International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 2, Issue 4, April 2015

Efficient Algorithm For Mining High Utility Item Sets From Large Datasets Using Vertical Approach

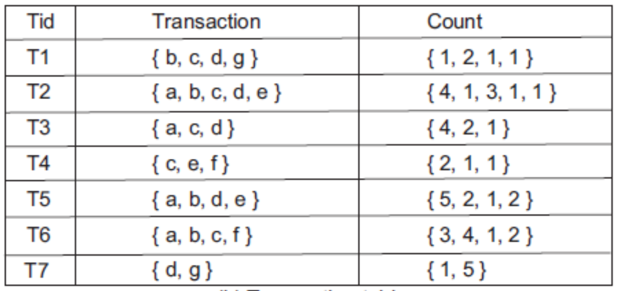
[1]Dr.S.G.Sanjeevi, [2] Aluguvelli Sindhu, [3] Shrutika Pimpalkar, [4] Sujith Srivardhan Arram
[1] [2] [3] [4]National Institute of Technology, Warangal
[1] sgs@nitw.ac.in, [2] sindhureddy216@gmail.com, [3] shrutika.nitw@gmail.com, [4], suji.srivardhan@gmail.com

Abstract- **High Utility Item set Mining is a challenging task as the Downward Closure Property present in frequent item set mining does not hold here. In recent times many algorithms have been proposed for mining high utility item set s ,but most of them follow a two-phase horizontal approach in which candidate item set s are generated first and then the actual high utility item set s are mined by performing another database scan. This approach generates a large number of candidate item set s which are not actual high utility item set s thus causing memory and time overhead to process them. To overcome this problem we propose a single phase algorithm which uses vertical database approach. Exhaustive search can mine all the high utility item set s but it is expensive and time consuming. Two strategies based on u-list structure and item pair co-existence map are used in this algorithm for efficiently pruning the search space to avoid exhaustive search. Experimental analysis over various databases show that the proposed algorithm outperforms the two-phase algorithms UP-Growth and other two phase algorithms in terms of running times and memory consumption.**

Index Terms — **high utility item set s, min_util, u-list, item pair coexistence map.**

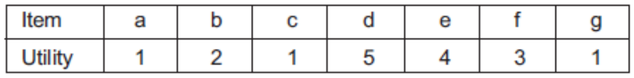
I. INTRODUCTION

Recent advances in database facilities led to the increased use of databases by many organizations leading to storage of large data. Extraction of knowledge and information from this data is a developing area of research . Frequent item set mining is identifying set if items whose count in the transaction database is greater than a predefined minimum value. Frequent item set mining is identifying set of items whose count in the transaction database is greater than a predefined minimum value. Frequent item set mining follows *downward closure property*. According to this property if an item set is infrequent then all the supersets of that item set are also infrequent so it is not required to check the supersets of the infrequent item set s thus preventing checking all the item set s exhaustively. But frequent item set mining doesn't take into account the profit/utility of each item and the importance of each item in a transaction. So the high utility item set mining is used to discover item set s with utility greater than a minimum threshold value. But the downward closure property which is used for pruning infrequent item set s does not hold in high utility item set mining. So mining high utility item set s is a complex task. Most of the existing high utility item set mining algorithms follow a two-phase approach in which the candidate item set s are found first and the actual high utility item set s among the candidate item set s are then identified in the second phase.In this paper we propose a single phase algorithm for mining high utility item set s using a vertical approach.



Tid	Transaction	Count
T1	{ b, c, d, g }	{ 1, 2, 1, 1 }
T2	{ a, b, c, d, e }	{ 4, 1, 3, 1, 1 }
T3	{ a, c, d }	{ 4, 2, 1 }
T4	{ c, e, f }	{ 2, 1, 1 }
T5	{ a, b, d, e }	{ 5, 2, 1, 2 }
T6	{ a, b, c, f }	{ 3, 4, 1, 2 }
T7	{ d, g }	{ 1, 5 }

