Efficient Vertical Mining of High Utility Quantitative Itemsets

Chia Hua Li ,Cheng-Wei Wu, Vincent S. Tseng
Departement of Computer Science and Information Engineering,
National Cheng Kung University, Taiwan, ROC
brandnew0426@idb.csie.ncku.edu.tw, silvemoonfox@hotmail.com, tsengsm@mail.ncku.edu.tw

***Abstract*—High utility quantitative itemset mining refers to discovering sets of items that carry not only high utilities (e.g., high profits) but also quantitative attributes. Although this topic is very important to many applications, it has not been deeply explored and existing algorithms for mining high utility quantitative itemsets remain computationally expensive. To address this problem, we propose a novel algorithm named VHUQI (Vertical mining of High Utility Quantitative Itemsets) for efficiently mining high utility quantitative itemsets in databases. VHUQI adopts a vertical representation to maintain the utility information of itemsets in databases with several effective strategies integrated to prune the search space. The experimental results on both real and synthetic datasets show that VHUQI outperforms the state-of-the-art algorithms substantially in terms of both execution time and memory consumption.**

Keywords—Quantitative itemset mining; high utility itemset mining; high utility quantitative itemset mining; utility mining;

I. INTRODUCTION

Discovering interesting *high utility patterns* in different kinds of databases is an important task in *utility mining*. Extensive studies have been proposed for mining various types of high utility patterns, such as *high utility itemsets* [3, 5, 6, 7, 8, 9, 10, 11, 15], *top-k high utility itemsets* [19], *closed*$^+$ *high utility itemsets* [18] and *high utility episodes* [17]. *High utility itemset mining* is probably the most popular research topic among them. In contrast to traditional frequent itemset mining, the importance of items and quantity attribute of items are considered in the framework of high utility itemset mining. Therefore, it can be used to discover itemsets carrying with high utilities (e.g., high profits). Although extensive studies have been proposed for high utility itemset mining, a critical limitation of these studies is that they ignore the quantity attribute of items in discovered high utility itemsets. However, such information can be very useful and valuable in many applications.

In view of this, the concept of *high utility quantitative itemset* (abbreviated as *HUQI*) *mining* is emerged [16, 21]. In the framework of HUQI mining, an item may have different quantities in the database and each item carrying with different quantity is regarded as *quantitative item* (abbreviated as *q-items*) [13, 14]. An itemset consisting of q-items is called *q-itemset*. A q-itemset is called *high utility quantitative itemset* if its utility is no less than a user-specified minimum utility threshold. Otherwise, it is called *low utility quantitative itemset.*

Mining HUQIs is not an easy task and faces the following challenges. First, the search space of HUQI mining cannot be effective pruned as it is done in frequent pattern mining because the *downward closure property* [1, 4, 12] does not hold for the utility of q-itemsets. Second, it may pose the large search space problem especially when database contains a large amount of q-items. The combination explosion of q-items causes the mining task to suffer from long execution time and huge memory consumption. To generate more useful information, quantities of some items in q-itemsets need to be combined to some ranges. Therefore, the third challenge is to develop an appropriate combining method that allows to generating useful information and reducing the computational costs.

To mine high utility quantitative itemsets from databases, Yen et al. proposed an algorithm named *HUQA* [21]. Because a superset of a low utility q-itemset can be high utility, they propose the concept of *weak utility q-itemset* (abbreviated as *WUQI*) to facilitate the mining process. Weak utility q-itemsets are those q-itemsets that can be extend to generate HUQIs by joining with the some q-items. HUQA is a recursive algorithm consist of two main steps. In the first step, it constructs conditional database for each HUQI and WUQI of length k. In the second step, HUQI and WUQI of length (k+1) are generated from the constructed conditional databases. Although HUQA is the first algorithm for mining HUQIs, it may produce a large number of conditional databases during the mining process, which degrades the execution time and occupies lots of memory space.

To address the issue mentioned above, we propose in this paper a vertical mining algorithm named *VHUQI* (*Vertical mining of High Utility Quantitative Itemsets*) for efficiently mining the complete set of high utility quantitative itemsets in the database. It converts a horizontal database into a special vertical representation named *utility-list* to facilitate the mining process. In VHUQI, each q-itemset is associated with a utility-list, which indicates the utility information of the q-itemset in the database. The main merit of the utility-list structure integrated in VHUQI is that it allows to efficiently generate HUQIs and their utilities in memory without scanning the original database. Experimental results on both real and synthetic datasets show that VHUQI outperforms the state-of-the-art algorithm HUQA in both execution time and space. Particularly, VHUQI is faster than HUQA over two orders of magnitude.

The rest of the paper is organized as follows. Section II introduces the background of HUQI mining. Section III describes the proposed VHUQI algorithm. Section IV presents the experimental study. Section V draws the conclusion.

978-1-4799-5464-3/14/$31.00 ©2014 IEEE

II. BACKGROUND

In this section, we define the problem of HUQI mining and then introduce related studies of HUQI mining.

