Efficient High Utility Itemset Mining using Utility Information Record

Prashant V. Barhate
Research Scholar,
MIT, Aurangabad.

S. R. Chaudhari
Department of Computer Science & Engineering, MIT, Aurangabad

P. C. Gill
Department of Computer Science & Engineering, MIT, Aurangabad

ABSTRACT
High utility itemsets refer to the sets of items with high utility like profit in a database, and efficient mining of high utility itemsets plays an important role in many real life applications and is an important research issue in data mining area. In recent years, the problems of high utility pattern mining become one of the most important research areas in data mining. The existing high utility mining algorithm generates large number of candidate itemsets, which takes much time to find utility value of all candidate itemsets.

In this paper we are implementing a data structure that stores the utility related to the item and using this data structure we are reducing time and space complexity of UP Growth and UP Growth+ Algorithms. Various Standard and synthetic datasets are used with Educational real data set. An algorithm is proposed to find set of high utility itemset which avoids the candidate itemsets generation.

Keywords
Utility, Utility Information Record, Effective High Utility Itemset Mining

1. INTRODUCTION
Data Mining can be described as an action that analyses the data and draws out some new nontrivial information from the large amount of databases. Data mining techniques have widely applied to extract useful rules or patterns in various practical applications, such as mobile data application and multimedia data applications. Discovering useful patterns hidden in a database plays an essential role in several data mining tasks, such as frequent pattern mining, association rule mining, and high utility pattern mining.

Mining frequent itemset [2] from the database DB is to find out set of itemset that occurs frequently. The frequency of itemset is the support count related to that itemset i.e. number of transactions containing that itemset. If the support of the itemset exceeds the *minimum support threshold* value then itemset is frequent.

Mining frequent itemset takes presence and absence of itemset into account, other relative information related to the item is not considered. To adderss this issue, the concept of weighted association rule mining was proposed. In weighted association rule mining, weights of items, such as unit profits of items in transaction databases, are considered. With this concept, even if some items appear infrequently, they might still be found if they have high weights. However, the quantities of items are not considered yet. This results in the research area of finding out high utility itemset from database. Utility is one of the important features of itemset in transaction that specifies a utility/profit of itemset with frequency.

Table 1: Profit table

Item	A	B	C	D	E	F	G	H
Profit	5	2	1	2	3	5	1	1

Recently, a number of high utility itemset mining algorithms [3] have been proposed. Most of the algorithms adopt a similar framework: initially generate candidate high utility itemsets from a database and then compute the exact utilities of the candidates by scanning the database to identify high utility itemsets. However, the algorithms often generate a very large number of candidate itemsets and thus are challenged with two problems:
(1) Excessive amount of memory required for storing candidate itemsets.
(2) A large amount of running time for generating candidates and computing their exact utilities.

Table 2 : Database Example

TID	Transaction	Quantity	TU
T_1	{A,C,D,}	{1,10,1}	17
T_2	{A,C,E,G}	{2,6,2,5}	27
T_3	{A,B,D,E,F}	{2,2,6,2,1}	37
T_4	{B,C,D,E}	{4,13,3,1}	30
T_5	{B,C,E,G}	{ 2,4,1,2}	13
T_6	{A,B,C,D,H}	{1,1,1,1,2}	12

When the number of candidates is so large that they cannot be stored in memory, the algorithms will fail or their performance will be degraded due to thrashing. To solve these problems, we propose in this paper an algorithm for high utility itemset mining.

The contributions of the paper are as follows:

1. A novel structure, called *utility information record*, is proposed. Utility information record stores the utility information about an itemset along with the heuristic information about whether the itemset should be pruned or not.

2. An efficient algorithm, called Efficient High Utility Itemset Mining (EHUIM) Algorithm, is developed. EHUIM Algorithm does not generate candidate high utility itemsets. After constructing the initial *utility information-record* from a mined database, EHUIM Algorithm, can mine high utility itemsets from these *utility information record.* We are using various standard and real data sets [4].

2. BACKGROUND

2.1 Problem Definition

Let I= {i1, i2, i3, . . . , in} be a set of items.the database DB is composed of a utility table and a transaction table. Each item in I has a utility value in the utility table. Each transaction T in the transaction table has a unique identifier (tid). An itemset is a subset of I and is called a k-itemset if it contains k items.