Fig. 1. A transaction database

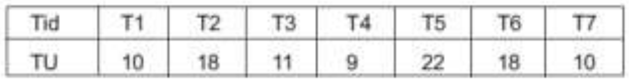


Item	a	b	c	d	e	f	g
Utility	1	2	1	5	4	3	1

Fig. 2. Profit values of each item

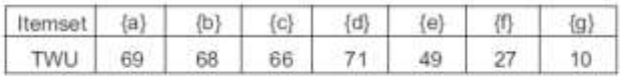
II. DEFINITIONS

Let of I be the set of items, I={i_1, i_2…i_m} and each item has a unit profit pr(i_p), 1<=p<=m. A set of distinct item set s {i_1, i_2…i_k} is called as item set X where i_j□I,1≤j≤k. k is the length of the item set X. An item set whose length is k is called k-item set . A transaction database D={T_1,T_2…T_n} contains set of transactions and each transaction has a unique identifier called as TID [4]. Each item i_p in transaction T_d is associated with a quantity q(i_p,T_d) which is the purchased quantity of the item i_p in T_d [4]. Definition 1: Utility of an item i_p in a transaction T_d is denoted as u(i_p,T_d) and defined as pr(i_p) X q(i_p,T_d). Definition 2: Utility of an item set X in T is defined as U(X,T) = □$_{i\in X □ X\subseteq T}$ u(i,T) [4]. Definition 3: Utility of an item set X in D is denoted as u(X) and defined as u(X) = □$_{X\subseteq T □ T □ D}$ u(X,T) [4]. Definition 4: An item set is called a high utility item set if its utility is no less than a user-specified minimum utility threshold which is denoted as min_util. Otherwise, it is called a low-utility item set [4]. Definition 5: Transaction utility of a transaction Td is denoted as TU(T_d) and defined as u(T_d,T_d)[4]. Definition 6: Transaction-weighted utility of an item set X is the sum of the transaction utilities of all the transactions containing X, which is denoted as TWU(X) and defined as TWU(X)= □$_{X\subseteq T □ T □ D}$TU(T) [4]. Definition 7: An item set X is called a high-transaction weighted utility item set (HTWUI) if TWU(X) is no less than min_util [4]. Property 1 : The Transaction-weighted utility of an item set follows the downward closure property that is if the item set X is not a high utility item set then any of the superset of X is not a high utility item set [4]



Tid	T1	T2	T3	T4	T5	T6	T7
TU	10	18	11	9	22	18	10

Fig. 3. Transaction utility values



Itemset	{a}	{b}	{c}	{d}	{e}	{f}	{g}
TWU	69	68	66	71	49	27	10

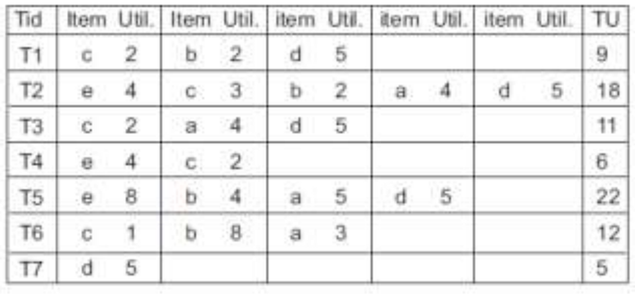
Fig. 4.Transaction weighted utility values Problem statement : Mining high utility item set s from a transaction database D given a user specified minimum utility threshold min_util is finding all the item set s whose utility is greater than min_util.

III. EXISTING APPROACH

An existing efficient algorithm for mining high utility item set s is UP-Growth. It uses a compact data structure called UP-tree which is constructed by scanning the database twice. Potential high utility item set s with overestimated utilities are generated from the UP-tree by applying the UP-Growth algorithm. After finding the potential high utility item set s another database scan is performed to find actual high utility item set s among potential high utility item set s. Drawbacks: This approach generates a large number of candidates but most of these may not be high utility item set s because of overestimated utilities. It results in large memory and time overhead in storing and processing these candidate item set s.

IV. METHODOLOGY

To overcome the problems faced by existing two-phase algorithms, we propose a single phase algorithm which discovers all high utility item set s using two pruning strategies based on u-lists structure and item pair co-existence map. These pruning strategies are used to efficiently prune the item set s in the search space which is otherwise exponentially high due to all the possible enumerations of items in the database. In the first step, Transaction Weighted Utility of each item and Transaction Utility of each transaction is calculated. The transactions are then reorganized by removing the items with utility less than min_util and by arranging remaining items in the ascending order their Transaction Weighted Utility.



