Mining Frequent utility sequential patterns in Progressive Databases

K.M.V. Madan Kumar[1*], B. Srinivasa Rao [2]

[1]Research scholar, ANU, Guntur & Department of IT, TKRCollege of Engineering &Technology, Hyderabad.
[2]Research Guide, Acharya Nagarjuna University, Guntur.
**Corresponding author*

ABSTRACT

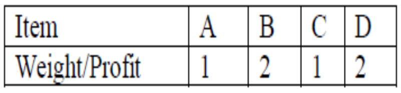
Sequential pattern mining is one of the most important aspects of data mining world and has a significant role in many applications like market analysis, biomedical analysis, weather forecasting etc. in the category of mining sequential patterns the usage of progressive database as an input database is relatively new and has a wide impact in decision-making system. In progressive sequential pattern, we discover the frequent sequences progressively with the help of period of Interest. As the traditional approaches of frequency based framework are not much more informative for decision making, in recent effort utility framework has been incorporated instead of frequency. This addressed many typical business concerns such as profit value associated with each pattern. In this paper, we applied the concept of frequent utility over the progressive database and discovered the sequential pattern efficiently. To do so we proposed an algorithm called U-DirApp which works progressively with the help of a quantitative progressive database. We conducted substantial experiments on the proposed algorithm and proved that this process performs well.

Keywords: Frequent Utility Sequences, Sequential Pattern Mining, Progressive Sequential Pattern, and Period of interest.

1. INTRODUCTION

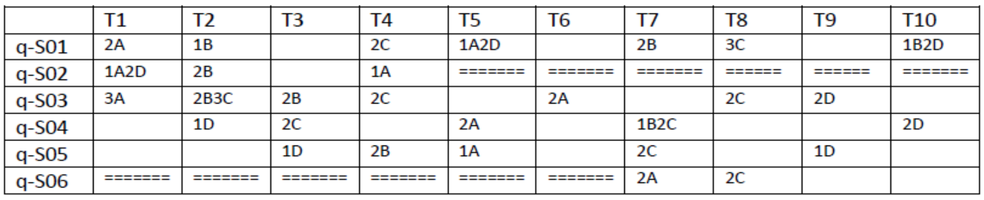
Mining sequential patterns is an important issue in the community of data mining. In time ordered-based critical event scenario, sequential pattern mining has a significant role. Such as mining weblogs, consumer behavior in retail business and gene analysis in bioinformatics. For example mining, sequential patterns are widely used in the retail business to predict the buying patterns of the customers for stock maintenance. In traditional sequential pattern mining, the sequence patterns will be selected based on the frequency/support framework and treated as significant. The pattern which has more or equivalent frequency/support than the user-defined minimum support will be treated as a frequent sequence. Different algorithms already proposed previously [10, 14, 5] for sequential pattern mining has used the concept of downward closure property popularly known as Appriori Property [1].

Most of the sequential patterns which are found as frequent by traditional algorithms are not much more informative for decision making as they do not consider the business value and impact. Because of low frequency, some truly interesting patterns may not be considered as frequent in some scenarios such as fraud detection. For example in the home appliance business selling a LED TV gives us more profit than selling a small induction stove. But the frequency of purchasing a TV in smaller than the induction stove. It leads us to be filtered the TV from frequent items as the frequency is much low. In another case, the transfer of a large amount of money to an unauthorized overseas account may occur very rarely (once in thousands) but have a larger impact on the fraud detection. The traditional support/frequency framework will not handle properly the above-mentioned case.



Item	A	B	C	D
Weight/Profit	1	2	1	2

Fig 1(a) Quality Table



	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
q-S01	2A	1B		2C	1A2D		2B	3C		1B2D
q-S02	1A2D	2B		1A	=======	=======	=======	======	======	=======
q-S03	3A	2B3C	2B	2C		2A		2C	2D	
q-S04		1D	2C		2A		1B2C			2D
q-S05			1D	2B	1A		2C		1D	
q-S06	=======	=======	=======	=======	=======	=======	2A	2C		