A. Problem Definition

Let $I^* = \{I_1, I_2, \ldots, I_N\}$ be a finite set of *items*. Each item $I \in I^*$ has a positive number $p(I)$ called its *external utility* (e.g., unit profit). A *database* $D = \{T_1, T_2, \ldots, T_M\}$ is a set of transactions, where each transaction $T_d \in D$, $(1 \le d \le M)$ is a subset of I^* and has an unique identifier d, called *TID*. Each item I in the transaction T_d has an positive internal utility $q(I, T_d)$ (e.g., quantity). A *quantitative item x* (abbreviated as *q-item*) is a triple (I, L, U), where I is an item, L and U are respectively lower bound and upper bound of it quantity. If $L = U$, x is denoted as (I, L). A *quantitative itemset* (abbreviated as *q-itemset*) X is a set of k distinct items $\{I_1, I_2, \ldots, I_k\}$, where $I_j \in I^*$, $1 \le j \le k$, and k is the length of X. A q-itemset of length k is called *k-q-itemset.* Let Table I be an example database of this paper. Each row in Table I represents a transaction, where each letter represents an item and has a quantity. Table II shows the external utility (e.g., unit profit) of each item.

Definition 1 (The support count of a q-item). The *support count of a q-item* $x = (I, L, U)$ is denoted as $SC(x)$ and defined as $\sum_{j=L}^{U} SC(I, j)$.

Definition 2 (The support count of a q-itemset). The *support count of a q-itemset* $X = \{(I_1, L_1, U_1)\ (I_2, L_2, U_2) \ldots (I_k, L_k, U_k)\}$ is denoted as $SC(X)$ and defined as $\sum_{j_1=L_1}^{U_1} \ldots \sum_{j_k=L_K}^{U_k} SC(\{(I_1, L_1, U_1)(I_2, L_2, U_2) \ldots (I_K, L_K, U_K)\})$.

For example, $SC((B,4,5)) = SC((B,4)) + SC((B,5)) = 1 + 1 = 2$. $SC(\{(A,2)(B,5,6)\}) = SC(\{(A,2)(B,5)\}) + SC(\{(A,2)(B,6)\}) = 1 + 1 = 2$.

Definition 3 (The absolute utility of a q-item in a transaction). The *absolute utility of a q-item* $x = (I, L)$ in the transaction T_d is defined as $au(x, T_d) = p(I) \times q(I, T_d)$.

Definition 4. (The absolute utility of a q-itemset) The *absolute utility of a q-itemset* $X = \{x_1, x_2, \ldots, x_k\}$ in the transaction T_d is defined as $au(X, T_d) = \sum_{x \in X \wedge X \subseteq T_d} au(x, T_d)$.

For example, $au((A,2), T_1) = p(A) \times q(A, T_1) = 3 \times 2 = 6$. $au(\{(A,2)(B,5)(C,2)\}, T_1) = au((A,2), T_1) + au((B,5), T_1) + au((C,6), T_1) = 6 + 5 + 4 = 15$.

Definition 5 (Total utility of the database). The Total utility of the database D is defined as $TotalU = \sum_{T_d \in D} \sum_{x \in T_d} au(X, T_d)$.

TABLE I. AN EXAMPLE DATABASE

TID	Transaction
T_1	(A,2) (B,5) (C,2) (D,1)
T_2	(B,4) (C,3)
T_3	(A,2) (C,2)
T_4	(A,2) (B,6) (D,1)

TABLE II. PROFIT TABLE

Item	A	B	C	D
Profit	3	1	2	2

Definition 6 (The utility of a q-itemset in the database). The *absolute utility of a q-itemset X in the database D* is defined as $au(X) = \sum_{X \subseteq T_d \wedge T_d \in D} au(X, T_d)$. The *relative utility of X* (or simply called *utility of X*) is denoted as $u(X)$ and defined as $au(X)/TotalU$.

For example, $au(\{(A,2)(D,1)\}) = au(\{(A,2)(D,1)\}, T_1) + au(\{(A,2)(D,1)\}, T_4) = 6 + 6 = 12$. $au(\{(A,2)(B,5,6)(D,1)\}) = au(\{(A,2)(B,5)(D,1)\}, T_1) + au(\{(A,2)(B,6)(D,1)\}, T_4) = 27$.

Definition 7 (Combining constraint proposed by [15]). Given a *user-specified minimum absolute utility threshold min_abs_util* ($0 < min\_abs\_util \le TotalU$), a *quantitative related coefficient qrc* ($qrc \ge 1$), and two low utility q-itemsets $X = \{(I_1, L_1, U_1)\ (I_2, L_2, U_2) \ldots (I_k, L_k, U_k)\}$ and $Y = \{(I_1, L_1, U_1)\ (I_2, L_2, U_2) \ldots (I_k, L_k')\}$, if (1) $L_k' = (U_k + 1)$ and (2) $au(X)$ and $au(Y)$ are no less than (min_util / qrc), then X can be combined with Y to form a q-itemset $\{(I_1, L_1, U_1)\ (I_2, L_2, U_2) \ldots (I_k, L_k, L_k')\}$.

