig.2.

l			
BID	TID	TU: 0	RU:0

q			
BID	TID	TU:0	RU:0

n			
BID	TID	TU:0	RU:0

m			
BID	TID	TU:0	RU:0

o			
BID	TID	TU:0	RU:0

p			
BID	TID	TU:0	RU:0

Fig.2. Template for global 1-item sets

Def. 8. Reorganized transaction: A transaction 't'is said to be reorganized, if the items present in 't' are arranged in the increasing order of their TWU values[21].For example, reorganized transaction for t_2=(m, n, o, p, q) is (q, n, m, o, p) because TWU of (q=151) < (n=182) < (m=193) < (o=238) < (p=294).

Def. 9. Set of items after item-set 'α': Given an item-set 'α' and a reorganized transaction 't' with $\alpha \subseteq t$, the set of items after 'α' in 't' is represented as A(t /{α}) and are available in 't' followed by 'α'. For example, set of items after item-set (q , n) in $t_2$given by A(t_2/{q,n}) = {m, o, p}

Def. 10.Remaining Utility of an item-set 'α': The remaining utility[22] of item-set 'α' in a reorganized transaction t_q is represented as RU({α}, t_q) and is calculated by,

$$RU(\alpha, t_q) = \sum_{i_k \subseteq t_q/\alpha} U(i_k, t_q) \tag{7}$$

For example, $RU(n, t_2) = U(m, t_2) + U(o, t_2) + U(p, t_2) = 54 + 5 + 6 = 65.$

After the completion of scanning w1and consequent steps, global utility1-item sets come to available. A second scanning starts from the first window and goes on for each recent window (i.e. w1, w2, w3....). The following sequence of steps perform in this stage.

3.2.1 Reorganize the transactions

While scanning transactions of the current window, each transaction needs to be reorganized (def. 8) to participate in the further processing steps. Reorganized transactions for Table 1 are given in Table 3.

Table 3. Reorganized transactions for Table 1.

	BID	TID	Reorgnaized transaction with utility	TU
w_1	b_1	t_1	l:3, n:18, p:6	27
		t_2	q:1, n:1, m:54, o:5, p:6	67
		t_3	l:6, n:1, p:6	13
	b_2	t_4	o:5, p:6	11
		t_5	n:4, p:12	16
		t_6	q:1, m:9	10
w_2	b_3	t_7	m:90, o:5, p:6	101
		t_8	l:9, q:4, n:25, o:15, p:6	59
		t_9	l:3, q:3, m:9	15
	b_4	t_{10}	n:2, m:54, o:10, p:12	78
		t_{11}	l:12, n:2, o:25	39
		t_{12}	q:3, n:10, o:3	16

3.2.2. Update the global utility 1-item sets

The updating process of global utility 1-item sets is performed immediately after the reorganization of each transaction. For each item in reorganized transaction,the TU (def. 4) and RU (def. 10) values update under the corresponding global utility 1-item sets. For example, reorganized transaction t1= (l:3, n:18, p:6) register entries in global utility 1-item sets under batch b1 asl:(TU=3, RU=24), n:(TU=18, RU=6) and p:(TU=6, RU=0). In the same way the updated global utility 1-item sets for transactions in window 'w1'of table. 3 is shown in the Fig 3.

l			
BID	TID	TU:21	RU:93
b_1 TU:9 RU:31	t_1	3	24
	t_3	6	7
b_3 TU:12 RU:62	t_8	9	50
	t_9	3	12

q			
BID	TID	TU:9	RU:130
b_1 TU:1 RU:66	t_2	1	66
b_2 TU:1 RU:9	t_6	1	9
b_3 TU:7 RU:55	t_8	4	46
	t_9	3	9

n			
BID	TID	TU:49	RU:110
b_1 TU:20 RU:77	t_1	18	6
	t_2	1	65
	t_3	1	6
b_2 TU:4 RU:12	t_5	4	12
b_3 TU:25 RU:21	t_8	25	21

m			
BID	TID	TU:162	RU:22
b_1 TU:54RU:11	t_2	54	11
b_2 TU:9 RU:0	t_6	9	0
b_3 TU:99RU:11	t_7	90	11
	t_9	9	0

o			
BID	TID	TU:30	RU:24
b_1 TU:5 RU:6	t_2	5	6
b_2 TU:5 RU:6	t_4	5	6
b_3 TU:20 RU:12	t_7	5	6
	t_8	15	6

p			
BID	TID	TU:48	RU:0
b_1 TU:18 RU:0	t_1	6	0
	t_2	6	0
	t_3	6	0
b_2 TU:18 RU:0	t_4	6	0
	t_5	12	0
b_3 TU:12 RU:0	t_7	6	0
	t_8	6	0