Fig 1(b) Quantitative Progressive Sequential Data Base

To tackle the above-said issue, frequent utility sequential pattern mining was introduced [13] where we use utility for finding out interesting patterns instead of frequency. Now let us discuss this frequent utility sequential pattern mining with some example. In the online retail store, the items and their profits (quality) were shown in Fig 1(a). The Fig 1(b) shows the different shopping q-sequences with their quantities: each q-sequence was formed by different q-transactions by customers and each q-transaction consisting of one or more elements (items) associated with their quantity. For example, the q-S01 in Fig 1(b), (2A)(1B)(2C) (1A2D) (2B) (3C) (1B2D) consists of seven itemsets with their quantity. E.g. the quantity of A is 1. The Fig 1(b) above represents this itemsets which are purchased by the customers at different timestamps. Here the timestamps are depending upon the behavior of the database what we have taken. It may range from one second, one minute, and one day to any duration soon and so forth. As per the concept of frequent utility pattern, the utility of the itemset will be the total profit (i.e. it's purchased quantity times its profit). And we can define the utility of an itemset is the sum of the utilities of all its items. Generally, the utility of a sequence will have different values because each item will contain different utility values in a sequence. For example the utility of <AD> in sequence 1 will have two different values i.e. (2*1 + 2*2) and (1*1+ 2*2). But we will consider the highest value i.e. (2*1 + 2*2) = (2+4) =6 instead of 5. So the utility of <AD> in the database is { {6,5}+{}+{7}+{6}+{3}+{}}={6+7+6+3}=22.

The frequent utility sequential pattern mining is very much different than the regular sequential pattern mining process and frequent utility itemset mining. In,frequent utility concepts downward closer property no longer holds because the subset of a frequent itemset may become infrequent and vice versa. So we cannot apply the traditional pruning strategies on directly on this. To solve this problem Junfu.Yin.et.al [16] proposed an algorithm USpan where they are Lexicographic Q- Sequence Tree. For mining sequential pattern mining in progressive databases Huary.et.al [15] proposed an algorithm DirApp where they used timestamp process and appended the item from one time stamp to another in a flat structure with respect to the period of Interest. For omitting the items which are obsolete. Here we have merged the concept of POI and frequent utility and extended DirApp[15] to U-DirApp which performed well. The process will be as follows.

Step 1:- By taking the concepts of quantity and quantity into consideration, we built the utility based sequences, to define the problem.

Step 2:- A complete U-DirApp table was constructed by appending the items or item sets from one time stamp to another time stamp in a flat structure along with this timestamp and utilities.

Step 3:- With the use of U-DirApp table we found out the sequences whole utility is more than user defined utility (ε).

Step 4:- which constructing U-DirApp table, we used the concept of a period of Interest to eliminate the items with this utility; whose time stamp is beyond the POI as an obsolete item.

The experiments show that the proposed U-DirApp algorithm works efficiently for finding the frequent utility sequential patterns in progressive databases. Section 2 talks about the previous work happened on frequent utility and progressive databases. The problem of mining sequential patterns in the progressive database was defined in Section 3. Section 4 deals with Implementation part of U-DirApp. Experimental results were discussed in section 5. Section 6 concludes the work.

2. LITERATURE SURVEY

2.1 Utility Based Itemset Mining

Utility itemset Mining is also known as frequent utility pattern mining was introduced in [13]. In this, each and every item was linked with two components known as an internal utility (i.e. quantity) and external utility (i.e. Quality or profit). In,frequent utility itemset mining, we will mine frequent utility itemsets or patterns whose utility is greater than the user specified minimum utility. In utility based pattern mining, the downward closure property will not hold. So the process of mining frequent utility itemset mining is much more complicated than the traditional itemset mining.

To remedial this problem, in 2004 a systematic approach called as UMining was proposed by some authors. But UMining could not mine the complete set of frequent utility patterns. A two-phase algorithm was proposed [8] which works on the principle of transaction weighted downward closure property which works faster than UMining. With an incremental procedure, IHUP [3] was proposed for frequent utility itemset mining. This IHUP [3] was faster than [8] as it omits multiple scanning of the database. Later with the help of UP-Tree, UP-Growth was proposed and it was much efficient than IHUP because it can reduce the number of interesting patterns, whom could not be pruned by IHUP.

2.2 Mining Sequential Patterns by Utility Framework

Sequential Pattern was first discovered by R.Agrawal.et.al[1] and later it became a popular topic with a quite a good number of researches such as SPAM[5], SPADE[14] and Pre-fix Span[10] which works on the basis on frequency or support mechanism. This traditional approach of support based mining process often resulting in a number of patterns, under which some of them may not be so interested in a business point of view. Here we may filter some sequences which may have less support than the user-defined minimum support, which may lead us more profits [6].