Definition 1. The external utility of item i, denoted as ext_u(i), is the utility value of i in the utility table of DB.

Definition 2. The internal utility of item i in transaction T, denoted as int_u(i, T), is the count value associated with i in T in the transaction table of DB.

Definition 3. The utility of item i in transaction T, denoted as u(i, T), is the product of int_u(i, T) and ext_u(i), where u(i, T) = int_u(i, T) × ext_u(i).

For example, in Table 2 and 3, ext_u(e) = 3, int_u(e, T5) = 1, and u(e,T5)= int_u(e, T5) × ext_u(e) = 1 x 3 =3.

Definition 4. The utility of itemset X in transaction T, denoted as u(X, T), is the sum of the utilities of all the items in X in T in which X is contained, where

u(X, T) = $\sum_{i \in X \wedge X \subseteq T} u(i, T)$

Definition 5. The utility of itemset X, denoted as u(X),is the sum of the utilities of X in all the transactions containing X in DB, where u(X) = $\sum_{T \in DB \wedge X \subseteq T} u(X;\ T)$

For example, in Table 2, u({ae}, T2) = u(a, T2) + u(e, T2)= 2 × 5 + 2 × 3 = 16, and u({ae}) = u({ae}, T2) + u({ae},T5) = 16 + 14 = 30.

Definition 6. The utility of transaction T, denoted as tu(T), is the sum of the utilities of all the items in T, where tu(T) =$\sum_{i \in T} u(i, T)$*,*and the total utility of DB is the sum of the utilities of all the transactions in DB.

2.2 Related Work

Many algorithms have been proposed for high utility itemset mining such as, Two-Phase [6], IHUP [7], and UP-Growth[5]. Two-Phase algorithm [6] was proposed by Liu et al. the algorithm consists of two phases. In phase I, Two-Phase algorithm employs Apriory based method to enumerate HTWUIs. It generates next set of candidate itemsets from the previous set of candidate itemsets and prunes candidate itemsets by TWDC property. In each pass, HTWUIs and their estimated utility values are computed by scanning database. After this, the complete set of HTWUIs is collected. In phase II, the original database is scanned to find out the high utility itemsets and their utilities

Although Two-Phase algorithm effectively reduces the search space and finds the complete set of high utility itemsets, it still generates too many candidates for HTWUIs and requires multiple database scans. To address this issue Ahmed et al. [7] proposed a tree-based algorithm, called IHUP. To maintain the information of high utility itemsets and transactions the algorithm uses an IHUP-Tree. Every node in IHUP-Tree consists of an item name, a support count, and a TWU value. The algorithm works in three steps, in first, items in the transaction are rearranged in a fixed order such as lexicographic order. The IHUP-tree is then constructed using rearranged transactions. In the second step, HTWUIs are generated from the IHUP-Tree. In third step, by scanning the original database, high utility itemsets and their utilities are identified from the set of HTWUIs.

Although IHUP finds HTWUIs without generating any candidates for HTWUIs and achieves a better performance than Two-Phase, it still produces too many HTWUIs in phase I. To address this issue, Vincent S. Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S. Yu [5] proposed the UP-Growth algorithm. A compact tree structure, called utility pattern tree (UP-Tree), for discovering high utility itemsets and maintaining important information related to utility patterns within databases are proposed. High-utility itemsets can be generated from UP-Tree efficiently with only two scans of original databases. Four new strategies are proposed namely DGU, DGN, DLU and DLN. First two strategies are applied on UP Tree to globally reduce unpromising items from obtained potential high utility itemsets. The next two strategies namely DLU and DLN are applied by the UP-Growth on the UP-Tree for reducing the local unpromising items. The actual high utility itemsets are then denfied from a set of potential high utility itemsets.

All these algorithms first produce candidate itemset which require more time and space. Here in this algorithm a search space from the UP Growth algorithm [5] is minimized. A Utility information record structure is used instead of UP Tree.

3. PROPOSED METHOD

The framework of the proposed method consists of following steps: 1) Scan database to construct utility Information Record. 2) Apply EHUI mining algorithm. 3) Generate High Utility Itemsets.

3.2 Utility Information Record Structure

In the section, we propose a utility information record structure to maintain the utility information about a database.

