Applied Mathematics and Computation 215 (2009) 767–778

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

An efficient algorithm for mining high utility itemsets with negative item values in large databases

Chun-Jung Chu [a,*], Vincent S. Tseng [b], Tyne Liang [a]

[a] Dept. of Computer Science, National Chiao Tung University, Taiwan, ROC
[b] Dept. Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC

ARTICLE INFO

Keywords:
Utility mining
High utility itemsets
Association rules

ABSTRACT

Utility itemsets typically consist of items with different values such as utilities, and the aim of utility mining is to identify the itemsets with highest utilities. In the past studies on utility mining, the values of utility itemsets were considered as positive. In some applications, however, an itemset may be associated with negative item values. Hence, discovery of high utility itemsets with negative item values is important for mining interesting patterns like association rules. In this paper, we propose a novel method, namely *HUINIV* (High Utility Itemsets with Negative Item Values)-*Mine*, for efficiently and effectively mining high utility itemsets from large databases with consideration of negative item values. To the best of our knowledge, this is the first work that considers the concept of negative item values in utility mining. The novel contribution of *HUINIV-Mine* is that it can effectively identify high utility itemsets by generating fewer high transaction-weighted utilization itemsets such that the execution time can be reduced substantially in mining the high utility itemsets. In this way, the process of discovering all high utility itemsets with consideration of negative item values can be accomplished effectively with less requirements on memory space and CPU I/O. This meets the critical requirements of temporal and spatial efficiency for mining high utility itemsets with negative item values. Through experimental evaluation, it is shown that *HUINIV-Mine* outperforms other methods substantially by generating much less candidate itemsets under different experimental conditions.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

Mining of association rules in large databases is a well studied technique in the field of data mining with typical methods like *Apriori* [1,2]. The problem surrounding association rules mining can be decomposed into two steps. The first step involves finding all frequent itemsets (or large itemsets) in a database. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

Most methods in finding frequent itemsets are designed for traditional databases. However, the frequency of an itemset may not be a sufficient indicator of significance, because frequency reflects only the number of transactions in the database that contain that itemset. It does not reveal the *utility* of an itemset, which can be measured in terms of cost, profit, or other expressions of user preference. On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be most interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company).

* Corresponding author. Address: Dept. of Computer Science, National Chiao Tung University, No. 1001, Ta-Hsueh Road, Hsinchu, Taiwan, ROC.
E-mail addresses: cjchu@cis.nctu.edu.tw (C.-J. Chu), tsengsm@mail.ncku.edu.tw (V.S. Tseng), tliang@cis.nctu.edu.tw (T. Liang).

0096-3003/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2009.05.066

Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact.

Utility mining is thus useful in a wide range of practical applications and was recently studied in [7,14,20,22]. However, a retail business may sale item with negative value. For example, many super markets may promote certain items to attract customers. In this scenario customers may buy specific items and then receive free goods. Free goods result in negative value for super markets. However, supermarkets may earn higher profits from other items that are cross-promoted with these free items. This practice is common. For example, if a customer bought three of item A, he would then receive one free item B as a promotion from the supermarket. Suppose the supermarket gets five dollars of profit from each unit of item A sold, and loses two dollars for each unit of item B given away. Although giving away a unit of item B results in a loss of two dollars for supermarkets, they could possibly earn 15 dollars from the three units of item A that are cross-promoted with item B. The supermarket thus may have a net gain of 13 dollars from this promotion. This example demonstrates why we propose the concept of mining for negative item values in utility mining. This also motivates our research in developing a new scheme for finding *high utility itemsets with negative item values (HUINIV)* from large databases.

Recently, a *utility mining* model has been defined in [22]. Utility is a measure of how "useful" (i.e. "profitable") an itemset is. The definition of the utility of an itemset X, $u(X)$, states that it is equal to the sum of the utilities of X of all the transactions containing X. The goal of utility mining is to identify high utility itemsets, which drive a large portion of the total utility. Traditional association rules of mining models assume that the utility of each item is always 1 and that the quantity of sales is either 0 or 1; thus it is only a special case of utility mining in which the utility or the quantity of sales of each item can be any number. If $u(X)$ is greater than a specified utility threshold, X is a high utility itemset; otherwise, it is a low utility itemset. Table 1 is an example of utility mining in a transaction database. The number associated with each transaction in Table 1a is the sales volume of each item, and the utility of each item is listed in Table 1b. For example, $u(\{A,D\}) = (1 \times 5 + 2 \times 6) + (3 \times 5+5 \times 6) + (3 \times 5 + 1 \times 6) = 83$. $\{A,D\}$ is a high utility itemset if the utility threshold is set at 80.

However, a high utility itemset may consist of low utility items. Another possibility is to adopt the level-wise searching schema that exists in fast algorithms, such as Apriori [3]. This algorithm does not apply to the *utility mining* model. For example, $u(A) = 55 < 80$, A is a low utility item, but its superset $\{A,D\}$ is a high utility itemset. If Apriori is used to find high utility itemsets, all combinations of all items must be generated. Moreover, in order to discover a long pattern, the number of candidates is prohibitively large. The cost in terms of either computation time or memory is intolerable, regardless of the method utilized. The challenge of utility mining is not only in restricting the size of the candidate set but also in simplifying the computation used to calculate its utility. Another challenge of utility mining is finding high utility itemsets with negative item values from large databases.

In this paper, we explore the issue of efficiently mining high utility itemsets with negative item values in large databases. We propose an algorithm named *HUINIV (High Utility Itemsets with Negative Item Values)-Mine* that can discover high utility itemsets with negative item values from large databases both efficiently and effectively. The underlying idea behind the *HUINIV-Mine* algorithm is based on the principle of the Two-Phase algorithm [14] and augments with negative item value for mining high utility itemsets efficiently. The novel contribution of *HUINIV-Mine* is that it can efficiently identify the utility of itemsets in large database so that the execution time for producing high utility itemsets with negative item values can be substantially reduced. That is, *HUINIV-Mine* can discover high utility itemsets with negative item values using limited memory and comparatively less computation time by the candidate itemsets filter method. In this way, the process of discovering all high utility itemsets in which all transactions are negative can be achieved effectively with limited memory, less

Table 1
A transaction database and its utility table.