The Considerations of utility in sequential pattern mining will solve the problem and the research started very recently. For frequent utility, sequential patterns in mobile applications were addressed in UMSP [11]. Here location identified was linked with each itemset in a sequence, so the utility of a mobile sequential pattern will be considered a single value in UMSP [11] and UMSP will traverse MTS- tree to mine the patterns. However there are some constraints, due to that, it can handle very limited and specific sequences.

In weblog sequences with utility framework, an algorithm was specifically proposed in [2]. Two tree structures UWAS tree and IUWAS-Tree were traversed in this algorithm. Generally, pattern may contain different utility values, and here the algorithm considers the maximum value of utility with the help of UWAS and IUWAS- trees to specify the utility of a pattern. But this algorithm will be suitable only for complex sequences because this cannot be supported for the sequences having elements with multiple items.

The traditional sequential patterns mining process was extended for utility sequential patterns in [4] by UI and US. In this UI and US the Problem definition was so specific and not generalized for frequent utility sequence analysis. To remedy the problem in [16] the authors proposed USPAN algorithm where they used Lexicographic Q- Sequence tree for construction of utility based sequences and two pruning strategies for finding out frequent utility sequential patterns(width pruning and depth pruning).

In [15] Huang.et.al proposed DirApp algorithm for finding out frequent sequences in progressive databases. Where they used the concepts of Period of Interest and time stamps. The Period of Interest (POI) is a sliding window protocol and advances progressively along with the time stamp. It progressively updates each sequence in a flat structure and accumulates the frequencies of candidate sequential patterns from one POI to another.

3. PROBLEM DEFINITION

Definition 1

(Q-itemset Containing) Given two q-itemsets l_a = [(i_{a_1}, q_{a_1}) (i_{a_2}, q_{a_2}) ….. (i_{a_n}, q_{a_n})]

l_b = [(i_{b_1}, q_{b_1}) (i_{b_2}, q_{b_2}) ….. (i_{b_m}, q_{b_m})],l_b contains $l_a$iff there exist integers $1 \leq j_1 \leq j_2 \leq \ldots \leq j_n \leq m$ such that $i_{a_k} = i_{bj_k} \wedge q_{a_k} = q_{bj_k}$ for $1 \leq k \leq n$, denoted as $l_a \subseteq l_b$.

Definition 2

(Q-sequence Containing) Given two q-sequences s $=\langle l_1, l_2, \ldots\ldots l_n\rangle$ and s’ $=\langle l'_1, l'_2, \ldots\ldots l'_n\rangle$, wesay s' contains s or s is a q-subsequence of s' if there exist integers $1 \leq j_1 \leq j_2 \leq \ldots \leq j_n \leq n'$ such that $l_k \subseteq l'_{jk}$ for $1 \leq k \leq n$, denoted as s ⊆s'.

Definition 3

(Length and Size) A (q-) sequence is called k (q-) sequence i.e. its length is k if there are k-(q) items in the (q-)sequence; the size of a (q-)sequence is the number of (q-)itemsets in the (q-)sequence.

Definition 4

(Matching) Given a q-sequence s = $\langle(q_1\, s_{1,})(q_2\, s_{2,})\ldots.(q_n\, s_n)\rangle$ and a sequence t $=\langle t_1 t_2 \ldots. t_m\rangle$. s matches $t$if$n = m$ and $s_{k,}$= t_k for $1 \leq k \leq n$, denoted as t ~ s.

Definition 5

(Q-item Utility) The *q-item utility* is the utility of a single q-item $(q\ i)$ denoted and defined as $u(q\ i)$:

$u(q\ i) = f_{u_i}(p(i), q)$ (1)

Where $f_{u_i}$is the function for calculating q-item utility.

Definition 6

(Q-itemset Utility) Q-itemset utility is the utility of a q-itemset$l = [(q_1\, i_1)(q_2\, i_2)(q_n\, i_n)]$ denoted and defined as

$$u(l):(l) = f_{u_{is}}\left(\bigcup_{j=1}^{n} u(q_j\ i_j)\right) \qquad (2)$$

$f_{u_{is}}$is the function for calculating q-itemset utility

Definition 7

(Q-sequence Utility) For a q-sequence s $=\langle l_1 l_2 \ldots\ldots l_m\rangle$, the q-sequence utility is $u(s)$:

$$u(s) = f_{u_s}\left(\bigcup_{j=1}^{m} u(l_j)\right) \qquad (3)$$

Where f_{u_s} is the utility function for q-sequences.