Tid	Item	Util.	Item	Util.	item	Util.	item	Util.	item	Util.	TU
T1	c	2	b	2	d	5					9
T2	e	4	c	3	b	2	a	4	d	5	18
T3	c	2	a	4	d	5					11
T4	e	4	c	2							6
T5	e	8	b	4	a	5	d	5			22
T6	c	1	b	8	a	3					12
T7	d	5									5

Fig. 5. Reorganized transactions

A. Item Pair co-existence map

After finding transaction weighted utilities of individual items, the item pair co-existence map is constructed in which each distinct item pair is mapped to its Transaction Weighted Utility. Definition : Transaction Weighted Utility of an item pair denoted as TWU(x,y) is defined as the sum of transaction utilities of all reorganized transactions in which both x and y are present where x and y are distinct items in the database. TWU(x,y) is calculated as □$_{x,y□ T□ T□ D}$TU(T). The Transaction weighted utilities of all distinct item pairs are calculated and stored in the Item Pair co-existence map (abbreviated as IPCM).

B. U-List structure

Definition: Remaining utility (ru) of an item set X in a reorganized transaction is the sum of utilities of all items after X in the transaction. The set of items after the item set X i.e remaining items after X in a reorganized transaction T is denoted as T/X. Each element in the U-List structure of every item set X consists of 3 fields . TID(Transaction ID), iu(item set utility) and ru(remaining utility) where TID is the transaction id of the transaction in which the item set X is present , iu is the utility and ru is the remaining utility of X in the reorganized transaction with transaction id TID.The U-List for each item is then constructed. First the U-Lists for all the 1-item set s with Transaction weighted utility greater than min_util are constructed. U-Lists for all 1-item set s of the Database shown in Fig. 1 are shown in fig.6.

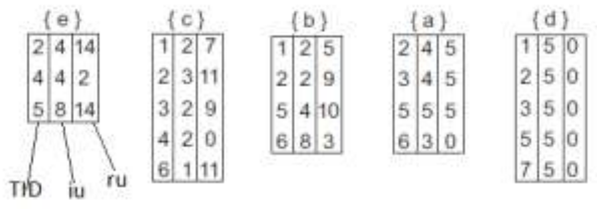


Fig. 6. U-Lists of 1-item set s

Then U-Lists for 2-item set s of the form {pq} are constructed from U-Lists of 1-item set s {p} and {q} by taking the intersection of U-Lists of {p} and {q}.The common TIDs from both U-Lists are identified and the iu of each element in the U-List of 2-item set {pq} is the sum of iu's of the corresponding element in U-Lists of {p} and {q} where as ru of each element in the U-List of 2-item set is the minimum of ru's of the corresponding element in the U-Lists of {p} and{q}.Fig.7shows the U-Lists of 2-item set s.

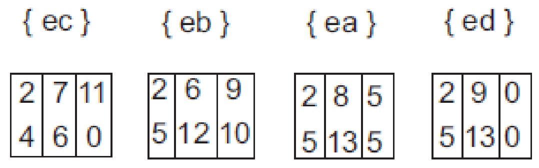


Fig. 7. U-Lists of 2-item set s

The U-Lists of k+1 item set P (i_1,i_2…….. i_k,i_{k+1}) can be constructed by intersecting the U-Lists of two k-item set s P1(i_1,i_2………i_{k-1},i_k) and P2(i_1,i_2………i_{k-1},i_{k+1}) respectively. The subroutine is shown below: Let UL(P) denote the u-list of P and E denote an element in the u-list. Join(P1,P2) Output: u-list of k+1 item set P for each element E_i in UL(P1) and E_j in UL(P2) if Ei.TID == E_j.TID E.TID = E_i.TID E.iu = E_i.iu + E_j.iu – iu(i_1i_2...i_{k-1}) E.ru = Minimum(E_i.ru, E_j.ru) Add E to UL(P) end if end for