TID	Item				
	A	*B*	*C*	*D*	*E*
(a) Transaction table					
T_1	1	0	0	2	1
T_2	0	1	2	6	0
T_3	3	0	0	5	0
T_4	1	0	0	0	1
T_5	0	1	2	6	0
T_6	0	1	1	0	2
T_7	2	0	0	0	0
T_8	3	0	0	1	0
T_9	0	1	1	4	0
T_{10}	1	0	0	0	1
Item					Value ($) (per unit)
(b) The utility table					
A					5
B					−3
C					−2
D					6
E					10

candidate itemsets, and CPU I/O. This meets the critical requirements of time and spatial efficiency for mining large databases. Through experimental evaluation, *HUINIV-Mine* is shown to produce fewer candidate itemsets in the process of finding high utility itemsets with negative item values, so it outperforms other methods in terms of efficiency. We found that the average improvement of *HUINIV-Mine* compared to the MEU algorithm is about 99.2%. Moreover, it also achieves high scalability in dealing with large databases. To the best of our knowledge, this is the first work to propose a negative item concept in utility mining and the first work on mining high utility itemsets with negative item values from large database.

The rest of this paper is organized as follows: Section 2 provides an overview of related work. Section 3 describes the proposed approach, *HUINIV-Mine*, for finding the high utility itemsets with negative item values. In Section 4, we describe our experimental results for evaluating the proposed method. The conclusion of the paper is provided in Section 5.

2. Related work

Apriori [3], DHP [16], Partition [13], and Anti-Skew Algorithms [17] have been proposed to find frequent itemsets using association rules mining. Many important applications have established the need for incremental mining. This is due to the increasing use of record-based databases to which data are being added continuously. Many algorithms like FUP [8], FUP_2 [9] and UWEP [4,5] have been proposed to find frequent itemsets in incremental databases. The FUP algorithm updates the association rules in a database when new transactions are added to it. The algorithm FUP is based on the framework of Apriori and is designed to discover new frequently occurring itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets is then obtained by scanning the original database. An extension to the work in [8] was reported in [9] where the authors proposed the algorithm FUP_2 for updating the existing association rules when transactions are added to or deleted from the database. UWEP (Update With Early Pruning) is an efficient incremental algorithm, that counts the original database at most once, and the increment exactly once. In addition, the number of candidates generated and counted is minimized.

A formal definition of utility mining and a theoretical model were proposed in [22], namely MEU, where utility is defined as the combination of utility of information in each transaction with additional resources. Since this model cannot rely on the downward closure property of Apriori to restrict the number of itemsets to be examined, a heuristic is used to predict whether an itemset should be added to the candidate set. However, the prediction usually overestimates; especially at the beginning stages, where the number of candidates approaches the number of all the combinations of items. The examination of all combinations is impractical, in terms of both computational cost and memory space, whenever the number of items is large or the utility threshold is low. Although this algorithm is not efficient or scalable, it is by far the best available to solve this specific problem.

Another algorithm named Two-Phase, proposed in [14], is based on the definition in [22] and is able to the find high utility itemsets. The Two-Phase algorithm is used to prune down the number of candidates and can obtain the complete set of high utility itemsets. In the first phase, a model that applies to the "transaction-weighted downward closure property" of search space is used to expedite the identification of candidates. In the second phase, one extra database scan is performed to identify high utility itemsets. However, this algorithm can not deal with negative item values in utility mining. In order to find high utility itemsets with negative item values some candidate itemsets are lost. Hence, the Two-Phase algorithm focuses on positive item values and is not suited to negative item values in utility mining.

An algorithm named THUI (Temporal High Utility Itemsets) was proposed in [20]. It is the first algorithm proposed for finding temporal high utility itemsets in temporal databases. The algorithm integrated the advantages of the Two-Phase algorithm [14] and the SWF algorithm [12] and augment with the incremental mining techniques for mining temporal high utility itemsets efficiently. However, this algorithm only focuses on high utility itemsets with positive item values and is not suited to negative item values. Hence, the algorithm cannot find high utility itemsets with negative item values.

Although there exist numerous studies on high utility itemset mining as described above, there is no algorithm proposed to find high utility itemsets with negative item values in large databases. This motivates our exploration of the issue of efficiently mining high utility itemsets with negative item values from large databases.

3. Proposed method: *HUINIV-Mine*

In this section, we present the *HUINIV-Mine* method. Section 3.1 describes the basic concept of *HUINIV-Mine*. Section 3.2 gives an example of mining temporal high utility itemsets. The procedure of the *HUINIV-Mine* algorithm is provided in Section 3.3.

3.1. Basic concept of HUINIV-Mine

The goal of utility mining is to discover all itemsets whose utility values exceed a user specified threshold in a transaction database. In [22] the goal of utility mining is to find all high utility itemsets. An itemset X is a *high utility itemset* if $u(X) \geqslant \varepsilon$, where $X \subseteq I$ and ε is the minimum utility threshold, otherwise, it is a *low utility itemset*. For example, in Table 1, $u(A,T_1) = 1 \times 5 = 5$, $u(\{A,E\},T_1) = u(A,T_1) + u(E,T_1) = 1 \times 5 + 1 \times 10 = 15$, and $u(\{A,E\}) = u(\{A,E\},T_1) + u(\{A,E\},T_4) + u(\{A,E\},T_{10}) =$