Definition 8

(Q-sequence Database Utility) For a utility oriented sequence database $s = \{\langle sid_1, s_1\rangle, \langle sid_2, s_2\rangle, \ldots., \langle sid_r, s_r\rangle\}$,the q-sequence database utility is $u(s)$:

$$u(s) = f_{u_{db}}\left(\bigcup_{j=1}^{r} u(s_j)\right) \qquad (4)$$

$f_{u_{db}}$is the function for aggregating utilities in the database.

In the above, utility functions $f_{u_i}, f_{u_{is}}, f_{u_s}$and $f_{u_{db}}$are all application-dependent, which may be determined through collaboration with domain experts.

Definition 9

(Sequence Utility) Given a utility-oriented database S and a sequence $t = \langle t_1, t_2, \ldots t_n\rangle$, t's utility in q-sequence $s = \langle l_1, l_2, \ldots l_m\rangle$, from s is denoted and defined as $v(t, s)$,which a utility is set:

$$v(t, s) = \bigcup_{s' \sim t \wedge s' \subseteq s} u(s') \qquad (5)$$

The utility of t in s is denoted and defined as $v(t)$,which al so a utility is set:

$$v(t) = \bigcup_{s \in S} u(t, s) \qquad (6)$$

Definition 10

Given an utility oriented database *S* and a sequence *β*, we call *β* is frequent utility sequence in *S* or *β* is a utility sequential pattern in *S* if *utility* (*β*) ≥ user defined utility (*s).*

Definition 11

Progressive utility sequential pattern mining problem.“Given a user-specified length of POI and a user-defined minimum support threshold, find the complete set of frequent subsequences whose occurrence utility are greater than or equal to the minimum utility times the number of sequences in the recent POI of a progressive utility database.”

4. IMPLEMENTATION

4.1. Utility Candidate set generation by U-DirApp

The main objective of the U-DirApp has to bring up the all the q- sequences in a flat structure and also assemble the candidate utility sequential patterns along with their corresponding utility from one POI to another. It also keeps a utility candidate set for each and individual q-sequence in the given database to store all utility sequential patterns of the candidate with their utility. U- DirApp generates all combinations of items along with their utility as a utility candidate elements to support arriving elements. To be given an example, Assuming that the arriving element consists of(X, Y, Z), the different possible combinations of candidate utility elements are as follows X, Y, Z,(X, Y),(X, Z),(Y, Z), and (X, Y, Z). If that is the case, U-DirApp identifies the utility candidate set of the q- sequence based on the sequence ID of the elements. In order to create the new utility candidate sequential patterns with matching begin timestamps, U-DirApp adds each combination of the components in the coming data to the already existed utility candidate set. When the utility candidate sequential patterns come into sight as a utility sequence of a progressive database the begin timestamp remains timestamp. Then, U-DirApp places the new utility candidate sequential patterns into the utility candidate set. Apart from it, the U-DirApp also gathers the utility values of all utility candidate sequential patterns in another candidate set. U-DirApp eliminates the utility candidate sequential patterns along with their utility values as an obsolete, whose begin timestamps are smaller than the current time minus POI, from utility candidate set. Eventually, U-DirApp produces frequent utility sequential patterns and omits out obsolete utility candidate sequential patterns for all q- sequences.