Pruning strategies

A. Strategy 1

U-List for k+1 item set s are formed only if sum of its iu's and ru's in the U-List of its corresponding k-item set s is greater than or equal to min_util i.e., if sum
of iu's and ru's in the U-List of an item set A is lesser than min_util, then any extension A' of the item set A cannot be a high utility item set . Proof: For all transactions T such that A' ⊆T Given A' is an extension of A. Let A'-A denote the items present in A' but not in A. As A'⊆T this implies A-A' ⊆ T/A So u(A',T) = u(A,T) +u((A'-A),T) = u(A,T) +□$_{i\epsilon(A'-A)}$u(i,T) ≤ u(A,T) +□$_{i\epsilon(T/A)}$u(i,T) = u(A,T) +ru(A,T) Let id(T) represent the id of transaction T, tids(A) and tids(A') represent the tid set in A's U-List and A' 's U-List respectively . As A⊆A' this implies tids(A) ⊆ tids(A') So, u(A') = □$_{id(T)\epsilon\ tids(A')}$u(A',T) ≤ □$_{id(T)\epsilon\ tids(A')}$ u(A,T) +ru(A,T) ≤ □$_{id(T)\epsilon\ tids(A)}$ u(A,T) +ru(A,T) Utility of an item set A' which is an extension of item set A is less or equal to sum of ru's and iu's in the U-List of A. Therefore if □$_{id(T)\epsilon\ tids(A)}$ u(A,T) +ru(A,T) < min_util then u(A') is less than min_util Hence Proved. For example consider the U-List of the item set {ec} in fig 7.If we consider min_util as 30 ,{ec} should be pruned from being extended because the sum of ru's and iu's is less than min_util.

B. Strategy 2

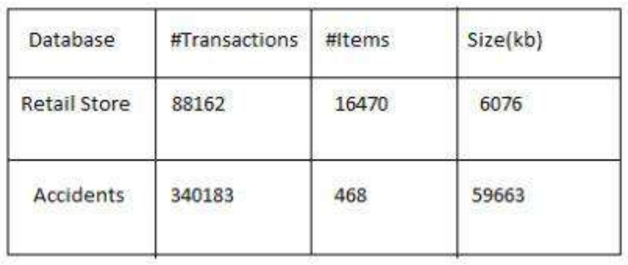
U-List for k+1item set P(i_1i_2…i_k,i_{k+1}) is formed from U-Lists of two k item set s P1(i_1i_2…$i_{k-1}i_k$) and P2(i_1i_2…$i_{k-1}i_{k+1}$) only if TWU(i_k, i_{k+1}) is greater than or equal to min_util. Proof: It is clear that P(i_1i_2…i_{k+1}) is super set of { i_k i_{k+1}} If TWU(i_k,i_{k+1}) <min_util then TWU(P(i_1i_2…i_{k+1})) is also less than min_util according to Property 1 If TWU(P(i_1i_2…i_{k+1}))<min_util then P is not a high utility item set . Therefore item set P can be pruned if TWU(i_ki_{k+1})<min_util. Hence Proved For example consider U-Lists of two item set s {abc} and {abe}. The U-Lists of {abc} and {abe} are joined to form U-List of {abed} only if TWU(c,e) is greater than or equal to min_util.From the reorganized database in fig 5 the TWU(c,e) can be calculated as follows TWU(c,e)= □ TU(<T1,T2,T3,T4,T6> □ <T2,T4,T5>) =TU(T2)+TU(T4) =18+6 =24 As TWU(c,e)<min_util the item set {abce} formed by joining {abc} and {abe} will not be a high utility item set and hence can be pruned before performing the join
.

VI. PROPOSED ALGORITHM

Algorithm: U-Vertical Algorithm Input : B: an item set (initially empty), Ext(B): a set of 1-extensions of B, the min_util threshold, the item pair co-existence map Output: all high utility item set s with B as prefix For each item set B_X □ Ext(B) if sum(UL(B$_X$.iu's)) ≥ min_util print B_X end if if sum(UL(B_X).iu's) + sum(UL(B_X).ru's)≥ min_util then //Strategy 1Ext(B_X)←NULL for each item set B_Y□ Ext(B) such that y□ t(x) /*t(x) Is the set of items with TWU less than TWU(x)*/If TWU(x, y)≥ min_uti //Strategy 2 B_{XY}←B_X U B_Y UL(B_{XY})←Join(B_X, B_Y) Ext(B_X)←Ext(B_X) U B_{XY} End if End for End if U-Vertical(B_X, Ext(B_X), min_util) End