$E\},T_1) + u(\{A,E\},T_4) + u(\{A,E\},T_{10}) = 15 + 15 + 15 = 45$. If $\varepsilon = 80$, $\{A,E\}$ is a low utility itemset. However, if an item is a low utility item, its superset may be a high utility itemset. For example, $u(A) = 55 < 80$, A is a low utility item, but its superset $\{A,D\}$ is a high utility itemset because $u(\{A,D\}) = 83 > 80$. Hence, all the combinations of all items should be processed in order to ensure that no high utility itemset will be lost. But the high cost of either computational time or memory is intolerable. In the formal definition of the utility mining problem, these terms can be generally defined as follows by referring to [22]:

- $I = \{i_1, i_2, \ldots, i_m\}$ is a set of items.
- $D = \{T_1, T_2, \ldots, T_n\}$ is a transaction database where each transaction $T_i \in D$ is a subset of I.
- $o(i_p, T_q)$, *local transaction utility value*, represents the quantity of item i_p in transaction T_q. For example, $o(A, T_3) = 3$, in Table 1a.
- $s(i_p)$, *external utility*, is the value associated with item i_p in the Utility Table. This value reflects the importance of an item, which is independent of transactions. For example, in Table 1b, the external utility of item A, $s(A)$, is 5.
- $u(i_p, T_q)$, *utility*, the quantitative measure of utility for item i_p in transaction T_q, is defined as: $o(i_p, T_q) \times s(i_p)$. For example, $u(A, T_3) = 3 \times 5$, in Table 1.
- $u(X, T_q)$, *utility of an itemset X in transaction Tq*, is defined as $\sum_{ip \in X} u(ip, Tq)$, where $X = \{i_1, i_2, \ldots, i_m\}$ is a k-itemset, $X \subseteq T_q$ and $1 \leqslant k \leqslant m$.
- $u(X)$, *utility of an itemset X*, is defined as $\sum_{Tq \in D \wedge X \subseteq Tq} u(X, Tq)$.

Liu et al. [14] proposed the Two-Phase algorithm for pruning candidate itemsets and simplifying the calculation of utility. First, Phase I overestimates some low utility itemsets, but it never underestimates any itemsets. For example in Table 1, the *transaction utility of transaction Tq*, denoted as $tu(T_q)$, is the sum of the utilities of all items in $Tq : tu(T_q) = \sum_{ip \in Tq} u(ip, Tq)$. And the *transaction-weighted utilization of an itemset X*, denoted as $twu(X)$, is the sum of the transaction utilities of all transactions containing $X : twu(X) = \sum_{X \subseteq Tq \in D} tu(Tq)$. For example, $twu(A) = tu(T_1) + tu(T_3) + tu(T_4) + tu(T_7) + tu(T_8) + tu(T_{10}) = 27 + 45 + 15 + 10 + 21 + 15 = 133$ and $twu(\{A,E\}) = tu(T_1) + tu(T_4) + tu(T_{10}) = 27 + 15 + 15 = 57$. In fact, $u(A) = u(\{A\},T_1) + u(\{A\},$-$(A) = u(\{A\},T_1) + u(\{A\},T_3) + u(\{A\},T_4) + u(\{A\},T_7) + u(\{A\},T_8) + u(\{A\},T_{10}) = 5 + 15 + 5 + 10 + 15 + 5 = 55$ and $u(\{A,E\}) = u(\{A,E\},$-$(\{A,E\}) = u(\{A,E\},T_1) + u(\{A,E\},T_4) + u(\{A,E\},T_{10}) = 15 + 15 + 15 = 45$. So while Phase I overestimates some low utility itemsets, it never underestimates any itemsets whatsoever. Table 2 gives the transaction utility of each transaction in Table 1. One extra database scan is performed to filter the overestimated itemsets in phase II. For example, $twu(A) = 126 > 80$ but $u(A) = 55 < 80$. After estimating the utility of item $\{A\}$, item $\{A\}$ is pruned since its utility, 55, is lower than 80,; otherwise, it is a high utility itemset. In the end, all of high utility itemsets have been discovered in this way. However, we cannot apply the Two-Phase algorithm to databases whose items include negative values. Some high utility itemsets may be lost in this way. For example, $twu(\{B,D\}) = tu(T_2) + tu(T_5) + tu\ (T_9) = 29 + 29 + 19 = 77$. If $\varepsilon = 80$, $twu(\{B,D\}) = 77 < 80$ is a low transaction-weighted utilization itemset, then $\{B,D\}$ will be deleted. In fact, $u(\{B,D\}) = u(\{B,D\},\ T_2) + u(\{B,D\},\ T_5) + u(\{B,D\},\ T_9) = 33 + 33 + 21 = 87 > 80$. $\{B,D\}$ should be a high utility itemset. Thus, the Two-Phase algorithm is not sufficient to answer question regarding items with negative values. As one possible solution that find high utility itemset, utility mining, useful over a wide range of practical applications, was recently studied in [7,14,20,22]. This also motivates our research in developing a new scheme for finding *high utility itemsets with negative item values (HUINIV)* from large databases.

Our algorithm *HUINIV-Mine* is based on the principle of the Two-Phase algorithm [14], and focuses on improving the response time by reducing candidate itemsets and CPU I/O in using transaction itemsets without negative value.. In essence, by removing items with negative values from a transaction in a large database, algorithm *HUINIV-Mine* employs a filtering threshold within the database to deal with the transaction-weighted utilization itemsets (TWUI) generated. Table 3 gives the transaction utility without negative item values for each transaction in Table 1. In this way, *HUINIV-Mine* can overestimate some low utility itemsets, but it never underestimates any itemsets and it never loses any itemsets that may be of high utility. In processing a database, a transaction-weighted utilization set of itemsets is generated by *HUINIV-Mine*. Explicitly, a transaction-weighted utilization set of itemsets is composed of the TWUI that were generated from the previous transaction-weighted utilization candidate sets during the previous phase. After the processing, the algorithm *HUINIV-Mine* outputs a high transaction-weighted utilization set of itemsets. However, some of the high transaction-weighted utilization sets of itemsets should be pruned in advance. Each item of the itemset that has negative value will never be part of a high utility itemset. At least one item within an timeset should have positive value, or the itemset need not scan the database. Hence, the algorithm *HUINIV-Mine* outputs real high transaction-weighted utilization candidate itemsets after filtering some itemsets.

