Efficient Algorithms for Mining Top-K High Utility Itemsets

Vincent S. Tseng, *Senior Member, IEEE*, Cheng-Wei Wu, Philippe Fournier-Viger, and Philip S. Yu, *Fellow, IEEE*

Abstract—*High utility itemsets* (*HUIs*) *mining* is an emerging topic in data mining, which refers to discovering all itemsets having a utility meeting a user-specified *minimum utility threshold min_util*. However, setting *min_util* appropriately is a difficult problem for users. Generally speaking, finding an appropriate minimum utility threshold by trial and error is a tedious process for users. If *min_util* is set too low, too many *HUIs* will be generated, which may cause the mining process to be very inefficient. On the other hand, if *min_util* is set too high, it is likely that no *HUIs* will be found. In this paper, we address the above issues by proposing a new framework for *top*-k *high utility itemset mining*, where *k* is the desired number of HUIs to be mined. Two types of efficient algorithms named *TKU* (*mining Top-K Utility itemsets*) and *TKO* (*mining Top-K utility itemsets in One phase*) are proposed for mining such itemsets without the need to set *min_util*. We provide a structural comparison of the two algorithms with discussions on their advantages and limitations. Empirical evaluations on both real and synthetic datasets show that the performance of the proposed algorithms is close to that of the optimal case of state-of-the-art utility mining algorithms.

Index Terms—Utility mining, high utility itemset mining, top-*k* pattern mining, top-*k* high utility itemset mining

1 Introduction

Frequent *itemset mining* (*FIM*) [1], [3], [8], [9], [18], [19], [20], [28], [29] is a fundamental research topic in data mining. However, the traditional FIM may discover a large amount of frequent but low-value itemsets and lose the information on valuable itemsets having low selling frequencies. Hence, it cannot satisfy the requirement of users who desire to discover itemsets with high utilities such as high profits. To address these issues, *utility mining* [2], [4], [7], [10], [11], [12], [13], [14], [15], [16], [17], [18], [22], [23], [25], [26], [27], [29], [34], [35], [36] emerges as an important topic in data mining and has received extensive attention in recent years. In utility mining, each item is associated with a utility (e.g. unit profit) and an occurrence count in each transaction (e.g. quantity). The utility of an itemset represents its importance, which can be measured in terms of weight, value, quantity or other information depending on the user specification. An itemset is called *high utility itemset* (*HUI*) if its utility is no less than a user-specified minimum utility threshold *min_util*. HUI mining is essential to many applications such as *streaming analysis* [2], [11], [35], *market analysis* [13], [17], [22], *mobile computing* [23] and *biomedicine* [4].

However, efficiently mining HUIs in databases is not an easy task because the *downward closure property* [1], [8] used in FIM does not hold for the utility of itemsets. In other words, pruning search space for HUI mining is difficult because a superset of a low utility itemset can be high utility. To tackle this problem, the concept of *transaction-weighted utilization* (*TWU*) model [13] was introduced to facilitate the performance of the mining task. In this model, an itemset is called *high transaction-weighted utilization itemset* (*HTWUI*) if its TWU is no less than *min_util*, where the TWU of an itemset represents an upper bound on its utility. Therefore, a HUI must be a HTWUI and all the HUIs must be included in the complete set of HTWUIs. A classical TWU model-based algorithm consists of two phases. In the first phase, called *phase I*, the complete set of HTWUIs are found. In the second phase, called *phase II*, all HUIs are obtained by calculating the exact utilities of HTWUIs with one database scan.

Although many studies have been devoted to HUI mining, it is difficult for users to choose an appropriate minimum utility threshold in practice. Depending on the threshold, the output size can be very small or very large. Besides, the choice of the threshold greatly influences the performance of the algorithms. If the threshold is set too low, too many HUIs will be presented to the users and it is difficult for the users to comprehend the results. A large number of HUIs also causes the mining algorithms to become inefficient or even run out of memory, because the more HUIs the algorithms generate, the more resources they consume. On the contrary, if the threshold is set too high, no HUI will be found. To find an appropriate value for the *min_util* threshold, users need to try different thresholds by guessing and re-executing the algorithms over and

- *V.S. Tseng and C.-W. Wu are with the Department of Computer Science, National Chao Tung University, 1001 University Road, Hsinchu City, Taiwan. E-mail: vtseng@cs.nctu.edu.tw, silvemoonfox@hotmail.com.*
- *P. Fournier-Viger is with the Department of Computer Science, University of Moncton, Moncton, NB, Canada. E-mail: philippe.fournier-viger@umoncton.ca.*
- *P.S. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, and the Institute for Data Science, Tsinghua University, Beijing, China. E-mail: psyu@cs.uic.edu.*

Manuscript received 25 Nov. 2014; revised 2 July 2015; accepted 7 July 2015. Date of publication 22 July 2015; date of current version 3 Dec. 2015.
Recommended for acceptance by X. Xiong.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2015.2458860

over until being satisfied with the results. This process is both inconvenient and time-consuming.

To precisely control the output size and discover the itemsets with the highest utilities without setting the thresholds, a promising solution is to redefine the task of mining HUIs as *mining top-k high utility itemsets* (*top-k HUIs*). The idea is to let the users specify *k*, i.e., the number of desired itemsets, instead of specifying the minimum utility threshold. Setting *k* is more intuitive than setting the threshold because *k* represents the number of itemsets that the users want to find whereas choosing the threshold depends primarily on database characteristics, which are often unknown to users.

Using a parameter *k* instead of the *min_util* threshold is very desirable for many applications. For example, to analyze customer purchase behavior, top-*k* HUI mining serves as a promising solution for users who desire to know "What are the top-*k* sets of products (i.e., itemsets) that contribute the highest profits to the company?" and "How to efficiently find these itemsets without setting the *min_util* threshold?". Although top-*k* HUI mining is essential to many applications, developing efficient algorithms for mining such patterns is not an easy task. It poses four major challenges as discussed below.

First, the utility of itemsets is neither *monotone* nor *anti-monotone* [1], [8]. In other words, the utility of an itemset may be equal to, higher or lower than that of its supersets and subsets. Therefore, many techniques developed in top-*k* frequent pattern mining that rely on anti-monotonicity to prune the search space cannot be directly applied to top-*k* high utility itemset mining.

The second challenge is how to incorporate the concept of top-*k* pattern mining with the TWU model. Although the TWU model is widely used in utility mining, it is difficult to adapt this model to top-*k* HUI mining because the exact utilities of itemsets are unknown in phase I. When a HTWUI is generated in phase I, we cannot guarantee that its utility is higher than other HTWUIs and that it is a top-*k* HUI before performing phase II. To guarantee that all the top-*k* HUIs can be captured in the set of HTWUIs, a naive approach is to run the algorithm with $min\_util = 0$. However, this approach may face the problem of a very large search space.

The third challenge is that the *min_util* threshold is not given in advance in top-*k* HUI mining. In traditional HUI mining, the search space can be efficiently pruned by the algorithms by using a given *min_util* threshold. However, in the scenario of top-*k* HUI mining, no *min_util* threshold is provided in advance. Therefore, the minimum utility threshold is initially set to 0 and the designed algorithm has to gradually raise the threshold to prune the search space. Such a threshold is an internal parameter of the designed algorithm and is called the *border minimum utility threshold* min_util_{Border} in this paper. It is different from the external parameter *min_util* that is given by users in advance. If an algorithm cannot raise the min_util_{Border} threshold effectively and efficiently, it would produce too many intermediate low utility itemsets during the mining process, which may degrade its performance in terms of execution time and memory usage. Thus the challenge is to design effective strategies that can raise the *min_util* threshold as high as possible and as quickly as possible, and further reduce as much as possible the number of candidates and intermediate low utility itemsets produced in the mining process.

The last challenge is how to effectively raise the min_util_{Border} threshold without missing any top-*k* HUIs. A good algorithm is one that can effectively raise the threshold during the mining process. However, if an incorrect method for raising the threshold is used, it may result in some top-*k* HUIs being pruned. Thus, how to raise the threshold efficiently and effectively without missing any top-*k* HUI is a crucial challenge for this work.

In this paper, we address all of the above challenges by proposing a novel framework for *top-k high utility itemset mining*, where *k* is the desired number of HUIs to be mined. Major contributions of this work are summarized as follows:

First, two efficient algorithms named *TKU* (*mining Top-K Utility itemsets*) and *TKO* (*mining Top-K utility itemsets in One phase*) are proposed for mining the complete set of top-*k* HUIs in databases without the need to specify the *min_util* threshold. The TKU algorithm adopts a compact tree-based structure named *UP-Tree* [25] to maintain the information of transactions and utilities of itemsets. TKU inherits useful properties from the TWU model and consists of two phases. In phase I, potential top-*k* high utility itemsets (*PKHUIs*) are generated. In phase II, top-*k* HUIs are identified from the set of PKHUIs discovered in phase I. On the other hand, the TKO algorithm uses a list-based structure named *utility-list* [14] to store the utility information of itemsets in the database. It uses vertical data representation techniques to discover top-*k* HUIs in only one phase.

Second, we investigate the properties of the TKU and TKO algorithms and develop different strategies to effectively raise the border thresholds in both algorithms. For TKU, we propose five strategies *PE*, *NU*, *MD*, *MC* and *SE* to effectively raise the border minimum utility threshold. For TKO, we integrate the novel strategies *RUC*, *RUZ* and *EPB* for pruning the search space.

Third, we conducted different kinds of experiments on synthetic and real datasets to evaluate the performance of the proposed algorithms and effectiveness of the proposed strategies. Empirical results show that the performance of the proposed algorithms is close to that of the state-of-the-art utility mining algorithms UP-Growth [25] and HUI-Miner [14] tuned with the optimal minimum utility thresholds.

The remainder of this paper is organized as follows. Section 2 introduces the background and related works of top-*k* HUI mining. Section 3 and Section 4 respectively present the proposed TKU and TKO algorithms. Experimental result is reported in Section 5. Finally, conclusion and future works are given in Section 6.

2 Background and Problem Definition

This section defines the problem of top-*k* high utility itemset mining and then introduces related studies.

2.1 Problem Definition

Let be a finite set of distinct *items* $I^* = \{I_1, I_2, \ldots, I_m\}$. A *transactional database* $D = \{T_1, T_2, \ldots, T_n\}$ is a set of transactions, where each transaction $T_r \in D$ is a subset of I^* and has an unique identifier *r*, called *Tid*. Each item $I_j \in T_r$ has a

TABLE 1
An Example Database

TID	Transaction	Transaction Utility (TU)
T_1	(A,1)(C,1)(D,1)	8
T_2	(A,2)(C,6)(E,2)(G,5)	27
T_3	(A,1)(B,2)(C,1)(D,6)(E,1)(F,5)	30
T_4	(B,4)(C,3)(D,3)(E,1)	20
T_5	(B,2)(C,2)(E,1)(G,2)	11

positive value $Q(I_j, T_r)$, called its *internal utility* in T_r. Each item $I_j \in I^*$ is associated with a positive number $P(I_j, D)$, called its *external utility*. An *itemset* $X = \{I_1, I_2, \ldots, I_L\}$ is a set of L distinct items, where $I_j \in I^*$ and L is the *length of* X. An *L-itemset* is an itemset of length L.