Fig.3. Updated global utility 1-item sets

property 1(Eligibility for an item-set to participate in the extension). Let, 'α' be an item-set then the remaining utility upper bound of 'α' is defined as RUB(α)=TU(α)+RU(α). If RUB(α)< minutil, then 'α' is a low utility item-set as well as all its extensions. Hence, 'α' is not eligible to participate in the extension operation.

property 2(Eligibility for items to join with eligible item-sets). Let 'α' be an eligible item-set to participate in the extension by property 1. An extended item-set of 'αi_k' can be obtained by appending an item i_k to 'α' such that, 'i_k' is the successor of the last item in 'α' according to the global utility 1-item sets sequence. The remaining utility upper bound of 'αi_k' is defined as RUB(αi_k)=TU(α)+TU(i_k)+RU(i_k). If RUB(αi_k)≱minutil, then 'αi_k' is a low utility item-set as well as all its extensions. Hence, 'i_k' is not eligible to join with 'α'.

property 3 (TU and RU values of extended item-set). Let 'a' is an eligible item-set to participate and 'b' is the eligible item to join with 'a' in the extension. Then, the TU(ab)=TU(a)+TU(b) and RU(ab)=RU(b).

After the completion of global utility 1-item sets updation, the following two steps (section 3.2.3, section 3.2.4) repeatedly execute one after the other in a loop until there is no eligible item to participate in extension operation.

3.2.3. Filter eligible global utility item-sets to participate in the extension.

Here, the set of eligible global utility item-sets from the set of candidate global utility item-sets filter at each level by using property 1. (i.e. 1-item sets in the first level, 2-item sets in the second level... etc). For example, among the global utility 1-item sets (l, q, n, m, o, p) only 'n' and 'm' are eligible to participate in the extension. Since, RUB(l)=114≱ minutil, RUB(q)=139≱ minutil, RUB(o)=54≱ minutil, RUB(p)=48≱ minutil. but, RUB(n)=159> minutil and RUB(m)=184> minutil.

3.2.4. Extend the eligible global utility item-sets.

Here, joining of eligible item-sets with after items (def. 9) is performed to detect high utility item-sets in the next level[23] by using the property 2. From these item-sets, extended utility lists are prepared by computing the utility and remaining utility as given in algorithm 2. For example, an item 'n' is eligible to participate in the extension and after item of 'n' are (m, o, p). Only 'm' is eligible to join with 'n'. Since, RUB(nm)=233>minutil but, RUB(no)= 103 ≱minutil and RUB(np)=97≱ minutil . The entries in the extended list is made by comparing the batch ids followed by transaction ids. If the two lists have common transactions, then the entries will be added to the newly extend list along with TU and RU values computed by using property 3. Similarly the list of eligible item-sets and possible extensions for the global 1-item sets given in Fig.3 is shown in Fig. 4.

n

BID	TID	TU:49	RU:110
b_1 TU:20 RU:77	t_1	18	6
	t_2	1	65
	t_3	1	6
b_2 TU:4 RU:12	t_5	4	12
b_3 TU:25 RU:21	t_8	25	21

nm

BID	TID	TU:55	RU:11
b_1 TU:55 RU:11	t_2	55	11

mo

BID	TID	TU:154	RU:12
b_1 TU:59 RU:6	t_2	59	6
b_3 TU:95 RU:6	t_7	95	6

mop

BID	TID	TU:166	RU:0
b_1 TU:65 RU:0	t_2	65	0
b_3 TU:101 RU:0	t_7	101	0

m

BID	TID	TU:162	RU:22
b_1 TU:54 RU:11	t_2	54	11
b_2 TU:9 RU:0	t_6	9	0
b_3 TU:99 RU:11	t_7	90	11
	t_9	9	0