VII. EXPERIMENTAL EVALUATION AND RESULTS

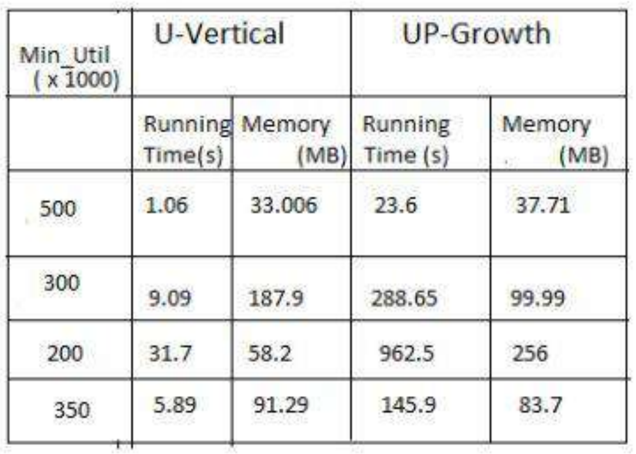
The algorithm presented in the paper had been experimented with real time databases Retail-Store and Accidents database.



Database	#Transactions	#Items	Size(kb)
Retail Store	88162	16470	6076
Accidents	340183	468	59663

Fig 8. Database Details

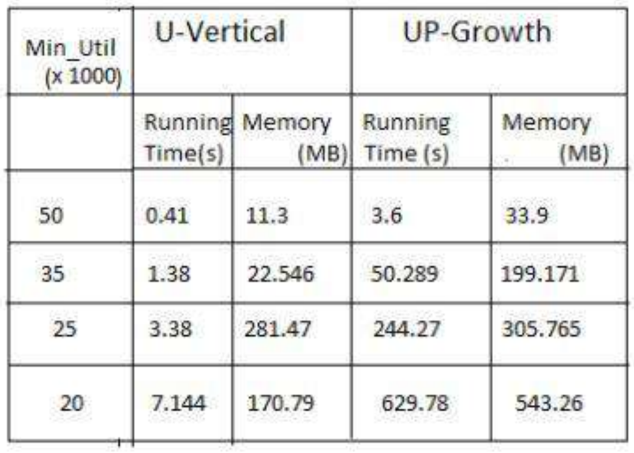
The running time and memory requirement values for various min_util values of retail store database is shown in fig 9.



Min_Util (x 1000)	U-Vertical		UP-Growth	
	Running Time(s)	Memory (MB)	Running Time (s)	Memory (MB)
500	1.06	33.006	23.6	37.71
300	9.09	187.9	288.65	99.99
200	31.7	58.2	962.5	256
350	5.89	91.29	145.9	83.7

Fig 9. Running times and memory requirement for retail-stores database.

The running time and memory requirement values for various min_util values of accidents database is shown in fig 10.



Min_Util (x 1000)	U-Vertical		UP-Growth	
	Running Time(s)	Memory (MB)	Running Time (s)	Memory (MB)
50	0.41	11.3	3.6	33.9
35	1.38	22.546	50.289	199.171
25	3.38	281.47	244.27	305.765
20	7.144	170.79	629.78	543.26

Fig 10.Running times and memory requirements for accidents database.

From the above values of running time and memory requirement it can be observed that the algorithm U-Vertical outperforms UP-Growth in terms of time and memory complexit.

CONCLUSION

In this paper we presented the algorithm for mining high utility item set s which outperforms UP-Growth and other two-phase algorithms. The algorithm proposed in the paper is designed for static databases. It can be further extended to design an efficient algorithm for mining high utility item set s from dynamic databases.

REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proc. of the 20th Int'l Conf. on Very Large Data Bases, pp. 487-499, 1994.

[2] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee. Efficient tree structures for high utility pattern mining in incremental databases. In IEEE Transactions on Knowledge and Data Engineering, Vol. 21, Issue 12, pp. 1708-1721, 2009.

[3] B.-E. Shie, V. S. Tseng, and P. S. Yu. Online mining of temporal maximal utility item set s from data streams. In Proc. of the 25th Annual ACM Symposium on Applied Computing, Switzerland, Mar., 2010..

[4] Vincent S. Tseng1 , Cheng-Wei Wu1 , Bai-En Shie1 , and Philip S. Yu2. UP-Growth: An Efficient Algorithm for High Utility Item set Mining, In IEEE Transactions on Knowledge and Data Engineering , Vol. 25, No. 8, August 2013.