Definition 8 (High utility q-itemset). A q-itemset X is called *high utility q-itemset* (abbreviated as *HUQI*) iff $au(X)$ is no less than *min_abs_util*. Otherwise, X is called *low utility q-itemset*. An equivalent definition is that X is high utility iff $u(X) \ge min\_util$ ($0 < min\_util < TotalU$), where $min\_util = min\_abs\_util / TotalU$.

Problem Statement. Given a database D, a *user-specified absolute minimum utility threshold min_abs_util* and a *quantitative related coefficient qrc*, the problem of HUQI mining is to discover all q-itemsets having a utility no less than *min_abs_util*.

B. Related Work

Extensive studies have been proposed for mining high utility itemsets, including *Two-Phase* [8], *IHUP* [2], *IIDS* [11], *UP-Growth*$^+$ [15] and *HUI-Miner* [9]. The former four algorithms are two-phase algorithms. A drawback of two-phase algorithms is that they may produce too many candidates during the mining process, which seriously degrades their performance. To avoid this drawback, a novel algorithm named *HUI-Miner* was recently proposed. HUI-Miner is a one-phase algorithm, which adopts a novel vertical representation named *utility-list* to facilitate the mining process. Though the above algorithms may perform well in some applications, they are not developed for mining HUQIs. The topic of HUQI mining was first introduced by Yen et al. [21]. They proposed an algorithm named *HUQA* for mining HUQIs in horizontal databases. In their proposed framework, a combining method (Definition 7) based on the idea of *quantitative related coefficient* is proposed for generating useful HUQIs. Although *HUQA* is the first algorithm for mining HUQI, it may produce too many conditional databases during the mining process, which degrades its execution time and space requirement (e.g., disk and memory). Wang et al. proposed the concept of *fuzzy high utility quantitative itemsets* (abbreviated as *FHUQI*) and an efficient algorithm for mining FHUQIs [16]. However, FHUQIs are difficult to be interpreted or comprehended by users. From the above literature review, we can see that the topic of HUQI mining has not been deeply explored and there is ample room for improving the performance of HUQA.

{(A,2)}		
T_1	6	11
T_3	6	4
T_4	6	8

{(B,4)}		
T_2	4	0

{(B,5)}		
T_1	15	4

{(B,6)}		
T_4	6	2

{(C,1)}		
T_1	4	2
T_3	4	0

{(C,2)}		
T_3	6	4

{(D,1)}		
T_1	2	0
T_4	2	0

Tids Eutils Rutils

Figure1. Initial utility-lists

{(B,4)}		
T_2	4	0

{(B,5)}		
T_1	5	6

{(B,5)(B,6)}		
T_1	5	6
T_4	6	2

Figure 2. An example for combing two utility-lists

{(A,2)}		
T_1	6	11
T_3	6	4
T_4	6	8

{(C,2)}		
T_1	4	2
T_3	4	0

{(A,2)(C,2)}		
T_1	10	2
T_3	10	0

Figure 3. An example for constructing utility-list of 2-q-itemset

{(A,2)(C,2)}		
T_1	10	2
T_3	10	0

{(A,2)(D,1)}		
T_1	8	0
T_4	8	0

{(A,2)(C,2)(D,1)}		
T_1	12	0

Figure 4. An example for constructing utility-list of *k*-q-itemset

III. THE PROPOSED METHOD

In this section, we first introduce utility-list structure [9] and then present the proposed *VHUQI* (*Vertical mining of High Utility Quantitative Itemsets*) algorithm.

A. Utility-list Structure

In this subsection, we introduce the utility-list structure and its related properties. For more details about the utility-list, readers can refer to [9]. In VHUQI, each q-item(set) X is associated with a utility-list denoted as $UL(X)$. The utility-lists of q-items are also called initial *utility-lists*. The initial utility-lists can be constructed by scanning the original database twice. In the first database scan, the absolute utility of each q-item is calculated. In the second database scan, q-items in each transaction are sorted in descending order of absolute utility. Each transaction containing the q-item x is converted to a tuple of $UL(x)$. After scanning all transactions in the original database, the initial utility-lists are constructed.

Figure 1 shows initial utility-lists for the database of Table 1. The utility-list of a q-item(set) X is composed of several tuples. Each tuple represents the utility information of X in the transaction T_d and has three fields: *Tid*, *Eutil* and *Rutil*. The *Tid* field indicates the *Tid* of the transaction T_d. The *Eutil* field indicates the absolute utility of X in T_d, which is denoted as $EU(X, T_d)$. In the utility-list of X, the sum of all its *Eutil* values is the absolute utility of X in the database. The sum of its *Eutil* values is denoted as $SumEutil(X)$. The *Rutil* field indicates the *remaining utility* of X in T_d [9]. The remaining utility of X in T_d is denoted as $RU(X, T_d)$ and defined as the sum of the absolute utility of each q-item after the last q-item of X. The remaining utility of X will be used later for pruning low utility q-itemsets.