mp

BID	TID	TU:156	RU:0
b_1 TU=60 RU=0	t_2	60	0
b_3 TU=96 RU=0	t_7	96	0

Fig. 4. Extended eligible global utility item-sets

3.2.5. Build/update EGUI-tree

The key idea is as follows. Suppose, window wn = {b1, b2, ..., bn-1, bn} then wn+1={b1, b2, ...bn, bn+1}. Here, except 'b1' remaining all batches of wn (i.e. b2, b3 ....., bn) are also present in 'wn+1'. The removal of the oldest batch and appending a new batch form a new window. Hence, It could be observed that (window size-1) batches are common in all adjacent windows. To analyze the transactions in wn, the batches present in it i.e. b1, b2, ..., bn-1, bn needs to process. Similarly for wn+1, the batches b1, b2, ...bn, bn+1 needs to process. The processing of all the batches in each window involves a lot of redundant work because of common batches.

Hence, there is a necessity to have an efficient data structure for storing the mined information [24] of each batch, which could be utilized in the processing of further sliding windows. The solution we adopted here is to store all extended global utility item-sets in a trie like structure named EGUI-tree. It is efficient to store and retrieve extended item-sets along with its batch-wise utility values. The construction of EGUI-tree starts with a null node called root. All the eligible 1-itemsets for extension join with a link to the root. Thereafter all the possible extensions of each node join to them. For example, fig. 5 shows the EGUI-tree constructed with the extended global utility item-sets shown in fig. 4. Insertion/search operations of an item-set 'α' in EGUI-tree is very fast, as it requires to traverse | α | nodes from the root. It requires to only traverse one path in the tree. For example, to search for 'mop' in the EGUI-tree shown in Fig.5, we need to traverse 'm', 'o' nodes from the root.

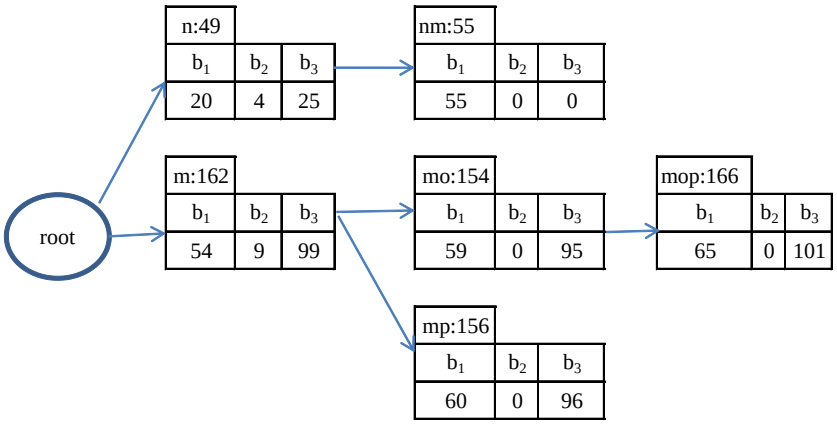


Fig. 5. Extended global utility lists tree

3.2.6. Mining high utility item-sets.

To filter high utility item-sets, a tree traversing mechanism is required. In this approach, a depth-first search traversal performed on EGUI-tree, at each node utility is compared with minutil. If the utility of the item-set at a node is greater than or equal to minutil, then it is treated as a high utility item-set. For example, DFS traversal on the EGUI-tree shown in fig. 5. extracts {m:162}, {mo:154}, {mop:166}, {mp:156} as high utility item-sets. Since their utility is more than minutil value 150.

3.2.7 Renew the result for each window slide

When a window slides down, the current window is built by dropping the oldest batch and appending the newly arriving batch to the previous window. The results obtained from the previous window need to be update to match with the data in the current window. To cope with the rebuilding of results the following rework need to do for each slide of the window. i. Update all global utility item-sets by removing the oldest batch records and adding new batch records. This process alters the list of eligible items for extension (i.e Sum of TU and RU could increase or decrease). For example, updated global utility item-sets for window 2 shown in Fig 6. ii. Traverse the EGUI-tree to make the utility contribution of the leaving batch as -1 at each node. Here, -1 indicates the corresponding entry is free. Fig.7. a. shows the status of EGUI-tree after leaving the batch 'b1' iii. Extend the global utility item-sets and update the newly joined batch utility contribution in EGUI-tree in the place of a recently left batch. In case, a new item gets the eligibility for extension then create a new branch to the root. The updated tree after the insertion of batch4 is shown in Fig.7.b. iv. Finally, traverse the tree to find the renewed high utility item-sets. In the example, item-sets {m:162}, {mo:159}, {mop:177}, {mp:162} are renewed high utility item-sets.