Table 2
Transaction utility of the transaction database.

TID	Transaction utility	TID	Transaction utility
T_1	27	T_6	15
T_2	29	T_7	10
T_3	45	T_8	21
T_4	15	T_9	19
T_5	29	T_{10}	15

Table 3
Transaction utility without negative item values of the transaction database.

TID	Transaction utility without negative item values	TID	Transaction utility without negative item values
T_1	27	T_6	20
T_2	36	T_7	10
T_3	45	T_8	21
T_4	15	T_9	24
T_5	36	T_{10}	15

Finally, *HUINIV-Mine* computes the occurrence counts of itemsets in the memory and then deletes itemsets that do not satisfy utility threshold within the database so as to find high utility itemsets with negative item values.

Taking these design features under consideration, the algorithm *HUINIV-Mine* is shown to perform very well at mining high utility itemsets with negative item values from large databases. In Section 3.2, we give an example of mining high utility itemsets with negative item values from large databases. The proposed algorithm, *HUINIV-Mine*, is described in detail in Section 3.3.

3.2. An example of mining high utility itemsets with negative item values

The proposed *HUINIV-Mine* algorithm can be best understood from the illustrative transaction database shown in Table 1 and Fig. 1 in which a scenario for generating high utility itemsets from large databases to mine high utility itemsets with negative item values is given. This type of illustrative transaction database resembles items that are sold by supermarkets in real life. This also means that utility mining has real-life applications. We set the utility threshold at 80 with 10 transactions. Without loss of generality, the mining problem can be decomposed into two procedures:

1. TWUI procedure: This procedure deals with mining the transaction database to generate TWUI.
2. Filter procedure: The procedure deals with filtering negative itemsets and generating high utility itemsets with negative item values from large databases.

The TWUI procedure is only utilized for the initial utility mining in the database. For the mining high utility itemsets, the filter procedure is employed. Consider the database in Table 1. Each transaction is scanned sequentially for the generation of candidate 1-itemsets in the first scan of the database. Itemsets whose transaction-weighted utility is below the utility threshold are removed. Then, as shown in Fig. 1, only {*A*,*B*,*C*,*D*}, marked by "◎", remain as high transaction-weighted utilization 1-itemsets. Although items *B* and *C* have negative values, they may constitute high utility itemset by combining with

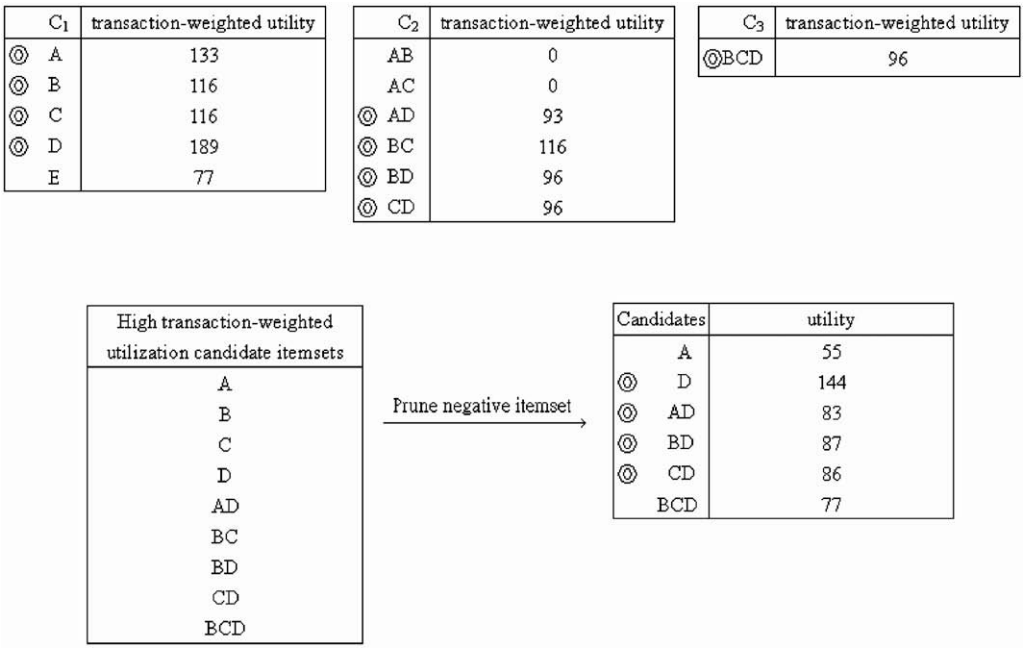

Fig. 1. High utility itemsets generated from database by *HUINIV-Mine*.

other items. These items should be preserved to combine with other items to generate the next candidate itemsets. The candidate 2-itemsets {*AB*,*AC*,*AD*,*BC*,*BD*,*CD*} are generated by high transaction-weighted utilization 1-itemsets. In the same way, only {*AD*,*BC*,*BD*,*CD*}, marked by "◎", remain as high transaction-weighted utilization 2-itemsets. The candidate 3-itemsets {*BCD*} are generated by high transaction-weighted utilization 2-itemsets; high transaction-weighted utilization 3-itemsets being those whose transaction-weighted utility is above the utility threshold.