B. The VHUQI Algorithm

In this subsection, we present the VHUQI algorithm. VHUQI takes as input a database D with internal and external utilities, a minimum absolute utility threshold *min_abs_util*, and a quantitative related coefficient *qrc* and it outputs the complete set of HQUIs in D. If a horizontal database has been converted into initial utility-lists, VHUQI can directly use it for mining QHUIs. VHUQI is performed as follows. First, the algorithm scans D twice to construct initial utility-lists $\{\varnothing\}$-*ULs*. All q-items are collected into an ordered list named *QList*, which is denoted as $\{\varnothing\}$-*QL*. In *QList*, q-items are stored in descending order of their absolute utilities. Then, the algorithm calls the HUQI-Search procedure with four input parameters $\{\varnothing\}$-*ULs*, $\{\varnothing\}$-*QL*, *min_abs_util* and *qrc* for searching HUQIs.

PROCEDURE: HUQI-Search

Input:
(1) $\{P\}$-*ULs*: utility-lists of q-itemsets w.r.t. prefix $\{P\}$;
(2) $\{P\}$-*QL*: a list of q-itemsets w.r.t prefix $\{P\}$;
(3) *min_abs_util*: minimum absolute utility threshold;
(4) *qrc*: related quantitative coefficient;
Output: the complete set of high utility quantitative itemsets;

```
01 For each X ∈ {P}-QL do
02 {   If(UL(X).SumEutils ≥ min_abs_util)
03     {   H := H ∪ X and Output X;   }
04     else
05     {   If (UL(X).SC) ≥ K-SupportBound(X))
06         {   W := W ∪ X;   }
07         If(UL(X).SumEutils ≥ (min_abs_util / rqc))
08         {   C := C ∪ X;   }
09     }
10 }
11 (HC, CULs) := Combine(C, {P}-ULs, {P}-QL);
12 JoinList := Sort(H ∪ W ∪ HC);
13 {P}-ULs := {P}-ULs ∪ CULs;
14 For each X = {x1, x2, …, xk= (Ik, Lk,Uk)}∈ JoinList do
15 {   Set {X}-ULs := ∅ and {X}-QL := ∅;
16     For each Y = {y1, y2, …, yk= (Ik', Lk')} ∈ JoinList
17     such that Ik = Ik' and Y≺ X do
18     {   Z = X ∪Y;
19         Construct UL(Z);
20         If (UL(Z) ≠ ∅)
21         {   {X}-ULs := {X}-ULs ∪ UL(Z);
22             {X}-QL := {X}-QL ∪ Z;
23         }
24     }
25     HUQI-Search({X}-ULs, {X}-QL, min_abs_utility, qrc);
26 }
```

Pseudo Code 1. The HUQI-Search procedure

C. The HUQI-Search Procedure

The pseudo code of the HUQI-Search procedure is shown in Pseudo Code 1, which is performed as follows. The procedure uses recursive depth-first exploration to discover HUQIs. It takes as input a set of utility-lists w.r.t. prefix P (denoted as $\{P\}$-*ULs*), a *QList* w.r.t. prefix P (denoted as $\{P\}$-*QL*), a minimum absolute utility threshold *min_abs_util*, and a quantitative related coefficient *qrc* and outputs the complete set of HQUIs in D.

For each q-itemset X in $\{P\}$-*QL*, if its absolute utility is no less than *min_abs_util*, the algorithm outputs X as a HUQI and collects X into a set H (line 2-3). The set H is used to store HQUIs whose prefixes are P. Otherwise, X is a low utility q-itemset. If X is low utility and its support count is no less than its *K-SupportBound* [21], X is a weak utility q-itemset (WUQI) (line 5-6). A WUQI is a low utility q-itemset but its superset may be high utility. The *K-support bound* of X is denoted as $KSB(X)$ and defined as:

$$\left\lceil \frac{min\_abs\_util \times TotalU}{Eutil(X)+Max(Rutil(X))} \right\rceil,$$

where $Max(Rutil(X))$ means the maximum remaining utility of X in database D. Because $Max(Rutil(X))$ can be easily obtained from $UL(X)$, $KSB(X)$ can be easily calculated.

If the absolute utility of X is no less than *min_abs_util/qrc*, the algorithm adds X to a set C (line 7-8). The set C is used to store q-itemsets whose absolute utilities are no less than *min_abs_util/qrc*. After processing each itemset in $\{P\}$-*UL* (line 10), the algorithm calls the *Combine* procedure with three input parameters C, $\{\varnothing\}$-*ULs* and $\{\varnothing\}$-*QL* (line 11). The *Combine* procedure applies Definition 7 to combine low utility q-itemset in C and returns *HC* and *CULs*, where *HC* is the set of HUQIs that are generated by combining q-itemsets in C and *CULs* is the set of utility-lists of HUQIs in *HC*. Then, the algorithm collects all q-itemsets in sets H, W and *HC* and put these q-itemsets in an ordered list called *JoinList* (line 12). The utility-lists *CULs* is added to the $\{P\}$-*UL* (line 13).