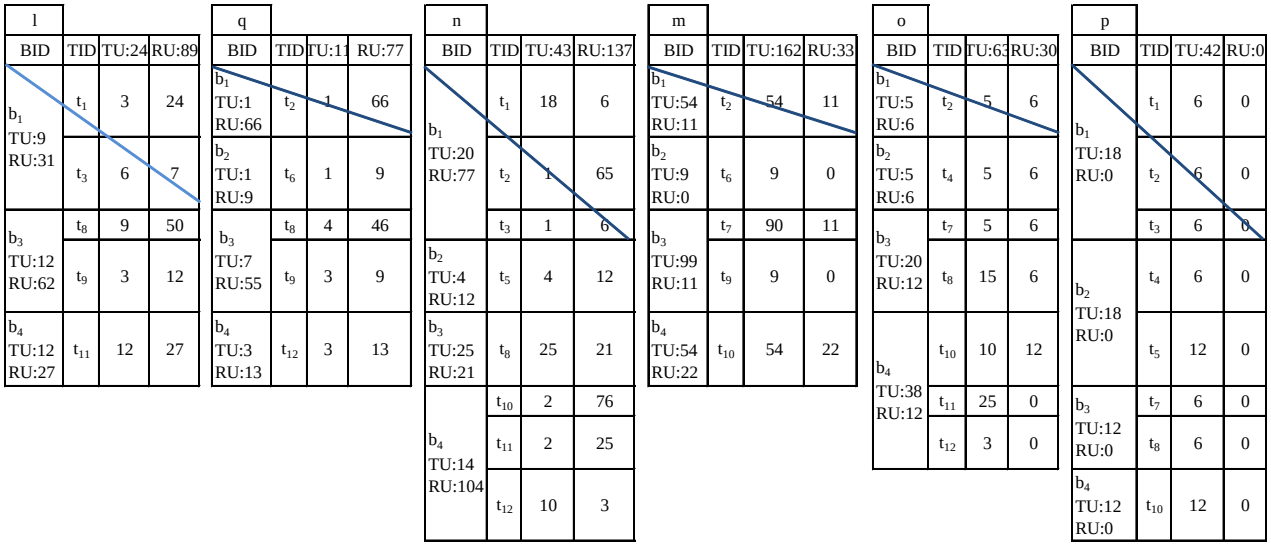


Fig. 6. Updated global utility item-sets for window 2.

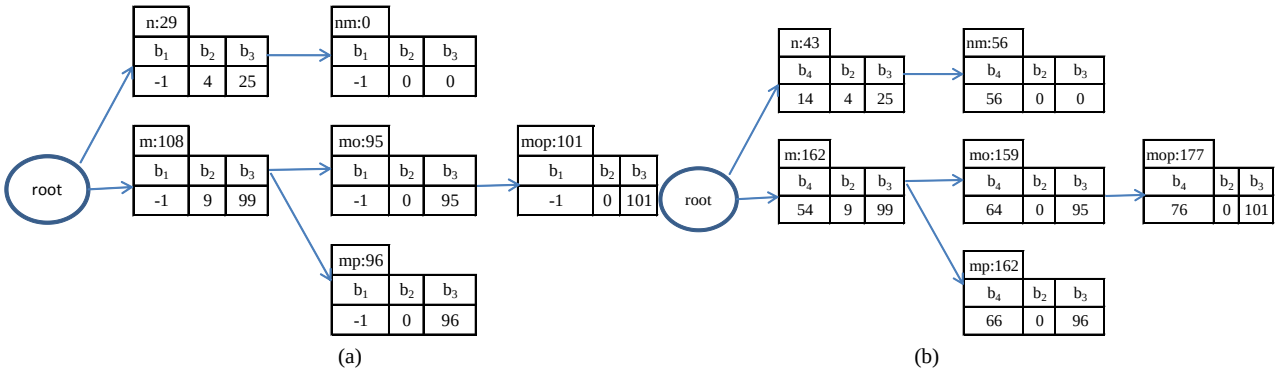


Fig. 7. Updated EGUI-tree (a) After leaving batch B1 (b) After updating batch B4.