Definition 1 (Absolute utility of an item). *The absolute utility of an item $I_j \in I^*$ in a transaction T_r is denoted as $EU(I_j, T_r)$ and defined as $P(I_j, D) \times Q(I_j, T_r)$.*

Definition 2 (Absolute utility of an itemset in a transaction). *The absolute utility of an itemset X in a transaction T_r is defined as $EU(X, T_r) = \sum_{I_j \in X} (I_j, T_r)$.*

Definition 3 (Absolute utility of an itemset in a database). *The absolute utility of an itemset X in D is defined as $EU(X) = \sum_{T_r \in D \wedge X \subseteq Tr} (X, T_r)$.*

Definition 4 (Transaction utility and total utility). *The transaction utility (TU) of a transaction T_r is defined as $TU(T_r) = EU(T_r, T_r)$. The total utility of a database D is denoted as $TotalU_{DB}$ and defined as $\sum_{T_r \in D} TU(T_r)$.*

Definition 5 (Utility of an itemset in a database). *The (relative) utility of X is defined as $U(X) = EU(X)/ TotalU_{DB}$.*

Definition 6 (High utility itemset). *An itemset X is called high utility itemset (HUI) iff U(X) is no less than a user-specified minimum utility threshold min_util ($0\% \leq$ min_util $\leq 100\%$). Otherwise, X is a low utility itemset An equivalent definition is that X is high utility iff $EU(X) \geq$ abs_min_util, where abs_min_util = min_util $\times$ $TotalU_{DB}$.*

Definition 7 (High utility itemset mining). *Let δ ($0\% \leq \delta \leq 100\%$) be the minimum utility threshold, the complete set of HUIs in D is denoted as $f_{HUI}(D, \delta)$. The goal of HUI mining is to discover $f_{HUI}(D, \delta)$.*

Example 1. Let Table 1 be an example database containing five transactions. Each row in Table 1 represents a transaction, where each letter represents an item and has a purchase quantity (i.e., internal utility). The unit profit (i.e., external utility) of each item is shown in Table 2. If *abs_min_util* = 30, the complete set of HUIs in Table 1 is {{BD}:30, {ACE}:31, {BCD}:34, {BCE}:31, {BDE}:36, {BCDE}:40, {ABCDEF}:30}, where the number beside each itemset is its absolute utility.

Because the utility of an itemset may be equal to, higher or lower than that of its supersets and subsets, we cannot directly use the anti-monotone property (also known as downward closure property) to prune the search space. To facilitate the mining task, Liu et al. introduced the concept of *transaction-weighted downward closure property* [13], which is based on the following definitions.

TABLE 2
Profit Table

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

Definition 8 (Transaction-weighted utilization). *The transaction-weighted utilization of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is defined as $TWU(X) = \sum_{X \subseteq Tr \wedge T_r \in D} TU(T_r)$.*

Definition 9 (High TWU itemset). *An itemset X is a high TWU itemset iff $TWU(X) \geq$ abs_min_util.*

Property 1 (TWDC property). *The transaction-weighted downward closure (TWDC) property states that for any itemset X that is not a high TWU itemset, all its supersets are low utility* [13].

Rationale. Let X be an itemset that is not a high TWU itemset and $g(X)$ be the set of transactions containing X. Let Y be a superset of X and $g(Y)$ be the set of transactions containing Y. Because $Y \subseteq X$, it follows that $g(Y) \subseteq g(X)$, and thus that $TWU(X) = \sum_{T_r \in D \wedge r \in g(X)} EU(T_r, T_r) \geq \sum_{T_s \in D \wedge s \in g(X)} EU(Y, T_s) = EU(Y)$. Because $TWU(X) \geq EU(Y)$ and $TWU(X) <$ *abs_min_util*, it yields that $EU(Y) <$ *abs_min_util*.

Example 2. The transaction utility of T_1 in Table 1 is $TU(T_1) = EU(\{A\}, T_1) + EU(\{C\}, T_1) + EU(\{D\}, T_1) = (1 \times 5 + 1 \times 1 + 1 \times 2) = 8$. The last column of Table 1 shows the transaction utility of each transaction. The set of transactions containing {G} is $g(\{G\}) = \{T_2, T_5\}$. The TWU of {G} is $TWU(\{G\}) = TU(T_2) + TU(T_5) = 38$. If *abs_min_util* = 40, all the supersets of {G} are low utility.

Definition 10 (Top-*k* high utility itemset). *An itemset X is called top-k high utility itemset (top-k HUI) in D iff there are less than k itemsets whose utilities are larger than EU(X) in $f_{HUI}(D, 0)$.*

Property 2. *Let KH be the complete set of top-k HUIs in D. KH may contain less than k itemsets when $|f_{HUI}(D, 0)| < k$. Besides, KH may contain more than k HUIs when some itemsets have the same utility.*

Definition 11 (Optimal minimum utility threshold). *An absolute minimum utility threshold δ^* is called optimal minimum utility threshold iff there does not exist a threshold $\delta > \delta^*$ such that $|f_{HUI}(D, \delta)| = |KH|$. An equivalent definition is that $\delta^* = min\{U(X) | X \in KH\}$.*

Problem Statement. *Given a transactional database D and the desired number of HUIs k, the problem of top-k high utility itemsets mining is to discover all the itemsets having a utility no less than δ^* in D.*

2.2 Related Works

This section introduces related works about top-*k* high utility itemset mining, including *high utility itemset mining*, *top-k frequent pattern mining* and *top-k high utility itemset mining*.

2.2.1 High Utility Itemset Mining

In recent years, high utility itemset mining has received lots of attention and many efficient algorithms have been proposed, such as *Two-Phase* [13], *IHUP* [2], *IIDS* [17], *UP-Growth* [25], *d^2HUP* [15] and *HUI-Miner* [14]. These algorithms can be generally categorized into two types: *two-phase* and *one-phase* algorithms. The main characteristic of two-phase algorithms is that they consist of two phases. In the first phase, they generate a set of candidates that are potential high utility itemsets. In the second phase, they calculate the exact utility of each candidate found in the first phase to identify high utility itemsets. Two-Phase, IHUP, IIDS and UP-Growth are two-phase based algorithms. UP-Growth is one of the state-of-the-art two-phase algorithms, which incorporates four effective strategies *DGU*, *DGN*, *DLU* and *DLN* for pruning candidates in the first phase.

One the contrary, the main characteristic of one-phase algorithms is that they discover high utility itemsets using only one phase and produce no candidates. *d^2HUP* and *HUI-Miner* are one-phase algorithms. *d^2HUP* transforms a horizontal database into a tree-based structure called *CAUL* [15] and adopts a pattern-growth strategy to directly discover high utility itemsets in databases. HUI-Miner considers a database of vertical format and transforms it into *utility-lists* [14]. The utility-list structure used in HUI-Miner allows directly computing the utility of generated itemsets in main memory without scanning the original database.

Although the above studies may perform well in some applications, they are not developed for top-*k* high utility itemset mining and still suffer from the subtle problem of setting appropriate thresholds.

2.2.2 Top-k Pattern Mining

Many studies have been proposed to mine different kinds of top-*k* patterns, such as *top-k frequent itemsets* [3], [19], [20], *top-k frequent closed itemsets* [3], [28], *top-k closed sequential patterns* [24], *top-k association rules* [6], *top-k sequential rules* [5], *top-k correlation patterns* [31], [32], [33] and *top-k cosine similarity interesting pairs* [38]. What distinguishes each top-*k* pattern mining algorithm is the type of patterns discovered, as well as the data structures and search strategies that are employed. For example, some algorithms [5], [6] use a rule expansion strategy for finding patterns, while others rely on a pattern-growth search using structures such as FP-Tree [19], [20], [28]. The choice of data structures and search strategy affect the efficiency of a top-*k* pattern mining algorithm in terms of both memory and execution time. However, the above algorithms discover top-*k* patterns according to traditional measures instead of the utility measure. As a consequence, they may miss patterns yielding high utility.

2.2.3 Top-k High Utility Pattern Mining

The task of top-*k* high utility pattern mining was introduced by Chan et al. [4]. But the definition of high utility itemset used in their study is different from the one used in this work. Chan et al.'s study has considered utilities of various items, but quantitative values of items in transactions were not taken into consideration. In [30], we have defined the task of *top-k high utility itemset mining* by considering both quantities and profits of items. This work has inspired a few studies for mining top-*k* high utility patterns. Zihayat and An [37] have proposed an efficient algorithm *T-HUDS* for mining top-*k* HUIs over data streams. Yin et al. [36] have proposed a new framework for mining top-*k* high utility sequential patterns. Recently, Ryang and Yun extended [30] to propose the *REPT* algorithm [21] with four strategies *PUD*, *RIU*, *RSD* and *SEP* for top-*k* HUI mining. In REPT, besides the parameter *k*, users need to set another parameter *N* to control the effectiveness of *RSD* [21]. However, it is not easy for users to choose an appropriate *N* value and the choice of *N* greatly influences the performance of REPT (see Section 5).

TABLE 3
Items and Their TWUs

Item	A	B	C	D	E	F	G
TWU	65	61	96	58	88	30	38

3 The TKU Algorithm

In this section, we propose an efficient algorithm named *TKU* (*mining Top-k Utility itemsets*) for discovering top-*k* HUIs without specifying *min_util*. We first present its basic version named *TKU$_{Base}$* and then describe the TKU algorithm, which includes several novel strategies.

3.1 The Baseline Approach TKU$_{\text{Base}}$

The baseline approach TKU$_{\text{Base}}$ is an extension of UP-Growth [25], a tree-based algorithm for mining HUIs. TKU$_{\text{Base}}$ adopts the UP-Tree structure of UP-Growth to maintain the information of transactions and top-*k* HUIs. TKU$_{\text{Base}}$ is executed in three steps: (1) constructing the UP-Tree, (2) generating potential top-*k* high utility itemsets (*PKHUIs*) from the UP-Tree, and (3) identifying top-*k* HUIs from the set of PKHUIs.

3.1.1 UP-Tree Structure

Then, we briefly introduce the UP-Tree structure. For more details about it, readers are referred to [25]. Each node *N* of a UP-Tree has five entries: *N.name* is the item name of *N*; *N. count* is the support count of *N*; *N.nu* is the node utility of *N*; *N.parent* indicates the parent node of *N*; *N.hlink* is a node link which may point to a node having the same item name as *N.name*. The *Header table* is a structure employed to facilitate the traversal of the UP-Tree.

A header table entry contains an item name, an estimated utility value, and a link. The link points to the first node in the UP-Tree having the same item name as the entry. The nodes whose item names are the same can be traversed efficiently by following the links in header table and the node links in the UP-Tree.