3.1.1 Initial Utility information record

Initial utility information record for storing the utility information about a mined database can be constructed by two scans of the database. Firstly, the transaction-weighted utilities of all items are collected by performing a database scan. If the transaction-weighted utility of an item is less than a given *minutil*, the item is no longer considered. For the items whose transaction-weighted utilities exceed the *minutil*, they are sorted in transacion-weighted-utility-ascending order.

3.1.2 Utility information record of 2-Itemsets

For constructing the utility information record of 2-itemsets there is no need to scan the database. The utility information record of 2-itemset{xy} can be constructed by the intersection of the utility list of {x} and that of {y}. The common transactions are identified by comparing the tids in the two utility information records by the algorithm. Suppose the lengths of the utility-information records are m and n respectively, and then (m + n) comparisons at most are

enough for identifying common transactions, because all tids in a utility information record are ordered. The identification process is actually a 2-way comparison.

3.1.3 Utility information record of k-Itemsets (k≥3)

To construct the utility information record of k-itemset $\{i_1 \cdots i_{(k-1)}i_k\}(k\geq 3)$, we can directly intersect the utility information record of$\{i_1 \cdots i_{(k-2)}i_{(k-1)}\}$and that of $\{i_1 \cdots i_{(k-2)}i_k\}$ as we do to construct the utility information record of a 2-itemset.

Algorithm 1: *Build - Tree Generation Algorithm*

Input: *P.UIR*, the utility information record of itemset *P*;
Px.UIR, the utility information record of itemset *Px*;
Py.UIR, the utility information record of itemset *Py*.

Output: *Pxy.UIR*, the utility information record of itemset *Pxy*.

1. *Pxy.UIR = NULL*;
2. **for each** element Ex ∈Px.UIR **do**
3. **if** ∃*Ey*∈*Py.UIR* and *Ex.tid==Ey.tid* **then**
4. **if** *P.UIR* is not empty **then**
5. search such element *E*∈*P.UIR* that

   E.tid==Ex.tid;
6. *Exy=<Ex.tid, Ex.iutil+Ey.iutil -E.iutil,*

   Ey.rutil>;
7. **Else**
8. *Exy=<Ex.tid, Ex.iutil+Ey.iutil, Ey.rutil>*;
9. **end if**
10. append *Exy to Pxy.UIR*;
11. **end if**
12. **end for**
13. **return** *Pxy.UIR*;

3.2 EHUIM Algorithm

After constructing a Utility information record a EHUIM Algorithm can mine all high utility itemset from database.

3.2.1 Domain Space

The domain space of the high utility itemset mining problem can be represented as a combination tree. Given a set of items $I = \{i_1, i_2, i_3, \ldots i_n\}$ and a total order on all items (suppose $i_1 < i_2 < \cdots < i_n$), a combination tree representing all itemsets can be constructed as follows.

Firstly, the root of the tree is created; secondly, the n-child nodes of the root representing n 1-itemsets are created, respectively; thirdly, for a node representing itemset$\{i_s \cdots i_e\}$ $(1 \leq s \leq e < n)$, the (n−e) child nodes of the node representing itemsets $\{i_s \cdots i_e i_{(e+1)}\}$,$\{i_s \cdots i_e i_{(e+2)}\}$, ...,$\{i_s \cdots i_e i_n\}$ are created. The third step is done repeatedly until all leaf nodes are created. For example, given I = {e, c, b, a, d} and e < c < b < a < d, a combination tree representing all itemsets of I is depicted in Fig. 1.