We could get high transaction-weighted utilization candidate itemsets {*A*,*B*,*C*,*D*,*AD*,*BC*,*BD*,*CD*,*BCD*}. However, some of the high transaction-weighted utilization sets of itemsets should be pruned in advance. If each item of the itemset's value is negative, it will not be a high utility itemset. For example, {*B*,*C*,*BC*} should be deleted. Hence, algorithm *HUINIV-Mine* outputs only six real high transaction-weighted utilization candidate itemsets {*A*,*D*,*AD*,*BD*,*CD*,*BCD*} after filtering the itemsets. Finally, all candidates can be stored in main memory, and we can find high utility itemsets with negative item values when the scan of the database is performed. The resulting high utility itemsets are {*D*}, {*AD*}, {*BD*} and {*CD*} because $u(D) = 144 > 80$, $u(\{A,D\}) = 83 > 80$, $u(\{B,D\}) = 87 > 80$ and $u(\{B,E\}) = 86 > 80$ as shown in Fig. 1.

3.3. HUINIV-Mine Algorithm

For clarification, the meanings of various symbols used are given in Table 4. The algorithm of *HUINIV-Mine* is shown in Fig. 2. Initially, it input the database DB (in step 1), and it finds high transaction-weighted utilization 1-itemsets from step 2 to step 5. The transaction-weighted utility without negative item value of itemset I is recorded in **I.twu**. An itemset, whose **I.twu** $\geqslant$ threshold (in step 3), will be kept in **htwu** (in step 4). Next, we use high transaction-weighted utilization itemsets to generate transaction-weighted utilization candidate itemsets. Then we scan the database to find high transaction-weighted utilization itemsets from step 6 to step 13. After identifying all **htwu**, we perform a last scan of the database from Step 14 to Step 19. Since each item's value in the itemset is negative, it cannot be a high utility itemset. At least one item's value in itemset I should be positive (in step 14), or else the itemset does not need to scan the database. Finally, those itemsets satisfying the constraint that **I.htwu** $\geqq$ threshold are finally obtained as the high utility itemsets with negative item values.

4. Experimental evaluation

To evaluate the performance of *HUINIV-Mine*, we conducted experiments using synthetic datasets generated via a randomized dataset generator provided by IBM Quest [3]. However, the IBM Quest data generator only generates quantities of 0 or 1 for each item in a transaction. In order to fit databases into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5; much like the model used in [14,20]. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from −100 to 1000. Since it is observed from real databases that most items are in the low value range and low negative value range, we generate the utility values using a log normal distribution; similarly to the model used in [14,20]. Fig. 3 shows the utility value distribution of 1000 items.

The simulation is implemented in C++ and conducted in a machine with 2.4 GHz CPU and 1G memory. The MEU algorithm [22] is also utilized in a negative itemsets scenario for comparison with the *HUINIV-Mine* algorithm. The scenario using MEU consists of scanning the database after collecting the data to find high utility itemsets with negative item values. The main performance metric used is execution time. We recorded the time that *HUINIV-Mine* uses to find high utility itemsets with negative item values. The number of candidate itemsets compared of *HUINIV-Mine* and MEU is presented in Section 4.1. Section 4.2 shows comparison in performance of a variety of IBM Quest data with *HUINIV-Mine*. The results of scale-up experiments are presented in Section 4.3. Section 4.4 shows the performance of *HUINIV-Mine* with real data.

4.1. Evaluation of number of generated candidates

In this experiment, we compare the average number of candidates generated in the first database scan with different support values for *HUINIV-Mine* and MEU [22]. Tables 5–8 show the average number of candidates generated by *HUINIV-Mine* and MEU. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The number of candidate itemsets generated by *HUINIV-Mine* during the first database scan decreases dramatically as the threshold increases. Particularly when the utility threshold is set to 1%, the number of candidate itemsets is generally 588, including all various

Table 4
Meanings of symbols used.

DB	Database
Threshold	Utility threshold in database
twu	Transaction-weighted utilization itemsets without negative item values
htwu$_i$	High transaction-weighted utilization i-itemsets without negative item values
I.value	Each item's value
hui	High utility itemsets with negative item values

Algorithm HUINIV-Mine

```
1.  Input: DB
2.  for k = 1 to n                    // Find high transaction-weighted utilization 1-itemsets
3.    if (I.twu ≧ threshold )
4.        htwu1 = htwu1 ∪ I.twu;
5.  end
6.  for i=2 ; htwui-1 ≠ φ; i++
7.    I.twu =I.twu_gen(htwui-1);    // Generate transaction-weighted utilization i-candidate itemsets
8.    if (I.twu ≠ φ)
9.        for k=1 to n
10.         if (I.twu ≧ threshold )
11.             htwui = htwui ∪ I.twu;
12.       end
13. end
14. for each itemset I∈htwu ≠ φ && I.value > 0 // Find high utility itemsets with negative item values
15.   for k = 1 to n
16.     if (I.htwu ≧ threshold)
17.         hui= hui ∪ I.htwu;
18.  end
19. end
20. return hui;
```

Fig. 2. Algorithm of *HUINIV-Mine*.

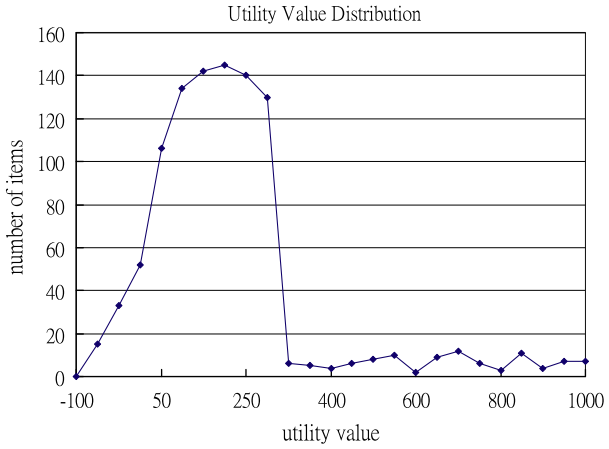

Fig. 3. Utility value distribution in utility table.

Table 5
The number of candidate itemsets and high utility itemsets generated from database T10.I4.D100K.