3.1.2 Construction of UP-Tree

A UP-Tree can be constructed by scanning the original database twice. In the first scan, the transaction utility of each transaction and TWU of each item are computed. Thus, items and their TWUs are obtained. For example, Table 3 shows items and their TWUs for the database of Table 1. Subsequently, items are inserted into the header table in descending order of their TWUs. During the second database scan, transactions are reorganized and then inserted

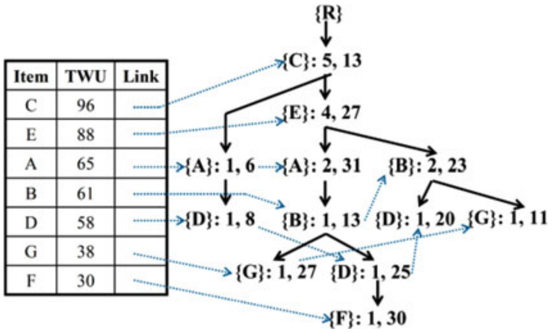


Fig. 1. A UP-Tree after inserting all the transactions in Table 1.

into the UP-Tree. Initially, the tree is created with a root R. When a transaction is retrieved, items in the transaction are sorted in descending order of TWU. A transaction after the above reorganization is called *reorganized transaction* and its transaction utility is called *Reorganized Transaction Utility (RTU)*. The RTU of a reorganized transaction T_r' is denoted as $RTU(T_r')$. When a reorganized transaction $T_r' = \{I_1, I_2, \ldots, I_M\}$ $(I_j \in I^*, 1 \le j \le M)$ is retrieved, TKU$_{\text{Base}}$ calls the function *Insert_Reorganized_Transaction*(N, I_j) and applies the strategy *Discarding Global Node utilities (DGN)* [25] to insert T_r'.

The function *Insert_Reorganized_Transaction* (N, I_j) takes a node N in the UP-Tree and an item I_j $(I_j \in T_r', 1 \le j \le M)$ in the reorganized transaction T_r' as inputs. The function is performed as follows:

1. If N has a child node ChN such that $ChN.item = I_j$, then increment $ChN.count$ by 1; Otherwise, create a new child node ChN with $ChN.item = I_j$, $ChN.count = 1$, $ChN.parent = N$ and $ChN.nu = 0$.
2. Increase $ChN.nu$ by $(RTU(T_r') - \sum_{i=(j+1)}^{M} EU(I_i, T'_r))$, where $I_j\, T_r'$ and $1 \le j \le M$).
3. Call *Insert_Reorgnized_Transaction*(ChN, I_{j+1}) if $j \le M$.

After inserting all the reorganized transactions, the construction of the UP-Tree is completed. Fig. 1 shows a UP-Tree after inserting all the transactions in Table 1 when $abs\_min\_util = 0$. In Fig. 1, the node utility and support count of node {C} are respectively 5 and 13.

3.1.3 Generating PKHUIs from the UP-Tree

The TKU$_{\text{Base}}$ algorithm uses an internal variable named *border minimum utility threshold* (denoted as min_util_{Border}) which is initially set to 0 and raised dynamically after a sufficient number of itemsets with higher utilities has been captured during the generation of PKHUIs. The development of the proposed method is based on the following definitions and lemmas.

Lemma 1. *Let $P = \langle X_1, X_2, \ldots, X_M\rangle$ be a set of itemsets $(M \ge k)$, where X_i is the ith itemset in P and $EU(X_i) \ge EU(X_j) > 0, \forall\, i < j$. In other words, X_i is the itemset with the ith highest utility in P. For any itemset Y, if $EU(Y) < EU(X_k)$, Y is not a top-k HUI.*

Rationale. According to Definition 10, if there exists k itemsets whose utilities are higher than the utility of Y, Y is not a top-k HUI.

Lemma 2. *Let $P = \langle X_1, X_2, \ldots, X_M\rangle$ be a set of itemsets $(M \ge k)$, where X_i is the i-th itemset in P and $EU(X_i) \ge EU(X_j) > 0, \forall\, i < j$. If $\delta_P = EU(X_k)$, $f_{HUI}(D, \delta^*) \subseteq f_{HUI}(D, \delta_P)$.*

TABLE 4
Items and Their Mius

Item	A	B	C	D	E	F	G
Miu	5	4	1	2	3	5	2

Rationale. Let KH be the complete set of top-k HUIs. If $|KH| \ge k$, $\delta^* = min\{EU(X)\,|\,X \in KH\}$ (by Definition 11). Because $\delta^* = min\{EU(X)\,|\,X \in KH\} \ge min\{EU(X_i)\,|\,X_i \in P, 1 \le i \le k\} = EU(X_k) = \delta_{\text{P}}$, $\delta^* \ge \delta_{\text{P}}$ and $f_{HUI}(D, \delta^*) \subseteq f_{HUI}(D, \delta_{\text{P}})$.

Example 3. Consider that $k = 4$ and $abs\_min\_util = 0$. Let P be the set of all 1-itemsets {{A}:20, {D}:20, {B}:16, {E}:15, {C}:13, G:7, F:5} in D, where the number beside each itemset is its absolute utility. By Lemma 1, {C}, {G}, {F} are unpromising to be the top-4 HUIs. Therefore abs_min_util can be raised to the fourth highest utility value in P (i.e., 15) and no top-k HUIs will be missed.

After raising abs_min_util, TKU$_{\text{Base}}$ applies the UP-Growth search procedure with $abs\_min\_util = min\_util_{Border}$ to generate PKHUIs. Though Lemma 1 provides a way to raise min_util_{Border}, it cannot be applied during the generation of PKHUIs in phase I. This is because the utilities of PKHUIs are unknown during phase I. A solution to this problem is to use a lower bound on the utility of PKHUIs during phase I to raise min_util_{Border}. A lower bound on the utility of PKHUIs is provided by the following definitions.

Definition 12 (Minimum utility of an item). *The minimum utility of an item $I \in I^*$ is denoted as $miu(I)$ and defined as the value $EU(I, T_i)$ for which $\neg\, \exists\, T_j \in D$ such that $0 < EU(I, T_j) < EU(I, T_i)$. An equivalent definition is that $miu(I) = min\{EU(I, T_r)\,|\,T_r \in D$ and $r \in g(I)\}$.*

Definition 13 (Minimum utility of an itemset). *The minimum utility of an itemset $X = \{\mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_M\}$ is defined as $\mathrm{MIU}(X) = \sum_{i=1}^{M} miu(I_i) \times SC(X)$.*

Example 4. Consider the database of Table 1. Table 4 shows the *mius* of items of this database. The minimum utility of item {B} is $miu(\{B\}) = min\{EU(\{B\}, T_3'), EU(\{B\}, T_4'), EU(\{B\}, T_5')\} = min\{4, 8, 4\} = 4$. The minimum utility of itemset {BC} is $MIU(\{BC\}) = [miu(\{B\}) + miu(\{C\})] \times SC(\{BC\}) = [4+1] \times 3 = 15$.

Lemma 3. *Let $C = \langle X_1, X_2, \ldots, X_M\rangle$ be a set of itemsets $(M \le k)$, where X_i is the ith itemset in C and $MIU(X_i) \le MIU(X_j) > 0$, $\forall i < j$. In other words, X_i is the itemset with the ith highest MUI value in C. For any itemset Y, if $TWU(Y) < \delta_{MC} = min\{MIU(X_i)\,|\,X_i \in C, 1 \le i \le k\}$, then Y is not a top-k HUI.*

Rationale. According to *Definition 8, $EU(Y) \le TWU(Y)$. If $TWU(Y) < \delta_{MC}$, we have $EU(Y) < \delta_{MC}$. Besides, $0 < EU(Y) < MIU(X_i) \le EU(X_i)$, $\forall X_i \in C$, $1 \le i \le k$. According to Definition 10, if there exist k itemsets whose utilities are higher than the utility of Y, Y is not a top-k HUI.*

Lemma 4. *Let $C = \langle X_1, X_2, \ldots, X_M\rangle$ be a set of itemsets $(M \le k)$, where X_i is the ith itemset in C and $MIU(X_i) \le MIU(X_j) > 0$, $\forall i < j$. If $\delta_{MC} = MIU(X_k)$, $f_{HUI}(D, \delta^*) \subseteq f_{HUI}(D, \delta_{MC})$.*

Rationale. Let KH be the complete set of top-k HUIs. If $|KH| \ge k$, $\delta^* = min\{EU(X)\,|\,X \in KH\}$ (by Definition 11). Because $\delta^* = min\{EU(X)\,|\,X \in KH\} \ge min\{EU(X_i)\,|\,X_i \in C, 1 \le i \le k\} \ge min$

TABLE 5
Items and Their Maus

Item	A	B	C	D	E	F	G
Mau	5	8	3	6	6	5	5

$\{MIU(X_i) \,|\, X_i \in C, 1 \le i \le k\} = MIU(X_k)$, we have $\delta^* \ge \delta_{MC}$. Therefore, $f_{HUI}(D, \delta^*) \subseteq f_{HUI}(D, \delta_{MC})$.

Lemma 5. *For any itemset X, if $TWU(X) < abs\_min\_util \le \delta^*$, X and all its supersets are not top-k HUIs.*

Definition 14 (Maximum utility of an item). *The maximum utility of an item $I \in I^*$ is denoted as mau(I) and defined as the value $EU(I, T_i)$ for which $\neg\exists\, T_j \in D$ such that $EU(I, T_j) > EU(I, T_i) > 0$. An equivalent definition is that $mau(I) = max\{EU(I, T_r) \,|\, T_r \in D, r \in g(I)\}$. Table 5 shows the mau of each item in the database of Table 1.*

Definition 15 (Maximum utility of an itemset). *The maximum utility of an itemset $X = \{I_1, I_2, \ldots, I_M\}$ is defined as $MAU(X) = \sum_{i=1}^{M} mau(I_i) \times SC(X)$.*

Lemma 6. *For any itemset X, if $MAU(X) < abs\_min\_util \le \delta^*$, X is not a top-*k *HUI*

Rationale. According to Definition 15, we have EU(X) ≤ MAU(X). If MAU(X) < $abs\_min\_util \le \delta^*$, EU(X) < $abs\_min\_util \le \delta^*$. By Definition 10, X is not a top-*k* HUI.

Lemma 7. *For any itemset X, the relationship between MAU(X), TWU(X), EU(X) and MIU(X) is $MIU(X) \le EU(X) \le min\{MAU(X), TWU(X)\}$.*

Definition 16 (Potential top-*k* high utility itemset). *An itemset is called Potential top-K High Utility Itemset (PKHUI) if its estimated utility value (i.e., TWU) and MAU are no less than the* $\mathrm{min\_util_{Border}}$ *threshold.*

$\mathrm{TKU_{Base}}$ integrates the UP-Growth search procedure to find top-*k* HUIs. A detailed running example of the UP-Tree construction and UP-Growth mining process can be found in [25].