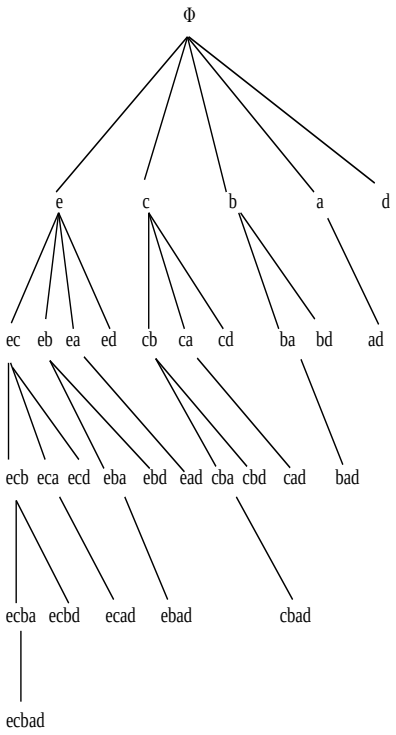

Figure 1: Combination Tree

3.2.2 Pruning Strategy

For a database with n items, exhaustive search has to check 2^n itemsets. To reduce the search space, we can exploit the iutils and rutils in the utility information record of an itemset. The sum of all the iutils in the utility information record of an itemset is the utility of the itemset according to Definition 5, and thus the itemset is high utility if the sum exceeds a given minutil. The sum of all the iutils and rutils in the utility information record provides EHUIM Algorithm with the key information about whether the itemset should be pruned or not.

Lemma 1. Given the utility information record of itemset X, if the sum of all the iutils and rutils in the utility information record is less than a given "minutil", any extension X' of X is not high utility.

3.2.3 EHUI Mining Algorithm

Algorithm 2 shows the pseudo-code of EHUIM Algorithm. For each utility information record X in ULs (the second parameter), if the sum of all the iutils in X exceeds minutil, and then the extension associated with X is high utility and outputted. According to Lemma 1, only when the sum of all the iutils and rutils in X exceeds minutil should it be processed further. The initial utility information records are constructed from a database and they are sorted and processed in transaction-weighted utility ascending order. Therefore, all the utility information records in UIRs are ordered as the initial utility information record are. To explore the search space, the algorithm intersects X and each utility information record Y after X in UIRs. Suppose X is the utility information record of itemset Px and Y that of itemset Py, and then

Build(P.UIR, X, Y) in line 8 is to construct the utility information record of itemset Pxy as stated in Algorithm 1. Finally, the set of utility information record of all the 1-extensions of itemset Px is recursively processed. Given a database and a minutil, after the initial utility information record IUIRs are constructed, EHUIM(∅, IUIRs, minutil) can mine all high utility itemsets.

Algorithm 2: *EHUI Mining Algorithm*

Input: *P.UIR*, the utility information record of itemset *P*, initially empty; *UIRs*, the set of utility information record of all *P*'s 1-extensions;
minutil, the minimum utility threshold.

Output: all the high utility itemsets with *P* as prefix.

1. **for each** utility information record *X* in *UIRs* **do**
2. **if** SUM(*X.iutils*)≥*minutil* **then**
3. output the extension associated with *X*;
4. **end if**
5. **if** SUM(*X.iutils*)+SUM(*X.rutils)*≥*minutil*
   then
6. *exULs = NULL*;
7. **for each** utility information record *Y* after *X* in *UIRs* **do**
8. *exUIRs = exUIRs*+Build(*P.UIR, X, Y*);
9. **end for**
10. EHUI(*X*, *exULs*, *minutil*);
11. **end if**
12. **end for**

4. EXPERIMENTAL EVALUATION

Performance of proposed algorithm is evaluated in this section. The experiments were performed on 2.20 GHz Core2 Duo Processor with 2GB memory. The operating system is Linux Fedora 14. The algorithms are implemented in Java language. Both real and standard datasets are used in this experiment. Standard data sets are obtained from FIMI Repository. Real datasets were generated from the actual values. Parameter descriptions and default values of datasets are shown in Table no. Educational dataset for evaluation of feedback report of faculty member is used as a real dataset.

Table 3: Statistics about Databases

Dataset	Chess Dataset	Feedback Dataset
Size	642kb	26kb
Transactions	3196	500
Items	75	10
Avg Length	37	10

4.2 Performance comparison on different data sets

Running Time

When measuring running time, we varied the *minutil* for each database. The lower the *minutil* is, the larger the number of high utility itemsets is, and thus the more the running time is. For example, for database *chess* in Fig.4 , when the *minutils* are 80% and 90%, the running times of EHUI are 1400 mSec and 800 mSec.

For almost all databases and *minutils*, EHUI performs the best. In Fig. , EHUI is slower than UPGrowth and UPGrowth+ for low *minutils*, and we found out in this case that UPGrowth+ requires less time. However, for high *minutils*, EHUI is even an order of magnitude faster than UPGrowth and UPGrowth+. For the Educational Feedback Dataset, when *minutils* are 50%, 60%, and 70%, the running time required for EHUI are 40mSec.