For each q-itemset $X = \{x_1, x_2, \ldots, x_k\}$ in *JoinList*, the algorithm performs as follows (line 14-26). First, $\{X\}$-*ULs* and $\{X\}$-*QL* are initialized to $\varnothing$ (line 15), where $\{X\}$-*ULs* is used to store utility-lists of q-itemsets w.r.t. prefix X and $\{X\}$-*QL* is used to store q-itemsets whose prefixes are X. Suppose that the last q-items respectively in X and Y are $x_k = (I_k, L_k, U_k)$ and $y_k = (I_k', L_k', U_k')$. Then, for each q-itemset $Y = \{y_1, y_2, \ldots, y_k\}$ appearing after X in *JoinList* (line 14-17), if $I_k = I_k'$ the algorithm generates a q-itemset Z by joining X and Y and constructs its utility-list $UL(Z)$ (line 18-19). If $UL(Z)$ is not empty, $UL(Z)$ and Z are respectively added to $UL(Z)$ and $\{X\}$-*QL* (line 20-23). After traversing each q-itemset appearing after X, the procedure HUI-Search is recursive called for exploring the search space of HUQIs whose prefixes are X (line 25). The algorithm continues searching for other HUQIs until no HUQIs is found.

D. The Combine Procedure

The pseudo code of the Combine procedure is shown in Pseudo Code 2, which is performed as follows. It takes as input a list of q-itemsets storing in C (denoted as $\{P\}$-*CQL*), a set of utility-lists of q-itemsets w.r.t. prefix P (denoted as $\{P\}$-*ULs*), a *QList* consisting of q-itemsets w.r.t prefix P (denoted as $\{P\}$-*QL*). Initially, two variables $\{P\}$-*HC* and $\{P\}$-*CUL* are set to $\varnothing$ (line 1). For each q-itemset $X = \{p_1, p_2, \ldots, p_{k-1}, (xI_k, xL_k, xU_k)\}$ in $\{P\}$-*CQL*, the algorithm performs as follows (line 2-20). First, a q-itemset Z and its $UL(Z)$ are initialized to $\varnothing$ (line 3). Then, the algorithm traverses each q-itemset appearing after X in $\{P\}$-*CQL* (line 4-19). For each traversed q-itemset $Y = \{p_1, p_2, \ldots, p_{k-1}, (yI_k, yL_k,)\}$, if (1) $xI_k = yI_k$, (2) $xU_k+1 = yL_k$ and (3) $Z = \varnothing$, the algorithm generates a q-itemset $Z = \{p_1, p_2, \ldots, p_{k-1}, (xI_k, xL_k, yL_k)\}$ by combining X and Y (line 6-7). Besides, the algorithm merges utility-lists of X and Y to form the utility-list of Z (i.e.,$UL(Z)$) (line 8). Otherwise, if $Z = \{p_1, p_2, \ldots, p_{k-1}, (zI_k, zL_k, zU_k)$, $yI_k=zI_k$ and $yL_k = zU_k + 1$, Z is combined with Y and $UL(Z)$ is merged with $UL(Y)$ (line 9-14). After that, if the absolute utility of Z is no less than *min_abs_util*, Z is added to $\{P\}$-*HC* and $UL(Z)$ is added to $\{P\}$-*CUL* (line 15-18). After processing all q-itemsets in $\{P\}$-*CQL*, the algorithm returns $\{P\}$-*HC* and $\{P\}$-*CUL* (line 20).

PROCEDURE: Combine

Input:
(1) $\{P\}$-*CQL*: a list of q-itemsets w.r.t prefix $\{P\}$;
(2) $\{P\}$-*ULs*: utility-lists of q-itemsets w.r.t. prefix $\{P\}$;
(3) $\{P\}$-*QL*: a list of q-itemsets w.r.t prefix $\{P\}$;
Return:
(1) $\{P\}$-*HC*: a set of high utility q-itemsets w.r.t. prefix $\{P\}$;
(2) $\{P\}$-*CUL*: a set of utility-lists w.r.t. prefix $\{P\}$;

```
01 Set {P}-HC := ∅; {P}-CUL := ∅;
02 For each X = {p1, p2, …, pk-1, (xIk, xLk, xUk)} ∈ {P}-CQL do
03 {   Set Z := ∅; UL(Z) := ∅;
04     For each Y = {p1, p2, …, pk-1, (yIk, yLk)} ∈ {P}-CQL
05     and Y ≺ X do
06     {   If(Z == ∅ and xIk == yIk and xUk + 1 = yLk)
07         {   Z := { p1, p2, …, pk-1, (xIk, xLk, yLk)};
08             UL(Z) := MergeUtilityList(UL(X), UL(Y));   }
09         else
10         {   Suppose Z = {p1, p2, …, pk-1, (zIk, zLk, zUk);
11             If(zIk== yIk and zUk + 1 = yLk)
12             {   Z := {p1, p2, …, pk-1, (zIk, zLk, yLk)};
13                 UL(Z) := MergeUtilityList(UL(Z),UL(Y));    }
14         }
15         If(UL(Z).SumEutils ≥ min_abs_util)
16         {   {P}-HC := {P}-HC ∪ Z;
17             {P}-CUL := {P}-CUL ∪ UL(Z);
18             break;                          }
19     }
20 } Return {P}-HC and {P}-CUL;
```