Property 3. *In UP-Growth, each candidate $X = \{I_1, I_2, \ldots, I_M\}$ is generated with its estimated utility value ESTU(X). This estimated utility value has the following properties: (1) $MIU(X) \le EU(X) \le ESTU(X) \le TWU(X)$; (2) $EU(X) \le min\{ESTU(X), MAU(X)\}$; (3) if $ESTU(X) < abs\_min\_util$, X and its concatenations are low utility.*

Based on the above lemmas and properties, we propose ideas to raise min_util_{Border} during the Phase I of $\mathrm{TKU_{Base}}$. Fig. 2 gives the resulting pseudo code of $\mathrm{TKU_{Base}}$. Each time a candidate itemset *X* is found by the UP-Growth search procedure, the $\mathrm{TKU_{Base}}$ algorithm checks whether its estimated utility value *ESTU*(*X*) is no less than min_util_{Border}. If *ESTU*(*X*) is less than min_util_{Border}, *X* and all its concatenations are not top-*k* HUIs (Property 3). Besides, $\mathrm{TKU_{Base}}$ checks whether *MAU*(*X*) is no less than min_util_{Border} (Line 5). If *MAU(X)* is smaller than min_util_{Border}, *X* is not a top-*k* HUI (Lemma 6). Otherwise, *X* is considered a candidate for Phase II and it is outputted with *min*{*ESTU*(*X*), *MAU*(*X*)} according to Property 3 (Line 6). If *X* is a valid PKHUI and *MIU*(*X*) ≥ min_util_{Border}, *MIU*(*X*) can be used to raise min_util_{Border} by the proposed strategy *MC* (*raising the threshold by MUIs of Candidates*) (Line 9-10), which is explained thereafter.

```
ALGORITHM: $\mathrm{TKU_{Base}}$
  Input:  (1) A database $D$;
          (2) The number of desired HUIs $k$;
Output:  (1) The complete set of PKHUIs $C$;
01. Set $min\_util_{Border} \leftarrow 0$; $TopK\text{-}MIU\text{-}List \leftarrow Ø$; $C \leftarrow Ø$;
02. Construct a UP-Tree by scanning $D$ twice;
03. //Apply a UP-Growth search procedure to generate PKHUIs;
04. For each PKHUI generated with estimated utility $ESTU(X)$ do
05. {  If ($ESTU(X) \ge min\_util_{Border}$ and $MAU(X) \ge min\_util_{Border}$)
06. |  {  Output $X$ and $min\{ESTU(X), MAU(X)$; $C \leftarrow C \cup X$;
07. |  |  If($MIU(X) \ge min\_util_{Border}$ )
08. |  |  {  //Raise $min\_util_{Border}$ by the strategy $MC$;
09. |  |  |  $min\_util_{Border} \leftarrow MC(MIU(X), TopK\text{-}MIU\text{-}Lit)$;
10. |  |  }
11. |  }
12. }
```

Fig. 2. The pseudo code of the $\mathrm{TKU_{Base}}$ algorithm.

To efficiently update min_util_{Border}, we use a min-heap structure named *TopK-MIU-List* to maintain the *k* highest MIU values of PKHUIs found until now. Each time a PKHUI *X* is found and its MIU is higher than min_util_{Border}, *X* is added into *TopK-MIU-List*. If there are less than *k* MIU values in *TopK-MIU-List*, min_util_{Border} will not change. Once *k* MIU values are found and the *k*-th MIU value (denoted as $MIU_{k\text{-}th}$) in *TopK-MIU-List* is higher than min_util_{Border}, min_util_{Border} is raised to $MIU_{k\text{-}th}$ in *TopK-MIU-List* according to Lemma 3. Otherwise, if there exist more than *k* MUI values in *TopK-MIU-List*, min_util_{Border} is raised to $MIU_{k\text{-}th}$ in *TopK-MIU-List* and the MIU values that are less than $MIU_{k\text{-}th}$ in *TopK-MIU-List* are removed. The algorithm continues searching for other PKHUIs until no candidate is found by the UP-Growth procedure.

Strategy 1 (MC). Raising the threshold by MUIs of Candidates. For any newly mined candidate itemset *X*, if *MIU(X)*, *ESTU(X) and MAU(X)* are no less than the current min_util_{Border}, then *MIU(X) is added* to *TopK-MIU-List*. If $|TopK\text{-}MIU\text{-}List| \ge k$ and $MIU_{k\text{-}th} > min\_util_{Border}$, min_util_{Border} can be safely raised to $MIU_{k\text{-}th}$.

3.1.4 Identifying Top-k HUIs from PKHUIs

After identifying PKHUIs, $\mathrm{TKU_{Base}}$ calculates the utility of PKHUIs by scanning the original database once, to identify the top-*k* HUIs. This process is similar to that of Phase II in [25]. However, in previous work [25], all the candidates were considered. In this work, we only consider a candidate itemset *X* if its estimated utility value reached after phase I is no less than min_util_{Border}, i.e., $min\{ESTU(X), MAU(X)\} \ge min\_util_{Border}$.

3.2 The TKU Algorithm

In this section, we propose four strategies to effectively raise min_util_{Border} during different stages of the mining process. The four strategies are incorporated in $\mathrm{TKU_{Base}}$ to form the advanced TKU algorithm.

3.2.1 Pre-Evaluation Step

Though $\mathrm{TKU_{Base}}$ provides a way to mine top-*k* HUIs, min_util_{Border} is set to 0 before the construction of the UP-Tree. This results in the construction of a full UP-Tree in memory, which degrades the performance of the mining task. If

Item	B	C	D	E	F	G
A	9	28	24	24	10	15
B		17	14	18	0	6
C			0	0	0	0
D				0	0	0
E					0	0
F						0

Fig. 3. Pre-evaluation matrix.

min_util_{Border} could be raised before the construction of the UP-Tree and prune more unpromising items [25] in transactions, the number of nodes maintained in memory could be reduced and the mining algorithm could achieve better performance. Based on this idea, we propose a strategy named *PE* (*Pre-evaluation Step*) to raise min_util_{Border} during the first scan of the database.

Strategy 2 (PE: Pre-Evaluation). The strategy *PE* uses a structure named *Pre-Evaluation Matrix (PEM)* to store lower bounds of the utilities of certain 2-itemsets. Each entry in *PEM* is denoted as *PEM[x][y]* and corresponds to the lower bound of $EU(\{x, y\})$, where $x, y \in I^*$. Initially, each value in *PEM* is set to 0. When a transaction $T_r = \{I_1, I_2, \ldots, I_M\}$ $(I_j \in I^*, 1 \leq i \leq M)$ is retrieved during the first database scan, the utility of $\{I_1\} \cup \{I_i\}$ $(1 < i \leq M)$ in T_r is added to the value of the corresponding entry of $PEM[I_1][I_i]$ in *PEM*. After scanning all the transactions, if the *k*-th highest value in *PEM* is higher than min_util_{Border}, min_util_{Border} can be raised to the *k*-th highest value in *PEM*. The space complexity of the strategy is $O(|I^*|^2/2)$, where $|I^*|$ is the number of distinct items in the database.

Example 5. Consider the database of Table 1. When T_1 = {(A,1), (C,1), (D,1)} is retrieved, the corresponding entries *PEM*[A][C], *PEM*[A][D] are accumulated with *EU*({AC}, T_1) = 6 and *EU*({AD}, T_1) = 7. The remaining transactions in the database are processed by the same procedure. After that, if min_util_{Border} is lower than the *k*-th highest value in *PEM*, min_util_{Border} is set to the *k*-th highest value in PEM. Fig. 3 shows the value of each entry in *PEM* after scanning the database. If $k = 4$, the fourth highest value in *PEM* is *PEM*[B][E] = 18. If min_util_{Border} is less than this value, min_util_{Border} is raised to 18.

Notice that in TKU_{Base}, the strategy *DGU* proposed in [25] cannot be applied, because min_util_{Border} is set to 0 before the construction of the UP-Tree. However, if we apply the strategy *PE* to raise min_util_{Border} during the first database scan, *DGU* can be further applied to prune those items whose TWUs are less than min_util_{Border}, which reduces the size of the UP-Tree and the number of candidates produced in phase I.

3.2.2 Raising the Threshold by Node Utilities

We also propose a strategy called *NU* (*raising the threshold by Node Utilities*), which is applied during the construction of the UP-Tree. The strategy NU is developed based on the following lemmas.

Lemma 8. *Let PATH* = $\langle N_1, N_2, \ldots, N_M, R\rangle$ *be a path from a node* N_1 *to the root R in UP-Tree and* $I_i \in I^*$ *be the item name of* N_i, $1 \leq i \leq M$. *PATH* = $\langle N_1, N_2, \ldots, N_M, R\rangle$ *represents a unique itemset* $X = \{I_1, I_2, \ldots, I_M\}$ *in the database. Besides, the node utility of* N_1 *is a lower bound on the utility of X.*

Rationale. The UP-Tree is constructed by applying the strategy DGN [25]. According to the rationale described in [25], the utility of the itemset $X = \{I_1, I_2, \ldots, I_M\}$ *is guaranteed* to be higher than the node utility of N_1. Therefore, $N_1.nu \leq EU(\{I_1, I_2, \ldots, I_M\})$.

Lemma 9. *If there are M nodes in the UP-Tree, there are at least M distinct itemsets whose utilities are higher than 0.*

Rationale. By Lemma 8, each path from a node in the UP-Tree to the root forms a unique path, which represents a unique itemset whose utility is higher than zero in the database. Therefore, *M* distinct nodes in the UP-Tree yield *M* distinct itemsets whose utilities are higher than zero.

Lemma 10. *Let SetNode* = $\langle N_1, N_2, \ldots, N_M\rangle$ *be an ordered set containing all nodes in the UP-Tree* $(M \geq k)$. *Let* N_i *be the* i*th node in SetNode and* $N_i.nu \geq N_j.nu > 0, \forall i < j$. *If* $\delta_{NU} = N_k.nu$, *then* $f_{HUI}(D, \delta^*) \subseteq f_{HUI}(D, \delta_{NU})$.

Rationale. By Lemma 8, each path from a node $N_i \in$ *SetNode* to the root R represents a unique itemset N_i, $1 \leq i \leq M$. Let *SetItemset* = $\langle X_1, X_2, \ldots, X_M\rangle$ be an ordered set of itemsets that are represented by the nodes in *SetNode*, where $EU(X_i) \geq EU(X_j) > 0, \forall i < j$. Let *KH* be the complete set of top-*k* HUIs in the database *D*. If $|KH| \geq k$, then $\delta^* = min\{EU(X) \mid X \in KH\}$ (Definition 11). Because $\delta^* = min\{EU(X) \mid X \in KH\} \geq min\{EU(X_i) \mid X_i \in SetItemset,\ 1 \leq i \leq k\} \geq min\{N_i.nu \mid N_i \in NodeSet, 1 \leq i \leq k\} = \delta_{NU}$, we have $\delta^* \geq \delta_{NU}$ and $f_{HUI}(D, \delta^*) \subseteq f_{HUI}(D, \delta_{NU})$.

By Lemma 8, 9 and 10, if there are no less than *k* nodes in the UP-Tree during its construction and the *k*-th highest node utility in the UP-Tree is higher than the current min_util_{Border}, min_util_{Border} can be safely raised to the *k*-th highest node utility in the UP-Tree.

Example 6. Let the notation N_α represents a node of the UP-Tree such that α is the item stored in N_α. If $k = 4$, when the first reorganized transaction $T_1{'}$ = {(C,1), (A,1), (D,1)} is inserted into the UP-Tree, the nodes $N_{\{C\}}$, $N_{\{A\}}$ and $N_{\{D\}}$ are created with node utilities 1, 6 and 8, which are respectively lower bounds on the utilities of itemsets {C}, {AC} and {DAC}. When the second reorganized transaction $T_2{'}$ = {(C,6), (E,2), (A,2), (G,5)} is inserted into the UP-Tree, there are more than four nodes in the tree. By Lemma 10, min_util_{Border} can be raised to the fourth highest node utility in the current UP-Tree.