Threshold (%)	Databases		
	T10.I4.D100K		
	HUINIV-Mine	MEU	High utility itemsets
0.2	11,306	499,500	285
0.3	3928	499,500	183
0.4	1691	499,500	115
0.6	846	499,500	58
0.8	676	499,500	29
1	588	499,500	14

candidate itemsets in database T10.I4.D100K where T denotes the average size of the transactions and I the average number of frequent itemsets. However, the number of candidates generated by MEU is always 499,500 because it needs to process all combinations of 1000 items to generate only 2-candidate itemsets. *HUINIV-Mine* generates far fewer candidates when compared to MEU.

Table 6
The number of candidate itemsets and high utility itemsets generated from database T10.I6.D100K.

Threshold (%)	Databases		
	T10.I6.D100K		
	HUINIV-Mine	MEU	High utility itemsets
0.2	16,304	499,500	335
0.3	5336	499,500	197
0.4	2469	499,500	130
0.6	1056	499,500	69
0.8	755	499,500	29
1	644	499,500	19

Table 7
The number of candidate itemsets and high utility itemsets generated from database T20.I4.D100K.

Threshold (%)	Databases		
	T20.I4.D100K		
	HUINIV-Mine	MEU	High utility itemsets
0.2	20,026	499,500	118
0.3	7958	499,500	55
0.4	3754	499,500	29
0.6	1308	499,500	8
0.8	774	499,500	4
1	599	499,500	2

Table 8
The number of candidate itemsets and high utility itemsets generated from database T20.I6.D100K.

Threshold (%)	Databases		
	T20.I6.D100K		
	HUINIV-Mine	MEU	High utility itemsets
0.2	27,357	499,500	127
0.3	8441	499,500	56
0.4	4095	499,500	23
0.6	1438	499,500	7
0.8	823	499,500	4
1	637	499,500	3

We obtain similar experimental results from different datasets. For example, only 644 candidate itemsets are generated by *HUINIV-Mine*, but 499,500 candidate itemsets are generated by MEU, respectively, when the utility threshold is set as 1% in dataset T10.I4.D100K. In the case of datasets T20.I4.D100K and T20.I6.D100K, more candidates are generated, because each transaction is longer than those in T10.I4.D100K and T10.I4.D100K. Overall, our algorithm *HUINIV-Mine* always generates far fewer candidates when compared to MEU for various kinds of databases. Thus, *HUINIV-Mine* is verified to be very effective in pruning candidate itemsets to find high utility itemsets with negative item values.

It is observed that *HUINIV-Mine* obtains fewer candidate itemsets than MEU with high stability with regard to finding high utility itemsets with negative item values. To measure how many candidate itemsets could be reduced substantially by using *HUINIV-Mine* compared to MEU algorithm, we define the *Improvement Ratio* as follows:

$$\text{Improvement Ratio} = \frac{(\text{candidate itemsets of MEU}) - (\text{candidate itemsets of HUIVP-Mine})}{\text{candidate itemsets of MEU}}$$

From the data illustrated in Table 5, we see that the Improvement Ratio is about 99.8% with the threshold set as 1%. In Table 8, the average improvement is about 99.2% with the minimum utility threshold varied from 0.2% to 1%. Obviously, *HUINIV-Mine* reduces substantially the candidate itemsets for finding high utility itemsets with negative item values. Moreover, the high utility itemsets obtained by MEU are not suitable for applications in large database since MEU requires more database scans, and increased execution times and candidate itemsets to find high utility itemsets with negative item values. Thus, *HUINIV-Mine* meets the requirements of being highly effective in terms of candidate itemsets for large database mining.

4.2. Evaluation of execution time

In this experiment, we show only the performance of *HUINIV-Mine* since MEU requires much higher execution time (longer than 10 h) to complete the second scan lacks basis for comparison because the number of candidate itemsets generated is

always 499,500. Therefore, *HUINIV-Mine* meets the requirements of efficiency in terms of execution time for large database mining.

Figs. 4 and 5 show the execution times for*HUINIV-Mine* as the minimum utility threshold is decreased from 1% to 0.2%. It is observed that when the minimum utility threshold is high, there are only a limited number of high utility itemsets produced. However, as the minimum utility threshold decreases, the execution times increase with more high utility itemsets produced. As shown in Figs. 4 and 5, the margin grows as the minimum utility threshold increases for different average sizes of transaction.

4.3. Scale-up on incremental mining

In this experiment, we investigate the effects of varying incremental transaction size $|d|$ on the scalability of proposed scheme in terms of execution time. To further understand the impact of $|d|$ on the performance of *HUINIV-Mine*, we conducted scale-up experiments with minimum support threshold set as 0.2% and 0.4%, respectively. Fig. 6 shows the experimental results, where the values in *y*-axis correspond to the execution time of *HUINIV-Mine* under different values of $|d|$.

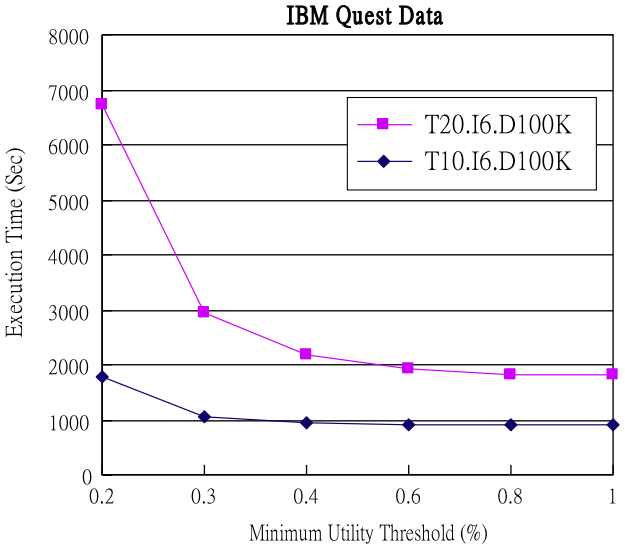

Fig. 4. Execution time for HUINIV on T20.I6.D100K and T10.I6.D100K.