Pseudo Code 2. The Combine procedure

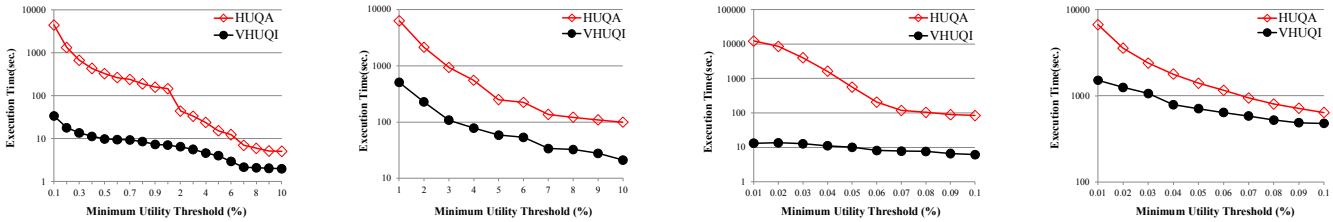
E. Constructing Utilit-lists of Q-itemsets

Given two itemsets X and Y and their prefix P, the utility-list of the itemset $Z = X \cup Y$ can be constructed as follows.

Constructing utility-lists of 2-q-itemsets. Let $X = -x_1''$ and $Y = -y_1''$ be two q-items, where x_1 appears after y_1 in $\{\varnothing\}$-*QL*. Let $Z = X \cup Y = -x_1, y_1''$ be a 2-q-itemset obtained by concatenating X with y_1. The utility-list of Z is obtained by the following process. For each transaction T_d containing both x_1 and y_1, a tuple ¡T_d, $EU(Z, T_d)$, $RU(Z, T_d)$¿ is created in $UL(Z)$, where $EU(Z, T_d)$ is the sum of the *Eutil* values in tuples associated with T_d in $UL(X)$ and $UL(Y)$, and where $RU(Z, T_d)$ is *Rutil* value associated with T_d in $UL(Y)$. That is, $EU(Z, T_d) = EU(x_1, T_d) + EU(y_1, T_d)$ and $RU(Z, T_d) = EU(y_1, T_d)$. Figure 3 shows an example for constructing utility-lists of 2-q-itemsets.

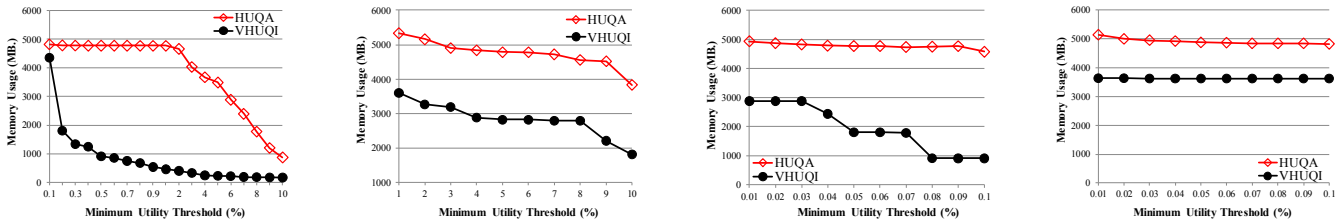
Constructing utility-lists of k-q-itemsets (k≥3). Let $X = -x_1, x_2, \square, x_{k-1}''$ and $Y = -y_1, y_2, \square, y_{k-1}''$ be two (k-1)-q-itemsets, where $x_i = y_i$ ($1 \le i \le k$-1) and y_{k-1} appears after x_{k-1}. Let $Z = X \cup Y = -x_1, x_2, \square, x_{k-1}, y_{k-1}''$ be a k-q-itemset obtained by concatenating X with y_{k-1}. Let $P = X \cap Y$ be the common prefix of X and Y. Given the utility-lists $UL(X)$, $UL(Y)$ and $UL(P)$, the utility-list of Z is obtained by the following process. For each transaction T_d containing both X and Y, a tuple ¡T_d, $EU(Z, T_d)$, $RU(Z, T_d)$¿ is created in $UL(Z)$, where $EU(Z, T_d)$ is the sum of the *Eutil* values in tuples associated with T_d in $UL(X)$ and $UL(Y)$ minus the *Eutil* value of the tuple associated with T_d in $UL(P)$, and where $RU(Z, T_d)$ is the *Rutil* value associated with Td in $UL(Y)$. That is, $EU(Z, T_d) = EU(X, T_d) + EU(Y, T_d) \square EU(P, T_d)$ and $RU(Z, T_d) = EU(Y, T_d)$. Figure 4 shows

an example for constructing utility-lists of k-q-itemsets ($k \geq 3$).