Strategy 3 (NU: raising the threshold by Node Utilities). The strategy NU is applied during the construction of the UP-Tree (during the second database scan). If there are more than *k* nodes in the current UP-Tree and the *k-th highest node utility value* $NU_{k\text{-}th}$ is higher than min_util_{Border}, min_util_{Border} can be raised to $NU_{k\text{-}th}$. After inserting all reorganized transactions, the size of the constructed UP-Tree can be further reduced by pruning items whose TWU values are less than min_util_{Border} in the UP-Tree.

3.2.3 Raising the Threshold by MIU Values of Descendents

The third strategy that we propose is called *MD* (*raising the threshold by MIU values of Descendents*). It is applied after the

TABLE 6
MIU Values of Descendents of Node $N_{\{C\}}$

Descendent	$N_{\{E\}}$	$N_{\{A\}}$	$N_{\{B\}}$	$N_{\{D\}}$	$N_{\{G\}}$	$N_{\{F\}}$
SC	4	3	3	3	2	1
MIU	16	18	15	9	3	6

construction of the UP-Tree and before the generation of PKHUIs.

Strategy 4 (MD: raising the threshold by MIU values of Descendents). Let the notation N_α represents a node of the UP-Tree such that α is the item stored in N. For each node N_α under the root of UP-Tree, the algorithm traverses the sub-tree under node N_α once to calculate the support count of the itemset $\{\alpha \cup \beta\}$ for every descendent node N_β of N_α. For each itemset $\{\alpha \cup \beta\}$, the MIU value of $\{\alpha \cup \beta\}$ is calculated. If the k-th highest MIU value is higher than min_util_{Border}, min_util_{Border} can be safely raised to that value.

Example 7. Consider the UP-Tree depicted in Fig. 2 and suppose $k = 4$. The node under the root is $N_{\{C\}}$. We traverse the sub-tree under the node $N_{\{C\}}$ once and calculate the MIU values of its descendents. For the descendent $N_{\{A\}}$, the total support count of {A} in the sub-tree of $N_{\{C\}}$ is $(1 + 2) = 3$. Therefore, the MIU of {AC} is $MIU(\{AC\}) = [miu(\{A\}) + miu(\{C\})] \times SC(\{AC\}) = [5+4] \times 3 = 27$. Table 6 shows the MIU values of descendents of $N_{\{C\}}$.

3.2.4 Raising the Threshold during Phase II

The fourth proposed strategy is called *SE* (*raising the threshold by Sorting and calculating Exact utility of candidates*), which is applied during the phase II of TKU.

Strategy 5 (SE: raising the threshold by Sorting and calculating Exact utility of candidates). Let C be the set of candidates produced in Phase I. *Candidates in C are sorted in descending order of their estimated utilities, i.e., min{ESTU(X), MAU(X)}.* Thus, candidates with higher estimated utility values will be considered before those having lower estimated utility values. During the phase II, if the utility of a newly considered HUI *X is larger than* min_util_{Border}, *X and EU(X)* are inserted into a min-heap structure named TopK-HUI-List. HUIs in *TopK-HUI-List are ordered by decreasing* utility. Then, min_util_{Border} is raised to the utility of the k-th HUI in *TopK-HUI-List, and HUIs having a utility lower than* min_util_{Border} are removed from *TopK-HUI-List. If the* estimated utility of the current candidate *Y, i.e., min{ESTU(Y), MAU(Y)}, is less than the raised* min_util_{Border}, *Y and the remaining* candidates do not need to be considered anymore because the upper bounds on their utilities are less than min_util_{Border}. *When the algorithm completes, TopK-HUI-List* captures all the top-k HUIs in the database.

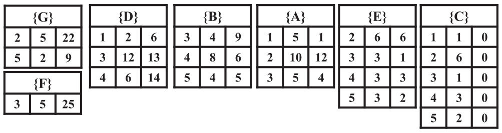
By this mechanism, itemsets with lower estimated utility values may not be checked if the min_util_{Border} has been previously raised. Thus, the I/O cost and execution time for Phase II can be further reduced.

4. THE TKO ALGORITHM

The second algorithm that we propose is *TKO* (*mining Top-k utility itemsets in One phase*). It can discover top-k HUIs in only one phase. It utilizes the basic search procedure of HUI-Miner and its utility-list structure [14]. Whenever an itemset is generated by TKO, its utility is calculated by its utility-list without scanning the original database. We first describe a basic version of TKO named TKO_{Base} and then the advanced version, which includes several strategies to increase its efficiency.

TABLE 7
Transactions for Constructing Utility-Lists

TID	Transaction	Transaction Utility (TU)
T_1	(D,1)(A,1)(C,1)	8
T_2	(G,5)(A,2)(E,2)(C,6)	27
T_3	(F,5)(D,6)(B,2)(A,1)(E,1)(C,1)	30
T_4	(D,3)(B,4)(E,1)(C,3)	20
T_5	(G,2)(B,2)(E,1)(C,2)	11



{G}		
2	5	22
5	2	9

{F}		
3	5	25

{D}		
1	2	6
3	12	13
4	6	14

{B}		
3	4	9
4	8	6
5	4	5

{A}		
1	5	1
2	10	12
3	5	4

{E}		
2	6	6
3	3	1
4	3	3
5	3	2

{C}		
1	1	0
2	6	0
3	1	0
4	3	0
5	2	0

Fig. 4. Initial utility-lists.

4.1 Construction of Utility-List Structure

In this section, we briefly introduce the utility-list structure and related properties. For details about utility-lists, readers are referred to [14]. In the TKO_{Base} and TKO algorithms, each item(set) is associated with a utility-list. The utility-lists of items are called *initial utility-lists*, which can be constructed by scanning the database twice. In the first database scan, the TWU and utility values of items are calculated. During the second database scan, items in each transaction are sorted in order of TWU values and the utility-list of each item is constructed.

Table 7 shows an example database, where items in each transaction are arranged in ascending order of TWU values. Fig. 4 shows utility-lists of items for the database in Table 7. The utility-list of an item(set) X consists of one or more tuples. Each tuple represents the information of X in a transaction T_r and has three fields: *Tid*, *iutil* and *rutil*. Fields *Tid* and *iutil* respectively contains the identifier of T_r and the utility of X in T_r. Field *rutil* indicates the remaining utility of X in T_r. The concept of remaining utility is based on the following definitions.

Definition 17 (Precede and succeed). *The ascending order of TWU is a total order such that an item I_i precedes an item I_j denoted as $I_i \prec I_j$ iff (1)$TWU(I_i) < TWU(I_j)$ or (2) $TWU(I_i) = TWU(I_j)$ and I_i is smaller than I_j according to the lexicographical order. If one of these conditions is not met, I_i is said to succeed I_j (denoted as $I_i \prec I_j$).*

Definition 18 (Concatenation of an itemset). *Let $X = \{x_1, x_2, \ldots, x_u\}$ $(x_i \in I^*, 1 \leq i \leq u)$ and $Y = \{y_1, y_2, \ldots, y_v\}$ $(y_j \in I^*, 1 \leq j \leq v)$ be itemsets, Y is a concatenation of X iff $X \subset Y$ and each item $y_j \notin X$ succeeds all items in X.*

Definition 19 (Appear after). *Given a finite set of items $I^* = \{I_1, I_2, \ldots, I_m\}$ and a total order $I_1 \prec I_2 \prec \ldots \prec I_m$ on all items (Definition 17). Suppose that items in itemsets and transactions are arranged according to this total order. An item $I_j \in I^*$ appears after an itemset $X = \{x_1, x_2, \ldots, x_L\}$ in a* *transaction T_r iff $I_j \in T_r$ and $x_1 \prec x_2 \prec \ldots \prec x_L \prec I_j$. The set of all the items appearing after X in T_r is denoted as T_r/X.*

Definition 20 (Remaining utility of an itemset in a transaction). *The remaining utility of an itemset X in a transaction T_r is defined as $RU(X, T_r) = \sum_{I_i \in T_r/X} EU(I_i, T_r)$.*

Definition 21 (Remaining utility of an itemset in a database). *The remaining utility of an itemset X in a database D is defined as $RU(X) = \sum_{T_r \in D} RU(X, T_r)$.*

Definition 22 (Utility-list structure). *The utility-list of an itemset X, denoted as ul(X) is a list containing $|g(X)|$ tuples of the form $\langle tid, iutil, rutil \rangle$. Each tuple is called an element and maintains the information of X in a transaction T_r, where $r \in g(X)$. An element with respect to the transaction T_r contains the information $\langle r, EU(X, T_r), RU(X, T_r) \rangle$.*

Example 8. Consider the database in Table 7. The remaining utility of {D} in T_1 is $RU(\{D\}, T_1) = EU(\{A\}, T_1) + EU(\{C\}, T_1) = (5+1) = 6$. The remaining utility of {D} in the database is $RU(\{D\}) = RU(\{D\}, T_1) + RU(\{D\}, T_3) + RU(\{D\}, T_4) = (6+13+14) = 33$. The remaining utility of {DE} is $RU(\{DE\}) = RU(\{DE\}, T_3) + RU(\{DE\}, T_4) = (1+3) = 4$.

Property 4. *In the utility-list of an itemset X, the sum of all its iutil values is the utility of X.*

Property 5. *Let X be an itemset and $Y \supset X$ be any concatenation of X. If the sum of all the rutil and iutil values in the utility-list of X is less than a threshold δ, the utility of Y must be less than δ.*

Rationale. If $X \subset Y$, $g(Y) \subseteq g(X)$. If a transaction T_r contains both X and Y, $(Y–X) \subseteq (T_r/X)$ and $(T_r/Y) \subseteq (T_r/X)$ (Definition 19). Let $\alpha = X \cup (T_r/X)$ and $\beta = Y \cup (T_r/Y)$. Because $Y = X \cup (Y–X)$, we have $\beta = X \cup (Y–X) \cup (T_r/Y)$. Since $(Y–X)$ and (T_r/Y) are subsets of (T_r/X), $(Y–X) \cup (T_r/Y) \subseteq (T_r/X)$, which yields $\beta \subseteq \alpha$. Besides, $EU(\alpha, T_r) = EU(X, T_r) + EU(T_r/X, T_r) = EU(X, T_r) + RU(X, T_r)$ and $EU(\beta, T_r) = EU(Y, T_r) + EU(T_r/Y, T_r) = EU(Y, T_r) + RU(Y, T_r)$. If $\beta \subseteq \alpha$, $EU(X, T_r) + RU(X, T_r) \geq EU(Y, T_r) + RU(Y, T_r) \geq EU(Y)$. Since $g(Y) \subseteq g(X)$, $EU(X) + RU(X) \geq EU(Y) + RU(Y) \geq EU(Y)$. If $(EU(X) + RU(X)) < \delta$, we have $EU(Y) < \delta$.

4.2 The TKO$_{Base}$ Algorithm

The TKO$_{Base}$ algorithm takes as input the parameter *k* and a transactional database *D* in horizontal format. But if a database has already been transformed into vertical format such as initial utility-lists, TKO$_{Base}$ can directly use it for mining top-*k* HUIs.