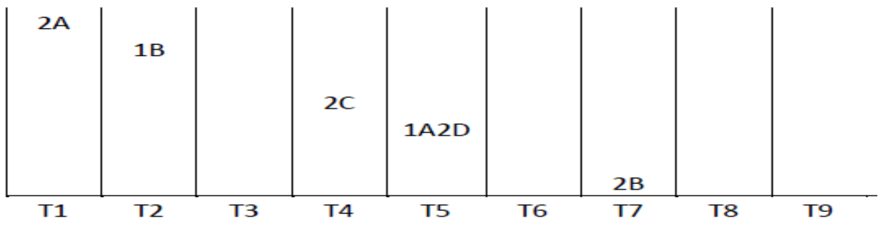
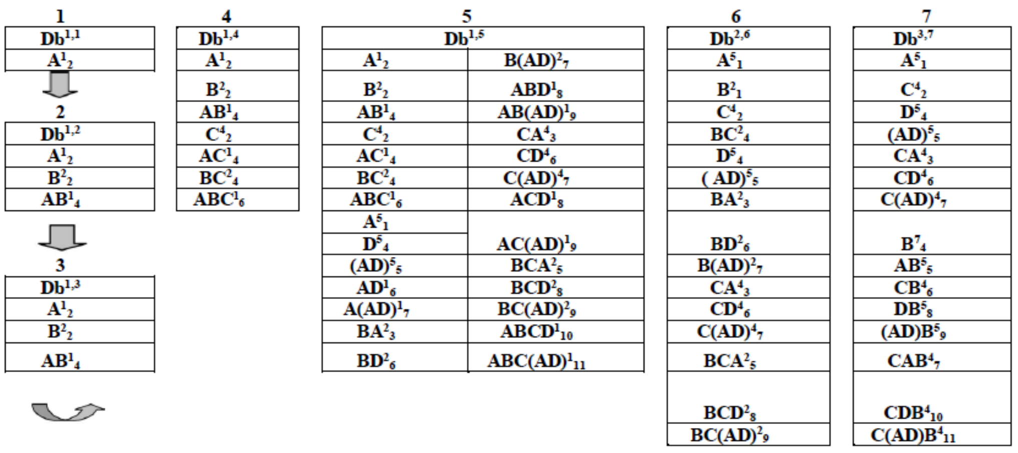


Fig 2 (a) Items with qualtity that Q-S01 has.



1
$Db^{1,1}$
A^1_2

2
$Db^{1,2}$
A^1_2
B^2_2
AB^1_4

3
$Db^{1,3}$
A^1_2
B^2_2
AB^1_4

4
$Db^{1,4}$
A^1_2
B^2_2
AB^1_4
C^4_2
AC^1_4
BC^2_4
ABC^1_6

5	
$Db^{1,5}$	
A^1_2	$B(AD)^2_7$
B^2_2	ABD^1_8
AB^1_4	$AB(AD)^1_9$
C^4_2	CA^4_3
AC^1_4	CD^4_6
BC^2_4	$C(AD)^4_7$
ABC^1_6	ACD^1_8
A^5_1	
D^5_4	$AC(AD)^1_9$
$(AD)^5_5$	BCA^2_5
AD^1_6	BCD^2_8
$A(AD)^1_7$	$BC(AD)^2_9$
BA^2_3	$ABCD^1_{10}$
BD^2_6	$ABC(AD)^1_{11}$

6
$Db^{2,6}$
A^5_1
B^2_1
C^4_2
BC^2_4
D^5_4
$(AD)^5_5$
BA^2_3
BD^2_6
$B(AD)^2_7$
CA^4_3
CD^4_6
$C(AD)^4_7$
BCA^2_5
BCD^2_8
$BC(AD)^2_9$

7
$Db^{3,7}$
A^5_1
C^4_2
D^5_4
$(AD)^5_5$
CA^4_3
CD^4_6
$C(AD)^4_7$
B^7_4
AB^5_5
CB^4_6
DB^5_8
$(AD)B^5_9$
CAB^4_7
CDB^4_{10}
$C(AD)B^4_{11}$