(a) Mushroom (b) Connect (c) Foodmart (d) Retail

Figure 5. Execution time on different types of datasets



(a) Mushroom (b) Connect (c) Foodmart (d) Retail

Figure 6. Memory consumption on different types of datasets

IV. Experimental Results

In this section, we evaluate the performance of the proposed algorithm VHUQI with the state-of-the-art algorithm HUQA [15]. The input parameters and output results of VHUQI are the same as that of HUQA. Both algorithms were implemented in Java. Experiments were performed on a desktop computer with an Intel® Core 2 Quad Core Processor @ 2.66 GHz running Windows XP and 2 GB of RAM. Different types of datasets were used to evaluate the performance of the algorithms. Mushroom, Connect and Retail datasets were real-life datasets obtained from FIMI Repository. Foodmart is a real-life dataset obtained from the Microsoft Foodmart 2000 database, which contains real external and internal utilities. For Mushroom, Connect and Retail datasets, the quantity of each item is randomly generated from 1 to 5 and the external utility of each item is randomly generated from 0.01 to 10.00. The external utility follows a log normal distribution. Table III shows the characteristics of the above datasets. Table IV shows the descriptions of parameters in Table III. In all experiments, the quantitative related coefficient is set to 3.

TABLE III. Characteristics of the Datasets

Dataset	D	T	N	Q	NQ	Type
Mushroom	8,124	23	119	5	585	Dense
Connect	67,557	43	130	5	639	Dense
Foodmart	4,141	4.4	1,559	5	5,305	Sparse
Retail	88,162	10.3	1,6470	5	61,191	Sparse

TABLE IV. Parameter Descriptions

Parameter Descriptions	
D	Total number of transactions
T	Average length per transaction
N	Number of distinct items
Q	Maximum number of purchased items in transactions
NQ	Number of distinct q-items

A. Experiments on Dense Datasets

The execution times of the algorithms on Mushroom and Connect datasets are respectively show in Figure 5(a) and Figure 5(b). Mushroom and Connect are dense datasets. A characteristic of dense dataset is that it consists of long transactions and contains a large number of HUQIs even when the minimum utility threshold is high. In Figure 5(a) and Figure 5(b), the execution time of HUQA is slower than VHUQI about 10 times. The reason is that both Mushroom and Connect datasets contain a large amount of HUQIs, which leads HUQA to generating too many conditional databases for HUQIs and intermediate weak utility q-itemsets. Generating conditional database of q-itemset takes lots of time because it involves expensive I/O operation. In contrast to HUQA, HUI-Miner uses utility-list to directly calculate the utility of HUQIs and WUQIs in memory, which is more efficient than generating conditional databases. For example, when *min_abs_util* = 1%, HUQA takes 6,692 seconds, while VHUQI only takes 506 seconds.

B. Experiments on Sparse Datasets

The execution time of the algorithms on Foodmart dataset is shown in Figure 5(c). The minimum utility threshold for this experiment is varied from 0.1% to 0.01%. The Foodmart dataset is a real-life spare dataset. When the minimum utility threshold is high (e.g., *min_abs_util* = 0.1%), both HUQA and HUQI achieves good performance. When the threshold decreases, the number of HUQIs dramatically increases due to the combination explosion of q-items. In this scenario, HUQA suffers from very long execution time. For example, when *min_abs_util* = 0.01%, HUQA takes 12,293 seconds, while VHUQI only takes 15 seconds. The performance of VHUQI is over 800 times faster than HUQA. From this experimental result, we can observe that VHUQI outperforms HUQA substantially, particularly for databases containing many q-items.

C. Experiments on Large Amount of Q-items

The execution time of the algorithms on Retail dataset is shown in Figure 5(d). The minimum utility threshold for this experiment is varied from 0.1% to 0.01%. The Retail dataset is a sparse dataset consisting of a large amount of q-items. When the minimum utility threshold is higher than 0.1%, the performance of HUQA and HUQI is similar. When the threshold decreases, the number of HUQIs increases and thus the more execution time is required for both algorithms. However, VHUQI remains runs faster than HUQA. The reason why the performance gap between HUQA and VHUQI is smaller for sparse dataset than for dense dataset is because sparse dataset typically contains much fewer q-itemsets than dense dataset.

D. Memory Usage

We measure the maximum memory usage of HUQA and VHUQI by using the Java API. Figure 6 shows the memory consumption of the algorithms on different types of datasets. In general, HUQA uses much more memory than VHUQI because the former needs to construct a huge size of horizontal conditional databases, while the latter uses vertical representation utility-lists.