TKO$_{Base}$ initially sets the min_util_{Border} threshold to 0 and initializes a min-heap structure *TopK-CI-List* for maintaining the current top-*k* HUIs during the search. The algorithm then scans *D* twice to build the initial utility-lists Φ-*ULs*. Then, TKO$_{Base}$ explores the search space of top-*k* HUI using a procedure that we name *TopK-HUI-Search*. It is the combination of a novel strategy named *RUC* (*Raising threshold by Utility of Candidates)* with the *HUI-Miner* search procedure [14]. During the search, TKO$_{Base}$ updates the list of current top-*k* HUIs in *TopK-CI-List* and gradually raises the min_util_{Border} threshold by the information of *TopK-CI-List*. When the algorithm terminates, the *TopK-CI-List* captures the complete set of top-*k* HUIs in the database.

```
PROCEDURE: TopK-HUI-Search
  Input:  (1) u(P): utility-list for a prefix P;
          (2) Class[P]: a set of itemsets w.r.t. the prefix P;
          (3) ULS[P]: a set of utility-lists w.r.t.the prefix P;
          (4) δ: border minimum utility threshold $min\_util_{Border}$;
          (5) TopK-CI-List: a list for storing candidate itemsets;
Results:  (1) Use TopK-CI-List to capture all the top-k HUIs
01.  For each $X=\{x_1, x_2,\ldots, x_L\} \in Class[P]$ do
02.  {  If $(SUM(X.iutils) \geq \delta)$
03.     {  //Raise $min\_util_{Border}$ by the strategy RUC;
04.        $\delta \leftarrow RUC(X, TopK\text{-}CI\text{-}List)$;
05.     }
06.     If $(SUM(X.iutils) + SUM(X.rutils) \geq \delta)$
07.     {  $Class[X] \leftarrow \emptyset$; $ULS[X] \leftarrow \emptyset$;
08.        For each $Y = \{y_1, y_2,\ldots, y_L\} \in Class[P] \mid y_L \succ x_L$ do
09.        {  $Z \leftarrow X \cup Y$;
10.           $ul(Z) \leftarrow Construct(ul(P), X, Y, ULS[P])$;
11.           $Class[X] \leftarrow Class[X] \cup Z$;
12.           $ULS[X] \leftarrow ULS[X] \cup ul(Z)$;
13.        }
14.       TopK-HUI-Search(X, ULS[X], Class[X], δ, TopK-CI-List);
15.     }
16.  }
```

Fig. 5. The pseudo code of TopK-HUI-search.

For each *L*-itemset $X = \{x_1, x_2, \ldots, x_L\}$ generated by the search procedure, if its utility is no less than min_util_{Border}, the proposed *RUC* strategy is applied to raise min_util_{Border}. *RUC* is performed as follows. First, *X* is added into *TopK-CI-List*. Then, if *EU*(*X*) is no less than min_util_{Border} and there are more than *k* itemsets already in *TopK-CI-List*, min_util_{Border} is raised to the utility of the *k*-th itemset in *TopK-CI-List*. The remaining itemsets having a utility lower than min_util_{Border} are removed. This ensures that all and only the top-*k* HUIs are kept. After the above process, the strategy raises the border minimum utility threshold.

Fig. 5 shows the pseudo code of *TopK-HUI-Search* procedure. It continues mining itemsets that are concatenations of an itemset *X* if the sum of *iutils* and *rutils* of *X* is no less than min_util_{Border} (Line 6). Two ordered sets *Class*[*X*] and *ULS*[*X*] are created to respectively store the concatenations of *X* and their utility-lists (Line 7). For each itemset $Y = \{y_1, y_2, \ldots, y_L\}$ in *Class*[*P*] ($y_L \prec x_L$ and $P = \{y_1, y_2, \ldots, y_{L-1}\}$), we create a candidate itemset $Z = X \cup Y$ by concatenating *X* with y_L and uses the *Construct* procedure to construct utility-list of *Z* (i.e., *ul*(*Z*)) (Line 9-10). Then, *Z* and *ul*(*Z*) are respectively added to *Class*[*X*] and *ULS*[*X*] (Line 11-12). After processing each itemset in *Class*[*P*], the procedure *TopK-HUI-Search* is called with *X*, *Class*[*X*], min_util_{Border} and *TopK-CI-List* to consider (*L*+1)-itemsets that are concatenations of *X*. This recursive process continues until no candidate itemset is found.

Strategy 6 (RUC: Raising the threshold by the Utilities of Candidates). This strategy can be incorporated with any one-phase mining algorithm where itemsets are found with their utilities. It adopts the *TopK-CI-List* structure to maintain top-*k* HUIs, where itemsets are sorted by descending order of utility. Initially, *TopK-CI-List* is empty. When an itemset *X* is found by the search procedure and its utility is no less than min_util_{Border}, *X* is added to *TopK-CI-List*. If there are more than *k* itemsets already in *TopK-CI-List*, min_util_{Border} can be safely raised to the utility of the *k*-th itemset in *TopK-CI-List*. After that, itemsets having a utility lower than the raised min_util_{Border} are removed from *TopK-CI-List*.

Given two itemsets *X* and *Y* and their prefix *P*, during the search process of *TopK-HUI-Search*, the utility-list of the itemset $Z = X \cup Y$, denoted as *ul*(*Z*), is obtained by applying

TABLE 8
Characteristics of Datasets

Dataset	#Trans.	Avg. Length of Trans.	#Items	Type
Foodmart	4,141	4.4	1,559	Sparse
Retail	88,162	10.3	16,470	Sparse
Chainstore	1,112,949	7.2	46,086	Large
Mushroom	8,124	23	119	Dense
Chess	3,196	37	76	Dense
Accident	340,183	33.8	468	Dense
T12I8D100K	100,000	12	1,000	Sparse

the *Construc*t procedure. The procedure consists of two cases and is performed as follows.

Case 1. If $X = \{x_1\}$ and $Y = \{y_1\}$ are 1-itemsets, where $x_1 \prec y_1$. Let $Z = X \cup Y = \{x_1, y_1\}$ be a 2-itemset obtained by concatenating X with y_1. The utility-lists $ul(X)$ and $ul(Y)$ are constructed during the initial database scans. The utility-list of Z is obtained by the following process. For each transaction $T_r \in g(X) \cap g(Y)$, an element $\langle T_r, EU(Z, T_r), RU(Z, T_r)\rangle$ is created in $ul(Z)$, where $EU(Z, T_r)$ is the sum of the *iutil* values in elements associated with T_r in $ul(X)$ and $ul(Y)$, and where $RU(Z, T_r)$ is the *rutil* value associated with T_r in $ul(Y)$. In brief, $EU(Z, T_r) = EU(x_1, T_r) + EU(y_1, T_r)$ and $RU(Z, T_r) = EU(y_1, T_r)$.

Case 2. If $X = \{x_1, x_2, \ldots, x_{L-1}\}$ and $Y = \{y_1, y_2, \ldots, y_{L-1}\}$ are (L-1)-itemsets ($L \geq 2$), where $x_i = y_i$ ($1 \leq i < L$-1) and $x_{L-1} \prec y_{l-1}$. Let $Z = X \cup Y = \{x_1, x_2, \ldots, x_{L-1}, y_{L-1}\}$ be an L-itemset obtained by concatenating X with y_L. Let $P = X \cap Y = \{x_1, x_2, \ldots, x_{L-2}\}$ 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 $T_r \in g(X) \cap g(Y)$, an element $\langle T_r, EU(Z, T_r), RU(Z, T_r)\rangle$ is created in $ul(Z)$, where $EU(Z, T_r)$ is the sum of the *iutil* values in elements associated with T_r in $ul(X)$ and $ul(Y)$ minus the *iutil* value of the element associated with T_r in $ul(P)$, and where $RU(Z, T_r)$ is the *rutil* value associated to T_r in $ul(Y)$. In brief, $EU(Z, T_r) = [EU(X, T_r) + EU(Y, T_r)] - EU(P, T_r)$ and $RU(Z, T_r) = EU(Y, T_r)$.

4.3 The TKO Algorithm and Effective Strategies

We incorporate four strategies to improve the efficiency of TKO$_{\text{Base}}$. The resulting algorithm is named *TKO*. The first two strategies are *PE* and *DGU*, which have been previously presented in Section 3. The third and fourth strategies are based on the following definitions and properties.

Definition 23 (Z-element). *An element is called Z-element iff its rutil value is equal to zero. Otherwise, the element is called NZ-element. The set of all Z-elements in the utility list of X is denoted as ZE(X).*

For example, the utility list of {DBC} consists of two Z-elements $ZE(\{DBC\}) = \{\langle T_3, 17, 0\rangle, \langle T_4, 17, 0\rangle\}$.

Property 6. *Let NZEU(X) be the sum of iutil values of NZ-elements of an itemset X. If [NZEU(X) + RU(X)] < min_util$_{Border}$, all the concatenations of X are not top-k* HUIs.

Strategy 7 (RUZ: Reducing estimated utility values by using Z-elements). The RUZ strategy is applied during the generation of candidate itemsets in the *TopK-HUI-Search* procedure. For any candidate X generated by the TKO algorithm, it is not necessary to explore the search space of concatenations of X if *[NZEU(X) + RU(X)]* is less than *min_util$_{Border}$* (Property 6). This strategy is achieved by replacing Line 6 of Fig. 5 with the following code: If *[NZEU(X) + RU(X)]* $\geq \delta$.

TABLE 9
Strategies Used by the Algorithms

Algorithm	Phase I				Phase II
	PE	NU	MD	MC	SE
TKU	Y	Y	Y	Y	Y
TKU$_{\text{NoSE}}$	Y	Y	Y	Y	
TKU$_{\text{Base}}$				Y	Y

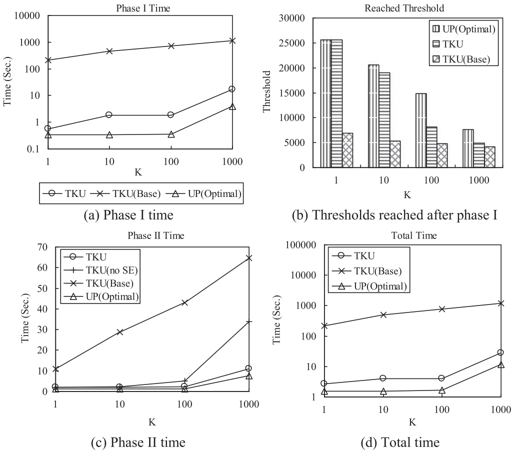
Strategy 8 (EPB: Exploring the most Promising Branches first). The EPB strategy aims at generating the candidate itemsets with the highest utility first. The reason is that if itemsets with higher utility are found earlier, TKO can raise its *min_util$_{Boarder}$* higher and earlier to prune the search space. Consider Lines 8–9 in Fig. 5, let Class*[P] = \{X$_1$, X$_2$, . . . , X$_M$\}* be the set of itemsets that share the same prefix P, where the last item of X_i precedes that of X_j $(1 \leq i < j \leq M)$. For each itemset X_j in Class*[P]*, let $R = \{X_{j+1}, X_{\_j+2}, \ldots, X_M\}$ denotes the itemsets in Class*[P]* whose last items precedes X_j. The TKO algorithm processes itemsets in R one by one in decreasing order of their estimated utility value (i.e., the sum of utility and remaining utility). The idea is to always try to extend the itemset having the largest estimated utility value first because it is more likely to generate itemsets having a higher utility and thus to allow to raise *min_util$_{Border}$* more quickly for pruning the search space.