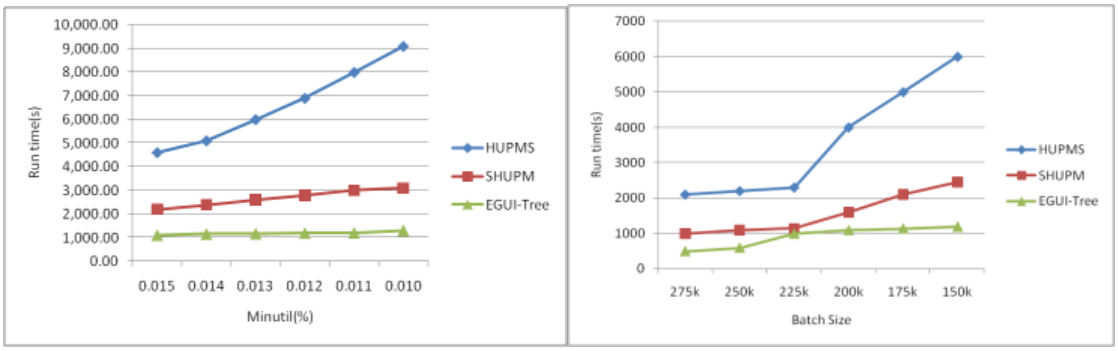
4. Experimental Results

Various experiments were conducted to test the performance of the proposed EGUI-tree algorithm by setting the system environment with a 3.30 GHz processor, and the windows7 operating system with 8 GB RAM. In each experiment execution time of EGUI-tree algorithm is compared with state-of-the-art algorithms SHUPM and HUPMS. The EGUI-tree algorithm is developed by the java programming language by using SPMF (http://www.philippe-fournier-viger.com/spmf/) open-source library. Table 4. shows the description of real-world data set Chain-store (http://cucis.ece.northwestern.edu/projects/ DMS/MineBench.html) used to conduct the experiments.

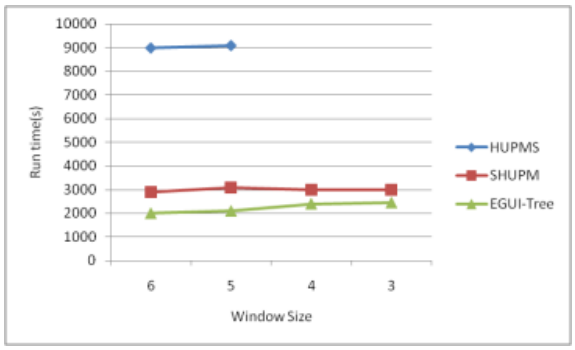
Table 4. Description of chain-store dataset

Characteristic	#value
Number of transactions	11,12,950
Number of distinct items	129
Transaction average length	2.511

Experiments are conducted as follows. Among three parameters such as minutil, batch size, window size, two are fixed and remaining one is varied gradually. Fig. 8(a). shows the execution time taken for varied minutil by fixing batch size and window size as 150 K and 5 respectively. Fig. 8(b). shows the execution time taken for varied batch size by fixing minutil and window size as 0.013% and 5 respectively. Fig. 8(c). shows the execution time taken for varied window size by fixing minutil and batch size as 0.013% and 150K. It can observe that HUPMS and SHUMP are taking more execution time than EGUI-tee in every case. The reason is that for each window SHUMP and HUPMS algorithms reprocess one less than window size batches unnecessarily because of not saving the mined information. EGUI-tree algorithm takes reasonable execution time for the first window as a cost of EGUI-tree construction. After that, for each window slide it takes very less time since, it drags the processed batches information from the EGUI-tree instead of reprocessing them from scratch. Note that, in Fig. 7 (c), HUPMS not terminated in 15000 seconds for window sizes 4 and 5.



(a) (b)



(c)

Fig. 8. Execution time on chain-store dataset under (a) varied minutil (b) varied batch size (c)varied window size.