Fig 2(b) Utility candidate set for Q-S01

Example 1:-The input database is the similar as reflected in Fig1.The elements formed by q-sequence q-S01 with their utility values from the corresponding existing POI are shown in Fig 2(a). It is also noticed that candidate utility sequential pattern for q-S01 is maintained by U-DirApp is shown in Fig 2(b). The $DB^{p,q}$ at the top of each table explains the database containing elements from timestamp p to q. U-DirApp stores the element A along with its corresponding utility at timestamp 1 and stores A as a utility candidate with begin timestamp 1 and utility (Quality*Quantity=1*2), i.e. A^1_2, in the utility candidate set. While getting the element B along with its utility at timestamp 2 to the utility candidate set, U-DirApp joins with the existing element in the candidate set. U-DirApp joins B^2_2 (1*2) to A^1_2 to be $AB^1_{4\,(2+2)}$. In addition, U-DirApp stores B^2 at the same time along with its utility value as prefix i.e. B^2_2. The second table in Fig 2(b) shows this process. The shadowed (and underlined at the same time) candidate patterns represent newly generated candidates. The same process continues until timestamp 4.When U- DirApp assigns $Db^{1,5}$ at timestamp 5, the outcome element is (AD). All combinations of this element along corresponding utility values are A^5_1,D^5_4 and $(AD)^5_5$. Even though there is existing A^1 in the utility candidate set, U-DirApp does not change its begin timestamp from 1 to 5 because the utility value of A in timestamp 1 and 5 may be different.Now U-DirApp joins all the elements especially whose timestamp between 2 & 4 to A^1along with their utility values into the q-candidate set. Thus we got BA^2_3, CA^4_3 , and $BCA^2{}_5$into q-candidate set as shown in the fifth table of Fig 2(b). The same process is applied to the elements and (AD).So ,D^5_4, $(AD)^5_5$, AD^1_6, $A(AD)^1_7$, BD^2_6, $B(AD)^2_7$, ABD^1_8, $AB(AD)^1_9$, CD^4_6, $C(AD)^4_7$, ACD^1_8, $AC(AD)^1_9$, BCD^2_8, $BC(AD)^2_9$, $ABCD^1_{10}$ and $ABC(AD)^1_{11}$ are inserted into the q- candidate set. After fifth time stamp, the POI goes to time interval [2, 6]. As reflected in the sixth table in Fig (2b) the q-candidate patterns which have the begin timestamp less than 2 should be deleted from the q-candidate set. When U-DirApp takes $DB^{3,7}$ the candidate patterns whose begin timestamp is less than 3 can also be deleted too.

S01
$Db^{1,2}$
A^1_2
B^2_2
AB^1_4

(1)

S02
$Db^{1,2}$
A^1_1
D^1_4
$(AD)^1_5$
B^2_4
AB^1_5
DB^1_8
$(AD)B^1_9$

(2)

S03
$Db^{1,2}$
A^1_3
B^2_4
C^2_3
$(BC)^2_7$
AB^1_7
AC^1_6
$A(BC)^1_{10}$

(3)

S04
$Db^{1,2}$
D^2_2

(4)

$DB^{1,2}$ (4)
AB^1_{16}
$A(BC)^1_{10}$
AC^1_6
$AD(B)^1_9$
DB^1_8

(5)

Fig 2(c) Utility candidate set for all Q-Sequences in $Db^{1,2}$

4.2 Frequent Utility Sequential Pattern Generation

Now in this section, we will discuss how the frequent utility sequential patterns will be generated from the utility candidate sequential patterns. Here we should construct the utility candidate sequential patterns for all the sequences (S01 to S06) shown in Fig1. The process of construction of utility candidate sequential patterns for the first sequence was discussed previously in section 4.1. The same procedure will be adapted for all the sequences mentioned in Fig 1. The next process is identifying the frequent utility sequential patterns with respect to the time stamps say for example $Db^{1,2}$, $Db^{1,3}$, $Db^{1,4}$, $Db^{1,5}$, $Db^{2,6}$, $Db^{3,7 for}$ all the sequences and accumulate the total utility of the sequences accordingly.

Algorithm UTI (Item, pi, qi)

```
1. var UTI;
2. var $I_1$…..$I_n$;// Items
3. For (all combinations of *items* in the *ele*);
4. If (*No of items*>1);
5. Find equivalence classes for all 1- q sequences;
6. for each q-[S]
7. UTI (*ele*) =Max ( $\sum_{i=1}^{n} pi * qi$);
8. **else**
9. UTI (*ele*) = Max ($pi * qi$)
END
```

Algorithm U-DirApp (Min$UTI(\varepsilon)$, POI)

```
1. var *PT; //* Progressive Table
2. var *current Time*; // timestamp now
3. var *eleSet* ; // used to store elements *ele*
4. **while** (there is still new transaction);
5. *eleSet* = read all *ele* at *current Time*;
6. **ck** (*currentTime, PT*);
7. *current Time++ ;*
 END
```