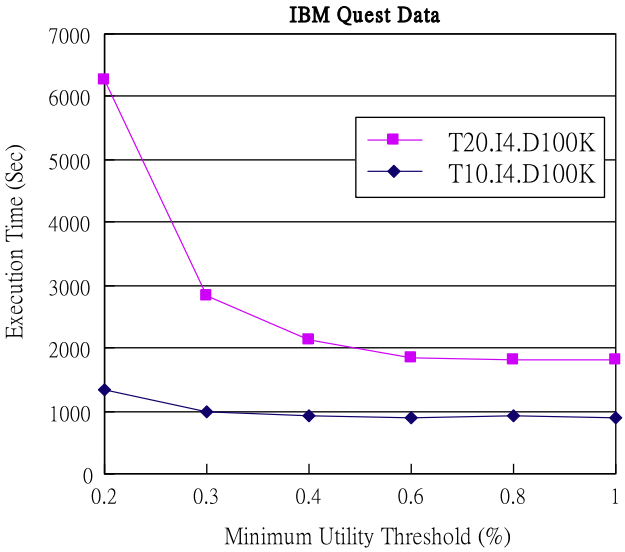

Fig. 5. Execution time for HUINIV on T20.I4.D100K and T10.I4.D100K.

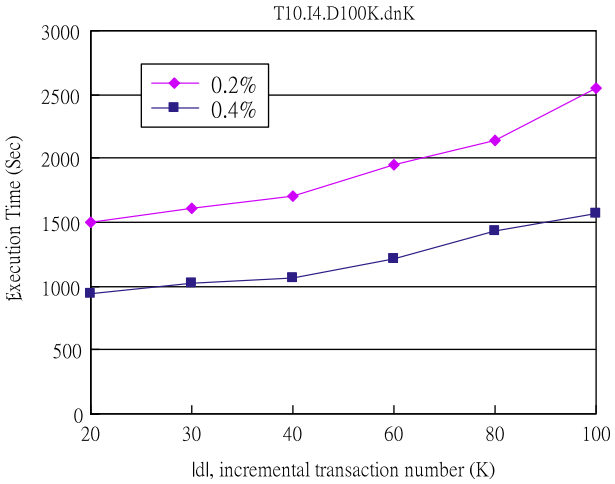

Fig. 6. Scale-up performance results for HUINIV on T10.I4.D100KdnK.

Although it can be seen that the execution time increases with the growth of the incremental transaction number $|d|$, *HUINIV-Mine* can still find high utility itemsets with negative item values with at most 4 database scans. In comparison, MEU requires 1000 database scans under various incremental transactions. This result also indicates that *HUINIV-Mine* is scalable for mining database with incremental transactions.

4.4. Evaluation with real data

We also evaluate our algorithm *HUINIV-Mine* with real data, BMS-POS. The BMS-POS dataset contains several years worth of point-of-sale data from a large electronics retailer. Since this retailer has so many different products, we used product categories as items. Each item thus represents a category, rather than an individual product. The transaction in this dataset is a customer's purchase transaction consisting of all product categories purchased at one time. The goal of this dataset is to find associations between product categories purchased by customers in a single visit to the retailer. Table 9 characterizes BMS-POS in terms of the number of transactions, the number of distinct items, the maximum transaction size, and the average transaction size.

This data set was used in the KDD-Cup 2000 competition and was recently made publicly available by Blue Martini Software (downloaded from http://www.ecn.purdue.edu/KDDCUP). In order to render databases suitable for utility mining, we also randomly generate the quantity of each item in each transaction, ranging from 1 to 5. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000.

Table 10 shows the average number of candidates generated by *HUINIV-Mine* and MEU. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The number of candidate itemsets generated by *HUINIV-*

Table 9
Database BMS-POS characteristics.

Transactions	Distinct items	Maximum transaction size	Average transaction size
BMS-POS			
515,597	1657	164	6.5

Table 10
The number of candidate itemsets and high utility itemsets generated on database BMS-POS.

Threshold (%)	Databases		
	BMS-POS		
	HUINIV-Mine	MEU	High utility itemsets
0.2	59,066	499,500	151
0.3	31,485	499,500	66
0.4	19,488	499,500	34
0.6	9603	499,500	16
0.8	5728	499,500	7
1	3789	499,500	4

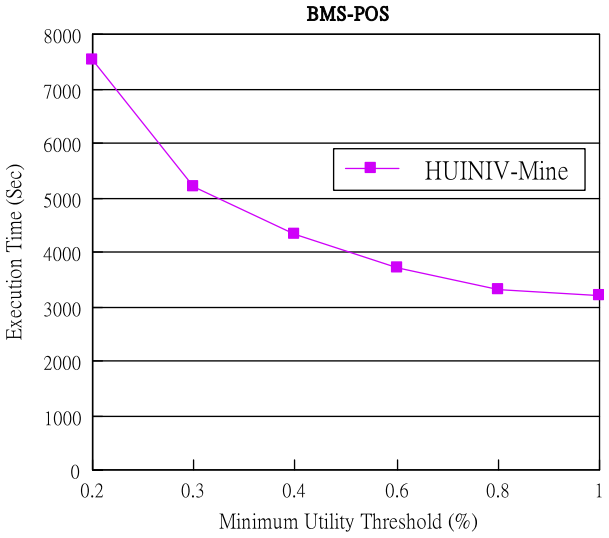

Fig. 7. Execution time for HUINIV on BMS-POS.