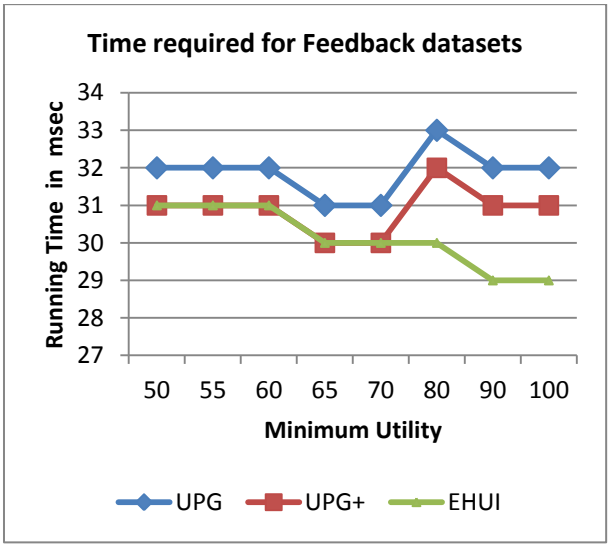

Figure 2: Time for Educational Dataset (Separate File)

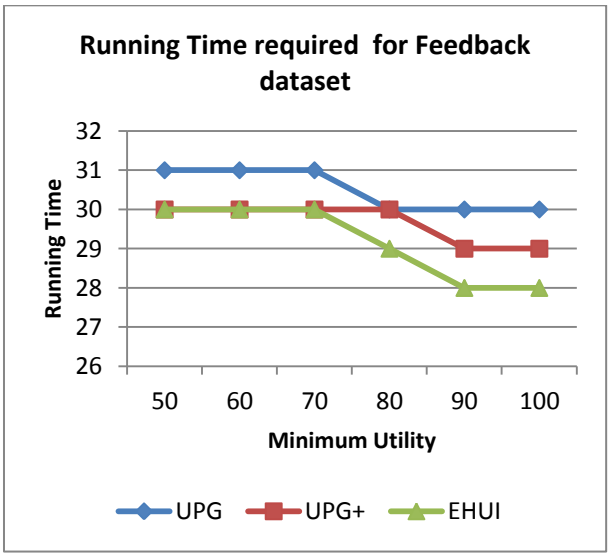

Figure 3: Time for Educational Dataset (Combine File)

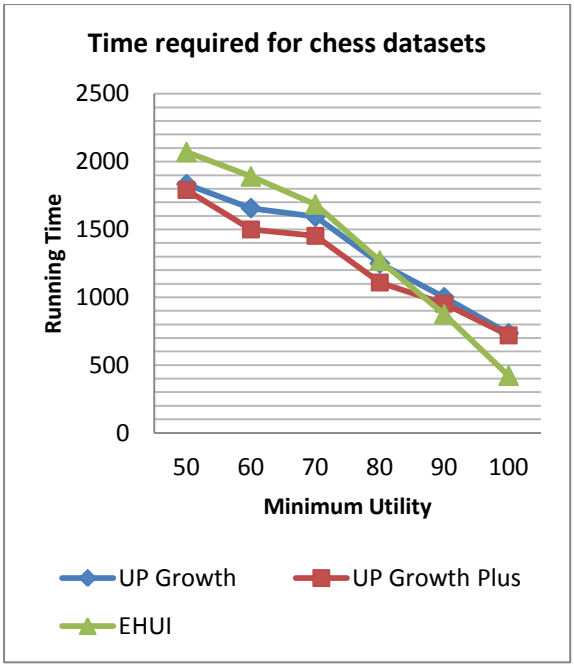

Figure 4: Time for Chess Dataset

Memory Consumption

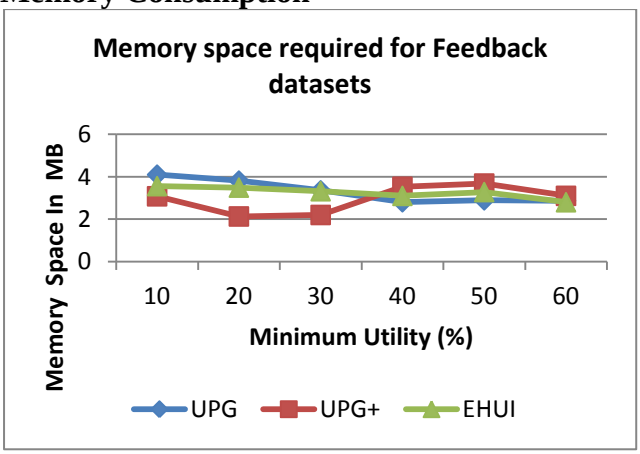

Figure 5: Memory Space for Educational Dataset (Separate File)