V. CONCLUSION

In this paper, we have presented a novel high utility quantitative itemset mining algorithm named *VHUQI* (*Vertical mining of High Utility Quantitative Itemsets*). It adopts the idea of utility-list to directly calculate the utility of q-itemsets in memory and uses k-support bound estimation to effectively prune the search space. An experimental study on both synthetic and real datasets shows that VHUQI is up to three orders of magnitude faster than HUQA, the state-of-art algorithm for high utility quantitative itemset mining. Meanwhile, it consumes less memory than HUQA as well. Particularly, VHUQI outperforms HUQA substantially when the database contains a large amount of q-items.

ACKNOWLEDGEMENT

This research was partially supported by Ministry of Science and Technology, Taiwan, R.O.C. under grant no. 101-2221-E-006-255-MY3.

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, September 12-15, 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," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no.12, pp. 1708-1721, 2009.

[3] R. Chan, Q. Yang, and Y. Shen, "Mining High Utility Itemsets," in *Proc. of the 3rd IEEE Int'l Conf. on Data Mining*, pp. 19, November 19-22, 2003.

[4] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 1-12, May 15-18, 2000, Dallas, Texas, USA.

[5] H.-F. Li, H.-Y. Huang, Y.-C. Chen, Y.-J. Liu, and S.-Y. Lee, "Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams," in *Proc. of the 2008 8th IEEE Int'l Conf. on Data Mining*, pp. 881-886, December 15-19, 2008.

[6] C.-W. Lin, T.-P. Hong, W.-H. Lu, "An Effective Tree Structure for Mining High Utility Itemsets," *Expert Systems with Applications*, pp. 7419-7424, 2011.

[7] G.-C. Lan, T.-P. Hong, V. S. Tseng, "An Efficient Projection-based Indexing Approach for Mining High Utility Itemsets," *Knowledge and Information System*, vol. 38, pp. 85-107, 2014.

[8] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in *Proc. of the 1st Int'l Workshop on Utility-Based Data Mining*, pp. 90-99, August 21-21, 2005, Chicago, Illinois, USA.

[9] M. Liu, and J. Qu, "Mining high utility itemsets without candidate generation," in *Proc. of the 21st ACM SIGKDD Int'l Conf. on Knowledge Management*, pp. 55-64, October 29-November 02, 2012, Maui, Hawaii, USA.

[10] J. Liu, K. Wang, and B. Fung, "Direct Discovery of High Utility Itemsets without Candidate Generation," in *Proc.of the 2012 IEEE 12th Int'l Conf. on Data Mining*, pp. 984-989, December 10-13, 2012.

[11] Y. Li, J. Yeh, and C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data & Knowledge Engineering*, vol. 64, no. 1, pp. 198-217, January, 2008.

[12] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-mine: Fast and Space-Preserving Frequent Pattern Mining in Large Databases," *IIE Transactions*, vol. 39, no. 6, pp. 593-605, June, 2007.

[13] R. Srikant, and R. Agrawal, "Mining quantitative association rules in large relational tables," in *Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 1-12, June, 1996, Montreal, Canada.

[14] P. S. M. Tsai, and C.-M. Chen, "Mining quantitative association rules in a large database of sales transactions," *Journal of Information Science and Engineering*, pp. 667-681, 2001.

[15] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "UP-Growth: an efficient algorithm for high utility itemset mining," in *Proc. of 16th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 253-262, July 25-28, 2010, Washington, DC, USA.

[16] C. Wang, S. Cheng, and Y. Huang, "A fuzzy approach for mining high utility quantitative itemsets," in *Proc. of IEEE Int'l Conf. on Fuzzy System*, pp. 1909-1913, 2009.

[17] C.-W. Wu,Y.-F. Lin, P. S. Yu, and V. S. Tseng, "Mining high utility episodes in complex event sequences," in *Proc. of 19th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 536-544, August 11-14, 2013, Chicago, Illinois, USA.

[18] C.-W. Wu, P. Fournier-Viger, P. S. Yu, and V. S. Tseng, "Efficient mining of a concise and lossless representation of high utility itemsets," in *Proc. of 11th IEEE Int'l Conf. on Data Mining*, pp. 824-833, December 11-14, 2011, Vancouver, BC.

[19] C.-W. Wu, B.E. Shie, V. S. Tseng, and P. S. Yu, "Mining Top-k high utility itemsets," in *Proc. of 18th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 78-86, August 12-16, 2012, Beijing, China.

[20] H. Yao, and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, vol. 59, no. 3, pp. 603-626, 2006.

[21] S.-J. Yen, and Y.-S. Lee, "Mining high utility quantitative association rules," in *Proc. of the 9th Int'l Conf. on Data Warehousing and Knowledge Discovery*, pp. 283-292, September 03-07, 2007, Regensburg, Germany.