Mine during the first database scan decreases dramatically as the threshold increases. Particularly when the utility threshold is set to 1%, the number of candidate itemsets is generally 3789, including all various candidate itemsets within the database BMS-POS. However, the number of candidates generated by MEU is always 499,500 because it must process all combinations of 1000 items to generate only 2-candidate itemsets. It is observed that*HUINIV-Mine* still generates far fewer candidates when compared to MEU even using the real data. Hence, this result indicates that *HUINIV-Mine* is useful for mining high utility itemsets with negative item values from both artificial data and real data. Fig. 7 shows the execution times for *HUINIV-Mine* as the minimum utility threshold is decreased from 1% to 0.2%.

5. Conclusions

In this paper, we addressed the problem of discovering high utility itemsets with negative item values in large databases, i.e., the itemsets containing negative item values that are larger than threshold in large databases. We propose a new approach, namely *HUINIV-Mine*, which can identify high utility itemsets with negative item values in large databases both efficiently and effectively. The novel contribution of *HUINIV-Mine* is that it can effectively identify high utility itemsets with negative item values in less high TWUI such that the execution time can be reduced efficiently for mining all high utility itemsets with negative item values in large databases. In this way, the process of discovering all high utility itemsets containing negative item values can be achieved effectively with limited memory space, less candidate itemsets and CPU I/O. This meets the critical requirements of time and space efficiency for mining high utility itemsets with negative item values.

The experimental results show that *HUINIV-Mine* can find high utility itemsets with negative item values with higher performance by generating fewer candidate itemsets compared to other algorithms under varied experimental conditions. It was found that *HUINIV-Mine* delivers an average improvement around 99.2% over MEU method in terms of execution performance. That is, the advantage of *HUINIV-Mine* over MEU is stable and less execution time is taken when the concept of negative item values is considered. Hence, *HUINIV-Mine* is promising for mining high utility itemsets in large databases with negative item values. In the future, we would explore to extend the concepts proposed in this work for discovering high utility itemsets with negative item values in temporal databases [6,10,11,21] or data streams [15,18,19].

Acknowledgement

This research was supported by National Science Council, Taiwan, ROC under Grant No. NSC 97-2631-H-006-001 and NSC 96-2221-E-006-143-MY3.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in Proceedings of 1993 ACM SIGMOD Intl. Conf. on Management of Data, Washington, DC, May 1993, pp. 207–216.
[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo, Fast discovery of association rules, in: Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996, 1996, pp. 307–328.
[3] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the 11th International Conference on Data Engineering, March 1995, pp. 3–14.
[4] N.F. Ayn, A.U. Tansel, E. Arun, An efficient algorithm to update large itemsets with early pruning, Technical Report BU-CEIS-9908, Department of CEIS Bilkent University, June 1999.

[5] N.F. Ayn, A.U. Tansel, E. Arun, An efficient algorithm to update large itemsets with early pruning, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, August 1999.
[6] C. Bettini, X.S. Wang, S. Jajodia, Testing complex temporal relationships involving multiple granularities and its application to data mining, in: Proc. of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3–5, 1996, Montreal, Canada, ACM Press, 1996, pp. 68–78.
[7] R. Chan, Q. Yang, Y. Shen, Mining high utility itemsets, in: Proceedings of IEEE ICDM, Florida, 2003.
[8] D. Cheung, J. Han, V. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating technique, in: Proceedings of 1996 International Conference on Data Engineering, February 1996, pp. 106–114.
[9] D. Cheung, S.D. Lee, B. Kao, A general incremental technique for updating discovered association rules, in: Proceedings of the International Conference on Database Systems For Advanced Applications, April 1997.
[10] Y. Chi, H. Wang, P.S. Yu, R. Richard, Muntz: moment: maintaining closed frequent itemsets over a stream sliding window, in: Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM'04).
[11] G. Das, K.I. Lin, H. Mannila, G. Renganathan, P. Smyth, Rule discovery from time series, in: Proceedings of the 4th ACM SIGKDD, August 1998, pp. 16–22.
[12] C.H. Lee, C.R. Lin, M.S. Chen, Sliding-window filtering: an efficient algorithm for incremental mining, in: International Conference on Information and Knowledge Management (CIKM01), November 2001, pp. 263–270.
[13] J.L. Lin, M.H. Dunham, Mining association rules: anti-skew algorithms, in: Proceedings of 1998 International Conference on Data Engineering, 1998, pp. 486–493.
[14] Y. Liu, W. Liao, A.A Choudhary, Fast high utility itemsets mining algorithm, in: Proceedings of the Utility-Based Data Mining Workshop, August 2005.
[15] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, in: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002.
[16] J.S. Park, M.S. Chen, P.S. Yu, Using a hash-based method with transaction trimming for mining association rules, IEEE Transactions on Knowledge and Data Engineering 9 (5) (1997) 813–825.
[17] A. Savasere, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, in: Proceedings of the 21th International Conference on Very Large Data Bases, September 1995, pp. 432–444.
[18] W.G. Teng, M.S. Chen, P.S. Yu, A regression-based temporal pattern mining scheme for data streams, in: Proceedings of the 29th International Conference on Very Large Data Bases, September 2003, pp. 93–104.
[19] W.G. Teng, M.S. Chen, P.S. Yu, Resource-aware mining with variable granularities in data streams, in: Proceedings of the 4th SIAM International Conference on Data Mining, Florida, USA, 2004.
[20] V.S. Tseng, C.J. Chu, T. Liang, Efficient mining of temporal high utility itemsets from data streams, in: Proceedings of ACM KDD Workshop Utility-Based Data Mining (UBDM'06), Philadelphia, Pennsylvania, USA, August, 2006.
[21] V.S. Tseng, C.J. Chu, T. Liang, An efficient method for mining temporal emerging itemsets from data streams, in: International Computer Symposium (ICS), Workshop on Software Engineering, Databases and Knowledge Discovery, Taipei, Taiwan, December 2006.
[22] H. Yao, H.J. Hamilton, C.J. Butz, A foundational approach to mining itemset utilities from databases, in: Proceedings of the 4th SIAM International Conference on Data Mining, Florida, USA, 2004.