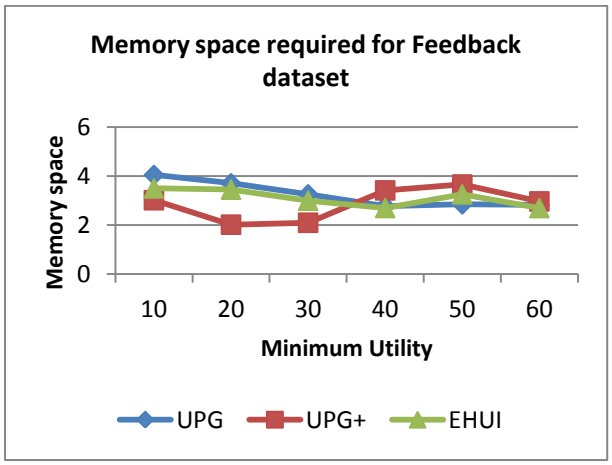

Figure 6: Memory Space for Educational Dataset (Combine File)

Generally, the memory consumption of the algorithms is proportional to the number of candidate itemsets they generate. For example, for database *Chess*, UP Growth generates 623, UP Growth+ generates 551 and that of IHUP generates 30 candidate itemsets and consumes 17.60MB, 21.81MB, and 16.02MB of memory respectively. Similar case is there for Educational Feedback Dataset. EHUI require less space than UPGrowth and in some cases of UPGrowth+ algorithm.

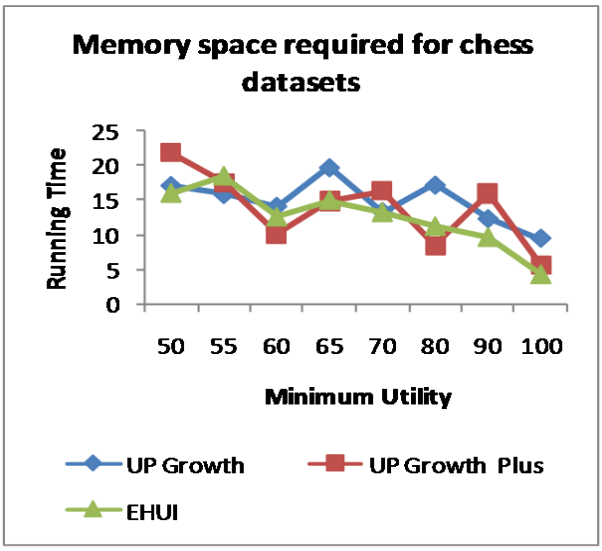

Figure 7: Memory Space for Chess Dataset

Itemsets Found

Higher the number of candidate itemsets in a algorithm, lower is the performance. EHUI mining always generate less number of itemsets than UPGrowth and UPGrowth+. In both chess and educational feedback datasets, the numbers of generated itemsets are less for EHUI mining algorithm for high value of *minutil*.

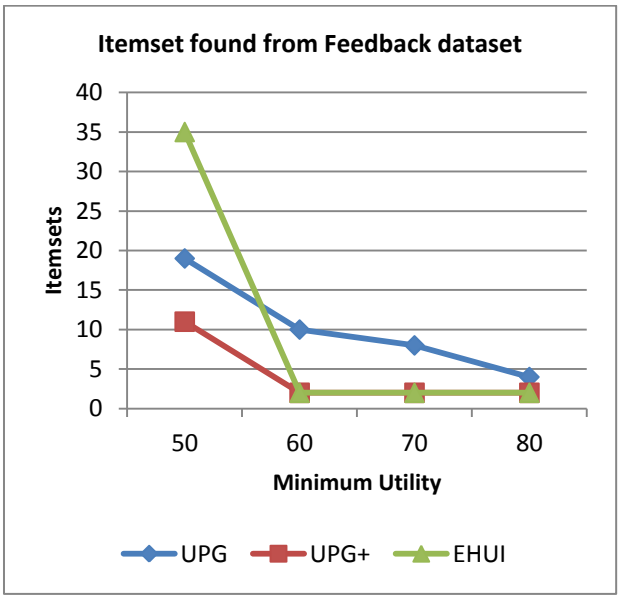

Figure 8: Itemset for Educational Dataset (Separate File)