To test the scalability an experiment was conducted by fixing window size=5, minutil=0.02%, and batch size = 15000 and number of transactions in the data set are gradually increased by using a group of Tx1Nx2Lx3 datasets. Fig. 9 shows the execution time of SHUPM, HUPMS, and EGUI-Tree under given settings. It can notice that the growth in the execution time of EGUI-tree algorithm with respect to an increase in the number of transactions is very less compared with the remaining two algorithms. The reason is that the number of windows increases with the effect of an increase in the number of transactions which leads to more reprocessing. However, the proposed algorithm avoids

reprocessing by using the previously processed results. Which prove that the EGUI-tree algorithm has better scalability performance compared to HUPMS and SHPM algorithms.

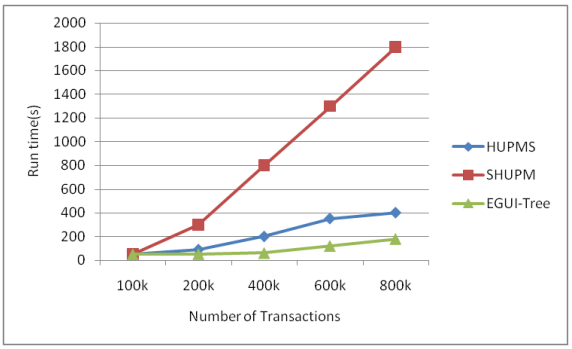


Fig. 9. Scalability of the algorithm under varied number of transactions on T1014Dx.

5. Conclusion

This paper presents an end-to-end study on the problem of extracting recent high utility item-sets by applying sliding window control over the data stream. There always exist a few common batches for all neighboring windows generated in the sliding window mechanism. Processing of all the batches in each window raises a lot of reprocessing cost by virtue of common batches. To address this problem the proposed algorithm uses a trie like structure called EGUI-Tree to store batch-wise utilities of all extended lists of processed windows. This saved information can be utilized in the further coming windows processing. The outcome of conducted Experiments proved that the stated algorithm has optimal execution time and good scalability on large datasets.

References

[1] R. Agrawal, T. Imielinski and A. Swami, "Database mining: a performance perspective," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914-925, Dec. 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, pp. 487-499, 1994.

[3] Dawar, Siddharth & Goyal, Vikram & Bera, Debajyoti, "A hybrid framework for mining high-utility itemsets in a sparse transaction database," *Applied Intelligence*, vol. 47, 2017. 10.1007/s10489-017-0932-1.

[4] P. Fournier-Viger, J. Chun-Wei Lin, T. Truong-Chi, R. Nkambou, "A Survey of High Utility Itemset Mining," *In: Fournier-Viger P., Lin JW., Nkambou R., Vo B., Tseng V. (eds) High-Utility Pattern Mining. Studies in Big Data*, vol 51, 2019. Springer, Cham

[5] Krishnamoorthy, Srikumar, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, pp. 2371-2381, 2015.

[6] Tin Truong, Hai Duong, Bac Le, Philippe Fournier-Viger, "FMaxCloHUSM: An efficient algorithm for mining frequent closed and maximal high utility sequences," *Engineering Applications of Artificial Intelligence*, vol. 85, 2019.

[7] Liu, Junqiang & Wang, ke & Fung, Benjamin, "Mining High Utility Patterns in One Phase without Generating Candidates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, pp. 1245-1257, 2016. 10.1109/TKDE.2015.2510012.

[8] P. A. Reddy and M. H. M. K. Prasad, "Challenges to find association rules over various types of data items: A Survey," *2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida*, pp. 180-184, 2017.

[9] Fournier Viger, Philippe & Lin, Chun-Wei & Rage, Uday & Koh, Yun Sing & Thomas, Rincy, "A Survey of Sequential Pattern Mining," *Data Science and Pattern Recognition*, vol. 1, pp. 54-77, 2017.

[10] C. Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Tzung-Pei Hong, and Hamido Fujita, "A survey of incremental high-utility itemset mining," *Wiley Int. Rev. Data Min. and Knowl. Disc.*, vol. 8, no. 2, March 2018.

[11] T. Truong-Chi, P. Fournier-Viger, "A Survey of High Utility Sequential Pattern Mining," *In: Fournier-Viger P., Lin JW., Nkambou R., Vo B., Tseng V. (eds) High-Utility Pattern Mining. Studies in Big Data, Springer, Cham*, vol 51, 2019.