5 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms. Experiments were performed on a computer with a 3.40 GHz Intel Core Processor and 4 GB of memory, running Windows 7. All the algorithms are implemented in Java. Both synthetic and real datasets were used in the experiments. Foodmart was acquired from Microsoft FoodMart 2000 database [40]; Retail, Mushroom, Chess and Accidents were obtained from the FIMI Repository [39]; Chainstore, a large dataset, was obtained from NU-MineBench 2.0 [18]. Foodmart and Chainstore already contain unit profits and purchase quantities. For other datasets, unit profits of items are generated between 1 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, as the settings of [25], [26], [27]. Synthetic datasets were generated from the data generator in [1]. Table 8 shows characteristics of the datasets.

5.1 Performance Evaluation of TKU$_{\text{Base}}$ and TKU

In this section, we compare the performance of TKU with UP-Growth [25] (one of the current best two-phase HUI mining algorithms). To evaluate the performance of the proposed strategies, we prepared three versions of TKU that we respectively name TKU, TKU$_{\text{NoSE}}$ and TKU$_{\text{Base}}$ as shown in Table 9. Because UP-Growth has not been designed for mining top-k HUIs, it cannot be directly compared with TKU. To compare them, we considered the scenario where users would choose the optimal parameters for UP-Growth to produce the same amount of patterns as TKU (denoted as UP(Optimal)).



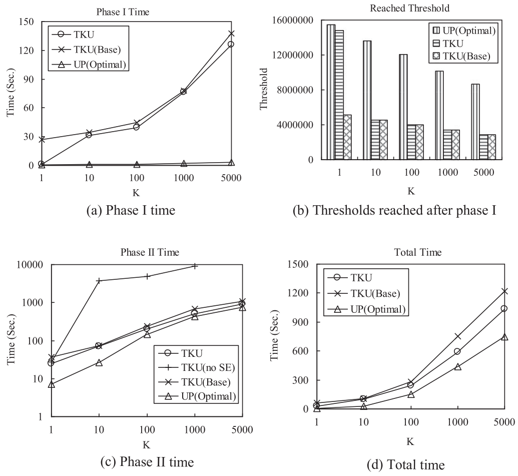
(a) Phase I time

(b) Thresholds reached after phase I

(c) Phase II time

(d) Total time

Fig. 6. Performance of the algorithms on Foodmart.



(a) Phase I time

(b) Thresholds reached after phase I

(c) Phase II time

(d) Total time

Fig. 7. Performance of the algorithms on Mushroom.

5.1.1 Performance Evaluation on Sparse Datasets

Fig. 6 shows the results on Foodmart. In Fig. 6a, the runtime of TKU for phase I is closed to that of UP(Optimal). TKU_{Base} has the worst performance among all the algorithms. Fig. 6b shows min_util_{Border} values reached by TKU and TKU_{Base} after completing phase I, as well as the optimal threshold values used by UP-Growth. In Fig. 6b, the min_util_{Border} values reached by TKU are closer to the optimal values than those reached by TKU_{Base}. This behavior is explained by the fact that TKU_{Base} does not apply the strategies PE, NU and MD. Thus, it constructs a full UP-Tree using $min\_util_{border} = 0$. Since raising the threshold for TKU_{Base} strictly depends on the *MC* strategy, its runtime is the longest. The ineffectiveness of raising the threshold for TKU_{Base} also influences the number of candidates generated in phase I. Table 10 shows the number of candidates generated by the algorithms in phase I. In Table 10, the number of candidates produced by TKU_{Base} is over 1,000 times larger than that of TKU when *k* is less than 1,000.

The reason is that strategies PE, NU and MD of TKU effectively raise the threshold at different stages. Fig. 6c. shows the runtime of the algorithms for Phase II. The performance of TKU_{NoSE} is worse than TKU because the latter uses the strategy SE, which reduces the number of candidates that need to be checked in Phase II. Fig. 6 d shows the overall runtimes of the algorithms. TKU is over 100 times faster than TKU_{Base}, and only about twice less than UP (Optimal).

5.1.2 Performance Evaluation on Dense Datasets

Fig. 7a shows the runtime of the algorithms for phase I. The runtime of TKU is close to that of TKU_{Base}. This is because for dense datasets the estimated utility values of itemsets are much larger than their utilities. Thus the thresholds cannot be raised effectively in phase I. Fig. 7b shows the thresholds reached by the algorithms. In Fig. 7b, when *k* is larger than 1, the thresholds reached by TKU are close to that of $\text{TKU}_{\text{Base.}}$ Table 10 shows the number of candidates generated by the algorithms in phase I.

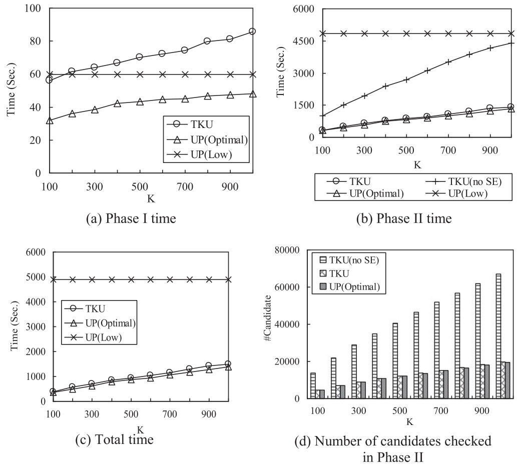
TABLE 10
Number of Candidates

	Mushroom		Foodmart	
K	TKU	TKU_{Base}	TKU	TKU_{Base}
1	427	508,462	1,379	2,466,459
10	597,301	713,793	1,503	2,494,446
100	803,377	920,040	2,456	2,537,225
1,000	1,540,583	1,657,403	39,289	2,585,300

In Table 10, we see that less candidates are produced by TKU than by TKU_{Base}. Fig. 7c shows the runtime of the algorithms for Phase II. The runtime of TKU_{NoSE} is the worst. This is because that, without using the SE strategy, TKU_{NoSE} needs to check all the candidates to determine which itemsets are top-*k* HUIs. When *k* is set to 5,000, the runtime of TKU_{NoSE} is too long to be executed (over 10,000 seconds). Fig. 7d shows the total runtime of the algorithms. In Fig. 7d, TKU is still more efficient than TKU_{Base}.

5.1.3 Performance Evaluation on Large Datasets

Fig. 8 shows the performance of the algorithms on a very large dataset Chainstore. Because the runtime of TKU_{Base} on this dataset is very slow (e.g., over 24 hours when $k = 1$), we instead use UP-Growth with a low minimum utility threshold (0.01 percent) as the baseline (denoted as UP(Low) in the experiments). The number of HUIs generated with $min\_util = 0.01\%$ is about 3,800. Fig. 8a shows the runtime of the algorithms for phase I. Since the threshold of UP (Low) is fixed, its runtime remains the same when *k* is



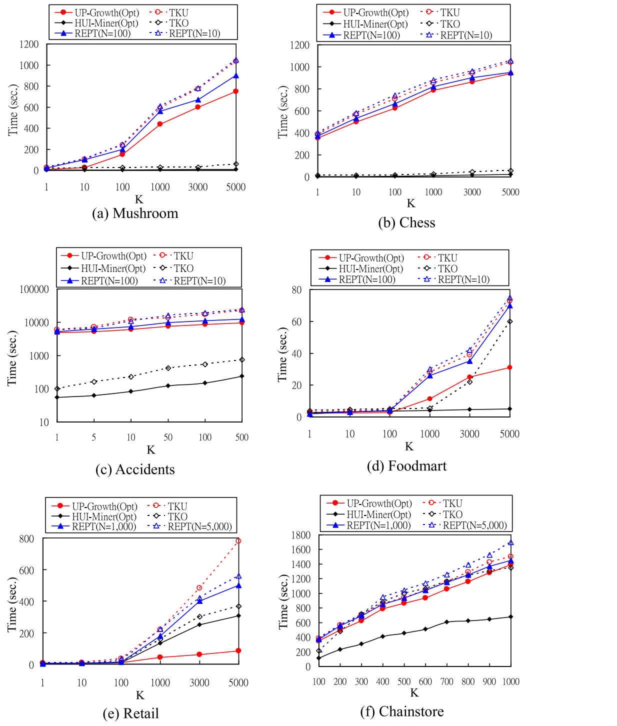
(a) Phase I time

(b) Phase II time

(c) Total time

(d) Number of candidates checked in Phase II

Fig. 8. Performance of the algorithms on Chainstore.



(a) Mushroom

(b) Chess

(c) Accidents

(d) Foodmart

(e) Retail

(f) Chainstore

Fig. 9. Runtime of REPT, TKU, and TKO.

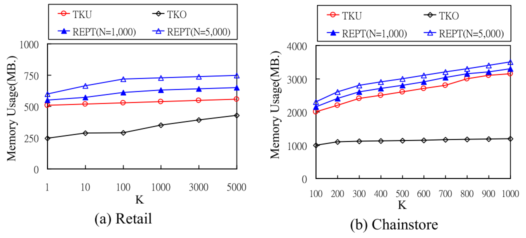
varied. In Fig. 8a, the runtime of TKU is worse than UP (Low) when $k > 200$. The reason is that TKU performs more operations to apply strategies to raise the threshold step by step. Fig. 8b shows the runtime for Phase II of the algorithms. In Fig. 8b, TKU is slightly slower than UP(Low). Fig. 8c shows the total runtime of the algorithms. Globally, TKU is much faster than UP(Low). This is because UP(Low) needs to check all candidates in Phase II, whereas TKU only needs to check some of them thanks to strategy SE. Fig. 8d shows the number of candidates checked by the algorithms in Phase II. Since TKU_{NoSE} checks all candidates, its performance is the worst. Besides, although TKU generates much more candidates in phase I, the number of candidates that need to be checked by TKU is close to that of UP(Optimal) in Phase II. This is because TKU avoids checking some candidates by using the SE strategy.

5.2 Performance Comparison of the REPT, TKU, and TKO Algorithms

In this section, we evaluate the performance of the proposed algorithms TKU and TKO against REPT [21] and two state-of-the-art HUI mining algorithms UP-Growth [25] and HUI-Miner [14]. Here, HUI-Miner(Opt) and UP-Growth (Opt) respectively represents HUI-Miner and UP-Growth tuned with the optimal thresholds. Besides, REPT with varied $N = y$ (i.e., the parameter for the RSD strategy [21]) is denoted as REPT($N = y$).

5.2.1 Performance Comparison on Dense Datasets

Figs. 9a, 9b and 9c show the runtime of the algorithms on three dense datasets Mushroom, Chess and Accidents with varied k respectively. In these figures, TKO has the best performance among top-k HUI mining algorithms. For example, on the Chess dataset, TKO only spends 23 seconds to complete the mining process, while REPT and TKU take more than 900 seconds. This is because TKO is a one-phase algorithm while TKU and REPT are two-phase algorithms. Because dense datasets generally contain lots of long itemsets and transactions, TKU and REPT tend to highly overestimate the upper bounds on utilities of generated candidates. However, whenever an itemset is produced by TKO, TKO immediately calculates its exact utility by the RUC and RUZ strategies, which allows TKO to efficiently and effectively raise the border thresholds. This avoids generating too many intermediate low utility or candidate itemsets during the mining process. On the contrary, whenever a candidate is generated by REPT or TKU in phase I, its exact utility is unknown. Thus, REPT and TKU cannot effectively raise the border minimum utility threshold and suffers from very long runtimes on dense datasets.