Algorithm ck (*current time, PT*)

```
1. for (each ck of PT in post order);
2. if (ck is null);
3. for (ele of every seq in eleSet);
4. for (all combination of elements in the ele);
5. if (element==label of one of ele.ck);
6. if (seq is in subck.seq_list);
7. Update timestamp of seq to currentTime, UTI (ele);
8. else
9. create a new sequence with current Time, UTI(ele);
10. else
11.create a new subck with element, current Time,UTI(ele);
12. else for (every seq in the seq_list);
13.if (seq.timestamp<= currentTime_POI);
14. delete seq, UTI (el) from seq_list and continue to next seq;
15.if (there is new ele of the seq in eleSet);
16. for (all combination of elements in the ele);
18. if (element is not in ck);
19. if(element==label of one of subck);
20. if (seq is in subck.seq_list);
21. Subck.seq_list. Seq.timestamp= seq.timestamp;
22. else
23. create a new sequence withseq. timestamp,UTI (ele);
24. else
25. Create a new subck with element, seq.timestamp, UTI (ele);
28. if (seq_list.Total UTI (el)>=ε);
29. Output the ele as a sequential pattern;
END
```

Fig 3 (a) (b) (c)

5. EXPERIMENT RESULTS

In this section the execution time of proposed algorithms has been recorded at each time stamp and then the accumulated time from the first time stamp to existing time stamp was shown as cumulative execution time. As shown in fig 8 (a),even the algorithm DirApp shows best performance in terms of cumulative execution time over U-DirApp, the proposed algorithm mines the frequent sequential patterns based on their utilities. Utility based sequential patterns gives us more information then sequential patterns in decision making.

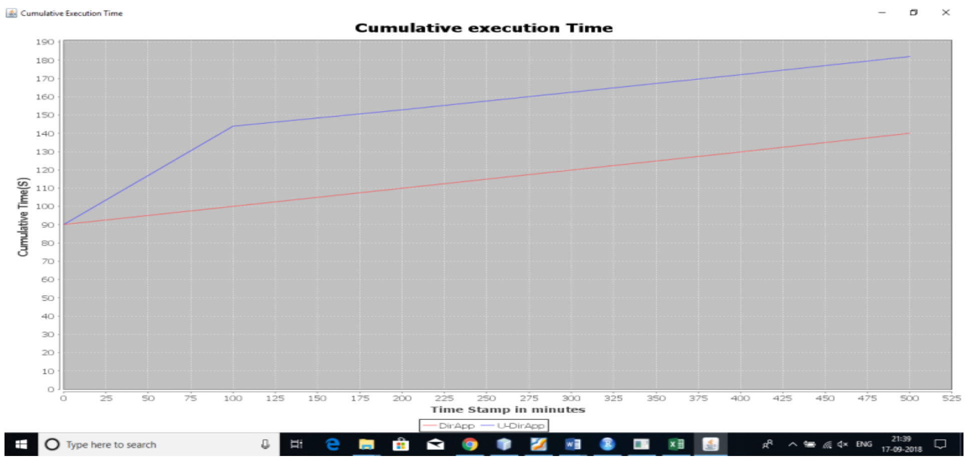


Fig 4(a) Cumulative time Vs Time Stamps

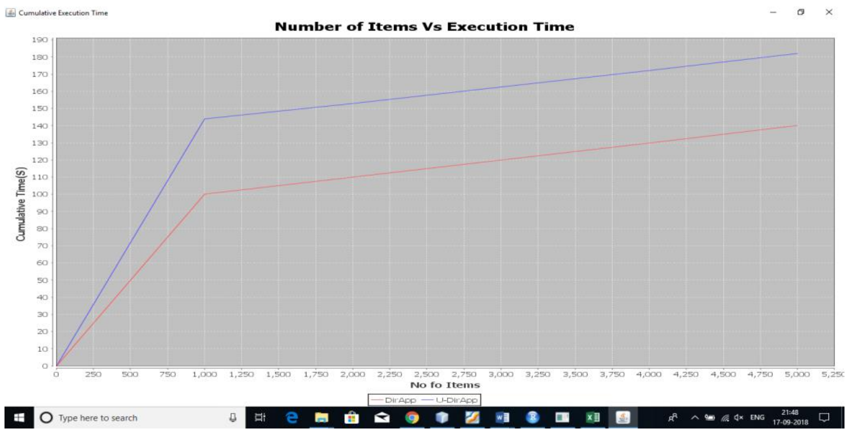


Fig 4(b) Execution time Vs No of Items

In this section we considered the different number of items as input parameters and examined the effects on the proposed algorithms. The figure 4 (b) shows the total execution time of both DirApp and U-DirApp. By the study conducted, we understood that the time taken increases as the number of items increases because the number of candidate frequent sequential patterns stored by proposed algorithms will be bigger as the number of items increases.