[12] Zhang, Chongsheng & Almpanidis, George & Wang, Wanwan & Liu, Changchang, "An Empirical Evaluation of High Utility Itemset Mining Algorithms," *Expert Systems with Applications*, 2018.

[13] V. Lee, R. Jin, G. Agrawal, "Frequent Pattern Mining in Data Streams," *In: Aggarwal C., Han J. (eds) Frequent Pattern Mining. Springer, Cham*, 2014.

[14] Bai, P. S. Deshpande and M. Dhabu, "Selective Database Projections Based Approach for Mining High-Utility Itemsets," *in IEEE Access*, vol. 6, pp. 14389-14409, 2018.

[15] Jerry Chun-Wei Lin, Jiexiong Zhang, Philippe Fournier-Viger, Tzung-Pei Hong, Ji Zhang, "A two-phase approach to mine short-period high-utility itemsets in transactional databases," *Advanced Engineering Informatics*, vol. 33, pp. 29-43, 2017.

[16] U. Yun, G. Lee and E. Yoon, "Efficient High Utility Pattern Mining for Establishing Manufacturing Plans With Sliding Window Control," in *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7239-7249, Sept. 2017.

[17] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," *in Proc. 9th Pacific-Asia Conf. Knowl. Discovery Data Mining*, pp. 689–695, May 2005.

[18] H. Ryang and U. Yun, "High utility pattern mining over data streams with sliding window technique," *Expert Syst. Appl.*, vol. 57, pp. 214–231, Sep. 2016.

[19] F Duong, Quang-Huy & Fournier Viger, Philippe & Ramampiaro, Heri & Nørvåg, Kjetil & Dam, Thu-Lan, "Efficient high utility itemset mining using buffered utility-lists," *Applied Intelligence*, 2017.
[20] Fournier Viger, Philippe & Zhang, Yimin & Lin, Chun-Wei & Fujita, Hamido & Koh, Yun Sing, "Mining Local and Peak High Utility Itemsets," *Information Sciences*, 2019.
[21] Dawar, Siddharth & Sharma, Veronica & Goyal, Vikram, "Mining top-k high-utility itemsets from a data stream under sliding window model," *Applied Intelligence*, 2017.
[22] Jayakrushna Sahoo, Ashok Kumar Das, and A. Goswami, "An efficient fast algorithm for discovering closed+ high utility itemsets," *Applied Intelligence*, vol. 45, no.1, pp. 44-74, July 2016.
[23] P. Fournier-Viger, C.-W. Wu, S. Zida and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," *In Proc. 21st*
[24] Fournier-Viger and Philippe, "Efficient Incremental High Utility Itemset Mining", *In Proceedings of the ASE BigData & SocialInformatics*, pp. 53, 2015.

Authors' Profiles

Amaranatha Reddy.P is currently working as Lecturer in Computer Applications in Government Degree College, Nandikotkur, Nandyal district, AP. Presently pursing Ph.D in JNTU kakinada. He completed M.Tech in 2012 from JNTU Hyderabad, B.Tech in 2010 from JNTU Ananthapur.

Dr. MHM Krishna Prasad is currently a Professor in the Department of Computer Science and Engineering, University College of Engineering, Kakinada (Autonomous), JNTUK, Andhra Pradesh. He did his B.E. from Osmania University, Hyderabad, M.Tech. and Ph.D. Computer Science and Engineering from JNTU, Hyderabad. He successfully completed a two year MIUR fellowship at University of Udine, Udine, Italy. He has about 50+ research papers in various International Journals and Conferences, and attended many national and international conferences in India and abroad. He is a member of Association for Computing Machinery (ACM), ISTE and IAENG (Germany) is an active member of the board of reviewers in various International Journals and Conferences. His research interests include data mining, Big Data Analytics and High Performance Computing.

How to cite this paper: P. Amaranatha Reddy, MHM Krishna Prasad, " Sliding Window Based High Utility Item-Sets Mining over Data Stream Using Extended Global Utility Item-Sets Tree", International Journal of Image, Graphics and Signal Processing(IJIGSP), Vol.14, No.5, pp. 72-83, 2022. DOI:10.5815/ijigsp.2022.05.06