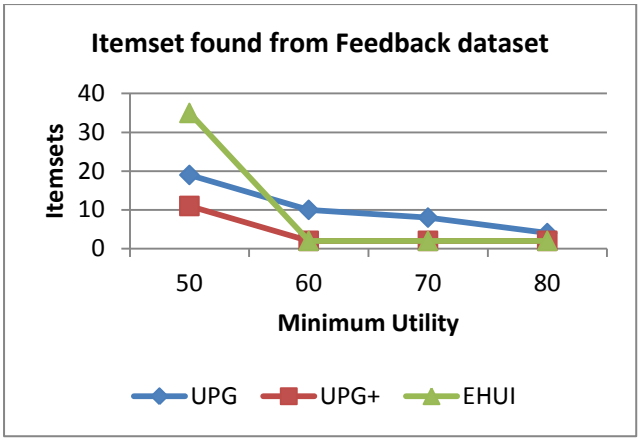

Figure 9: Itemset for Educational Dataset (Combine File)

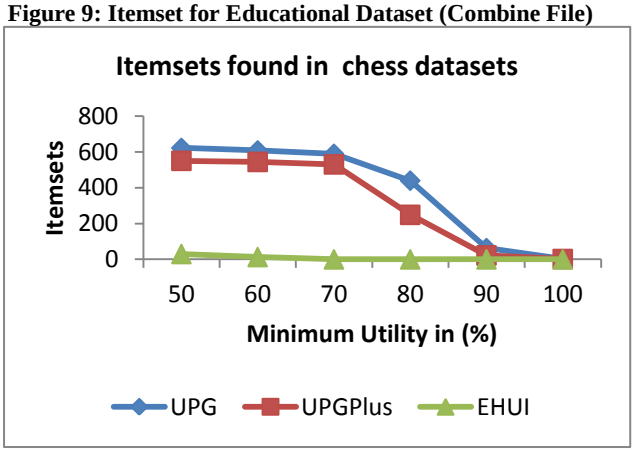

Figure 10: Itemset for Chess Dataset

5. CONCLUSION

In this paper, we have proposed a novel data structure, utility information record, and developed an efficient algorithm, EHUI, for high utility itemset mining. Utility information record provide not only utility information about itemsets but also important pruning information for EHUI. We have used Educational real time and standard datasets. Previous algorithms have to process a very large number of candidate itemsets during their mining processes. However, most candidate itemsets are not high utility and are discarded finally. EHUI Algorithm can mine high utility itemsets without candidate generation, so that complexity of UPGrowth and UPGrowth+ is reduced as it require less time and space, which avoids the costly generation and utility computation of candidates. However in future we can again reduce the complexity by reducing the joining cost of utility information record.

6. REFERENCES

[1] Jyothi Pillai, O.P.Vyas "Overview of Itemset Utility Mining and its Applications" IJCA(0975 – 8887) Volume 5– No.11, August 2010.

[2] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery, 15(1):55–86, 2007.

[3] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K.Lee. Efficient tree structures for high utility patternmining in incremental databases. IEEE Transactions onKnowledge and Data Engineering, 21(12):1708–1721,2009.

[4] Frequent Itemset Mining Implementations Repository, http://fimi.cs.helsinki.fi/, 2013.

[5] Vincent S. Tseng, Bai-En Shie, Cheng Wei Wu, and Philip S. Yu, Fellow, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases" IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 25, NO. 8, AUGUST 2013.

[6] Y. Liu, W. Liao, and A. Choudhary. A fast high utility itemsets mining algorithm. In Proc. of the Utility-Based Data Mining Workshop, 2005.

[7] 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.