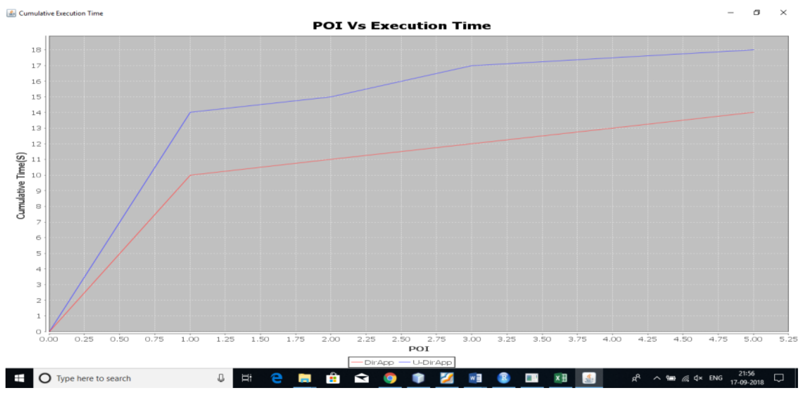


Fig 4(c) Execution time Vs POI.

Here we have taken length of POI as input parameters and recorded the execution time for the proposed algorithms. The figure 4(c) shows the total execution time of both DirApp and U-DirApp with respect to length of POI . By the study conducted, we understood that the time taken increases as the length of POI increases because the number of candidate frequent sequential patterns stored by proposed algorithms will be bigger as length of POI increases.

7. REFERENCES

1. R. Agarwal and R. Srikanth, Mining sequential patterns, ICDE 1995, pp.3-14.
2. C.F.Ahmed, S.K. Tanbeer and B.Jeong, Mining High Utility Web Access Sequences in Dynamic Web Log Data, SNPD 2010, pp.76-81.
3. C.F. Ahmed, S.K. Tanbeer, J. Byeong-Soo and L.Young-Koo, Efficient tree structures for high utility pattern mining in incremental databases, TKDE 2009, vol.21, pp.1708-1721.
4. C.F. Ahmed, S.K. Tanbeer and B. Jeong, A Novel Approach for Mining High-Utility Sequential patterns in Sequential Databases, ETRI Journal ,2010, vol.32,no.5, pp.676-686.
5. J.Ayres, J. Flannick, J. Gehrke and T. Yiu, Sequential PAttern mining using a bitmap representation , ICDM 2002, pp.429-435.
6. L.Cao, P.Yu, C.Zhang and Y.Zhao. Domain Driven Data Mining. Springer 2010.
7. Y. Li, J. Yeh and C. Chang, Isolated items discarding strategy for discovering high utility itemsets, Data and knowledge Engineering, Vol.64, Issue 1, pp.198-217, Jan., 2008.
8. Y.Liu, W. Liao and A. choudhary, A two-phase algorithm for fast discovery of high utility itemsets, PAKKD 2005, vol.3518, pp.689-695.
9. N.R Mabroukeh and C.I.Ezeife, A taxonomy of sequential pattern mining algorithms, ACM Comput. Surv., 2010, vol. 43, pp.1-41.
10. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q.Chen, U.Dayal and M.C. Hsu., PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, ICDE 2001, pp. 215-224.
11. B. Shie, H.Hsiao, V.S.Tseng and P.S.Yu, Mining high utility mobile sequential patterns in mobile commerce environments, DASFAA 2011, pp.224-238.
12. V.S. Tseng, C.-W. Wu, B.-E. Shie and P.S. Yu, UP-Growth: an efficient algorithm for high utility itemset mining, KDD 2010, pp.253-262.
13. H. Yao, H. J. Hamilton and C.J.Butz, A foundational approach to mining itemset utilities from databases, ICDM 2004, pp.31-60.
14. M.J.Zaki, SPADE: An Efficient Algorithm for Mining Frequent Sequences, Machine Learning, 2001, vol.42, pp.31-60.
15. Jen-Wei Huang, Chi-Yao Tseng, Jian- Chih Ou, and Ming-Syan Chen, A General Model for Sequential Pattern Mining with a Progressive Database, IEEE Transactions On Knowledge And Data Engineering , Vol. 20, No. 9, September 2008.pp. 1153-1167.
16. Junfu Yin, Zhigang Zheng and Longbing Cao, USpan: An Efficient Algorithm for Mining High Utility Sequential Patterns.KDD 12,