(a) Retail

(b) Chainstore

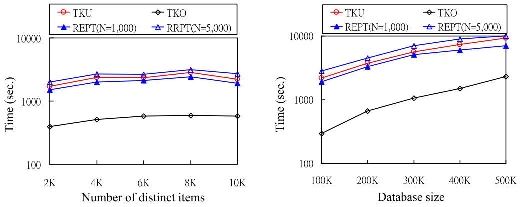
Fig. 10. Memory consumption of REPT, TKU, and TKO.

5.2.2 Performance Comparison on Sparse Datasets

Figs. 9d, 9e and 9f show the runtime of the algorithms on three sparse datasets Foodmart, Retail and Chainstore under varied k. In these figures, the one-phase algorithm TKO generally has the best performance. For two-phase algorithms, REPT runs slightly slower than TKU when N is set to 10. When a smaller N is set for REPT, the RSD strategy used in REPT cannot effectively raise the border minimum utility threshold and thus it produces more candidates and runs slower than TKU. When N is set appropriately, REPT may run faster than TKU. For example, on Retail dataset, when N is set to 1,000, REPT is faster than TKU. However, setting an appropriate N for REPT may be difficult for users who are not domain experts. Besides, the selection of N has major influence on the performance of REPT, especially on large datasets. For example, on the Chainstore dataset, when a too large N (i.e., $N = 5{,}000$) is set for REPT, REPT becomes very inefficient because it spends a lot of time enumerating 2-itemsets consisting of promising items from each transaction by using the RSD strategy. On the contrary, when a too small N is set (i.e., $N \leq 1{,}000$) for REPT, the RSD strategy used in REPT cannot effectively raise the threshold and causes REPT to suffer from a large number of candidates and a long runtime for Phase II.

5.3 Memory Usage of the Algorithms

Figs. 10a and 10b respectively show the memory usage of the algorithms on Retail and Chainstore. In Fig. 10, TKO generally uses less memory than TKU and REPT. This is because TKU and REPT are two-phase algorithms. When they could not effectively raise the border minimum utility



(a) Varied number of items

(b) Varied database sizes

Fig. 11. Scalability of the algorithms under different settings.

thresholds, they may consider too many candidates and local UP-Trees during the mining process, which causes them to consume much more memory than TKO. Besides, the memory consumption of REPT(N = 5,000) is higher than that of TKU. This is because REPT maintains not only a global UP-Tree in memory but also a RSD matrix. When there are many promising items and N is set too large for REPT, the RSD matrix could be very large and make REPT uses more memory.

5.4 Scalability of the Algorithms under Different Parameter Settings

Then, we test the scalability of the algorithms under different parameter settings. In the experiments, k is set to 5,000. Fig. 11a shows the runtime of the algorithms on T12I8D100KQ5 when the number of distinct items is varied from 2K to 10K. Fig. 11b shows the runtime of the algorithms on T12I8N1KQ5 when the database size is varied from 100K to 500K. As shown in Fig. 11, the proposed algorithms have good scalability under different parameter settings.

6 Conclusion and Future Works

In this paper, we have studied the problem of *top-k high utility itemsets mining*, where k is the desired number of high utility itemsets to be mined. Two efficient algorithms *TKU* (*mining Top-K Utility itemsets*) and *TKO* (*mining Top-K utility itemsets in One phase*) are proposed for mining such itemsets without setting minimum utility thresholds. TKU is the first two-phase algorithm for mining top-k high utility itemsets, which incorporates five strategies *PE*, *NU*, *MD*, *MC* and *SE* to effectively raise the border minimum utility thresholds and further prune the search space. On the other hand, TKO is the first one-phase algorithm developed for top-k HUI mining, which integrates the novel strategies *RUC*, *RUZ* and *EPB* to greatly improve its performance. Empirical evaluations on different types of real and synthetic datasets show that the proposed algorithms have good scalability on large datasets and the performance of the proposed algorithms is close to the optimal case of the state-of-the-art two-phase and one-phase utility mining algorithms [14], [25].

Although we have proposed a new framework for top-k HUI mining, it has not yet been incorporated with other utility mining tasks to discover different types of top-k high utility patterns such as *top-k high utility episodes*, *top-k closed*$^{+}$ *high utility itemsets*, *top-k high utility web access patterns* and *top-k mobile high utility sequential patterns*. These leave wide rooms for exploration as future work.

Acknowledgments

This work is supported in part by Ministry of Science and Technology, Taiwan, R.O.C. under grant 101-2221-E-006-255-MY3 and 103-2627-B-009-001, and by the US National Science Foundation (NSF) through grant CNS-1115234, and Google Research Award.

Reference

[1] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proc. Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.

[2] C. Ahmed, S. Tanbeer, B. Jeong, and Y. Lee, “Efficient tree structures for high-utility pattern mining in incremental databases,” *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.

[3] K. Chuang, J. Huang, and M. Chen, “Mining top-*k* frequent patterns in the presence of the memory constraint,” *VLDB J.*, vol. 17, pp. 1321–1344, 2008.

[4] R. Chan, Q. Yang, and Y. Shen, “Mining high-utility itemsets,” in *Proc. IEEE Int. Conf. Data Mining*, 2003, pp. 19–26.

[5] P. Fournier-Viger and V. S. Tseng, “Mining top-*k* sequential rules,” in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2011, pp. 180–194.

[6] P. Fournier-Viger, C. Wu, and V. S. Tseng, “Mining top-*k* association rules,” in *Proc. Int. Conf. Can. Conf. Adv. Artif. Intell.*, 2012, pp. 61–73.

[7] P. Fournier-Viger, C. Wu, and V. S. Tseng, “Novel concise representations of high utility itemsets using generator patterns,” in *Proc. Int. Conf. Adv. Data Mining Appl. Lecture Notes Comput. Sci.*, 2014, vol. 8933, pp. 30–43.

[8] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2000, pp. 1–12.

[9] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, “Mining top-*k* frequent closed patterns without minimum support,” in *Proc. IEEE Int. Conf. Data Mining*, 2002, pp. 211–218.

[10] S. Krishnamoorthy, “Pruning strategies for mining high utility itemsets,” *Expert Syst. Appl.*, vol. 42, no. 5, pp. 2371–2381, 2015.

[11] C. Lin, T. Hong, G. Lan, J. Wong, and W. Lin, “Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases,” *Adv. Eng. Informat.*, vol. 29, no. 1, pp. 16–27, 2015.

[12] G. Lan, T. Hong, V. S. Tseng, and S. Wang, “Applying the maximum utility measure in high utility sequential pattern mining,” *Expert Syst. Appl.*, vol. 41, no. 11, pp. 5071–5081, 2014.

[13] Y. Liu, W. Liao, and A. Choudhary, “A fast high utility itemsets mining algorithm,” in *Proc. Utility-Based Data Mining Workshop*, 2005, pp. 90–99.

[14] M. Liu and J. Qu, “Mining high utility itemsets without candidate generation,” in *Proc. ACM Int. Conf. Inf. Knowl. Manag.*, 2012, pp. 55–64.

[15] J. Liu, K. Wang, and B. Fung, “Direct discovery of high utility itemsets without candidate generation,” in *Proc. IEEE Int. Conf. Data Mining*, 2012, pp. 984–989.

[16] Y. Lin, C. Wu, and V. S. Tseng, “Mining high utility itemsets in big data,” in *Proc. Int. Conf. Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2015, pp. 649–661.

[17] Y. Li, J. Yeh, and C. Chang, “Isolated items discarding strategy for discovering high-utility itemsets,” *Data Knowl. Eng.*, vol. 64, no. 1, pp. 198–217, 2008.

[18] J. Pisharath, Y. Liu, B. Ozisikyilmaz, R. Narayanan, W. K. Liao, A. Choudhary, and G. Memik, NU-MineBench version 2.0 dataset and technical report [Online]. Available: http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, 2005.

[19] G. Pyun and U. Yun, “Mining top-*k* frequent patterns with combination reducing techniques,” *Appl. Intell.*, vol. 41, no. 1, pp. 76–98, 2014.

[20] T. Quang, S. Oyanagi, and K. Yamazaki, “ExMiner: An efficient algorithm for mining top-*k* frequent patterns,” in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2006, pp. 436 – 447.

[21] H. Ryang and U. Yun, “Top-*k* high utility pattern mining with effective threshold raising Strategies,” *Knowl.-Based Syst.*, vol. 76, pp. 109–126, 2015.

[22] H. Ryang, U. Yun, and K. Ryu, “Discovering high utility itemsets with multiple minimum supports,” *Intell. Data Anal.*, vol. 18, no. 6, pp. 1027–1047, 2014.

[23] B. Shie, H. Hsiao, V. S. Tseng, and P. S. Yu, “Mining high utility mobile sequential patterns in mobile commerce environments,” in *Proc. Int. Conf. Database Syst. Adv. Appl. Lecture Notes Comput. Sci.*, 2011, vol. 6587, pp. 224–238.
[24] P. Tzvetkov, X. Yan, and J. Han, “TSP: Mining top-*k* closed sequential patterns,” *Knowl. Inf. Syst.*, vol. 7, no. 4, pp. 438–457, 2005.
[25] V. S. Tseng, C. Wu, B. Shie, and P. S. Yu, “UP-Growth: An efficient algorithm for high utility itemset mining,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 253–262.
[26] V. S. Tseng, C. Wu, P. Fournier-Viger, and P. S. Yu, “Efficient algorithms for mining the concise and lossless representation of high utility itemsets,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3, pp. 726–739, Mar. 1, 2015.
[27] C. 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. IEEE Int. Conf. Data Mining*, 2011, pp. 824–833.
[28] J. Wang and J. Han, “TFP: An efficient algorithm for mining top-*k* frequent closed itemsets,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 5, pp. 652–663, May 2005.
[29] C. Wu, Y. Lin, P. S. Yu, and V. S. Tseng, “Mining high utility episodes in complex event sequences,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 536–544.
[30] C. Wu, B. Shie, V. S. Tseng, and P. S. Yu, “Mining top-*k* high utility itemsets,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 78–86.
[31] H. Xiong, M. Brodie, and S. Ma, “TOP-COP: Mining TOP-K strongly correlated pairs in large databases,” in *Proc. IEEE Int. Conf. Data Mining*, 2006, pp. 1162–1166.
[32] H. Xiong, P. Tan, and V. Kumar, “Mining strong affinity association patterns in data sets with skewed support distribution,” in *Proc. IEEE Int. Conf. Data Mining*, 2003, pp. 387–394.
[33] H. Xiong, P. Tan, and V. Kumar, “Hyperclique pattern discovery,” *Data Mining Knowl. Discovery*, vol. 13, no. 2, pp. 219–242, 2006.
[34] U. Yun and J. Kim, “A fast perturbation algorithm using tree structure for privacy preserving utility mining,” *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1149–1165, 2015.
[35] U. Yun and H. Ryang, “Incremental high utility pattern mining with static and dynamic databases,” *Appl. Intell.*, vol. 42, no. 2, pp. 323–352, 2015.
[36] J. Yin, Z. Zheng, L. Cao, Y. Song, and W. Wei, “Mining top-*k* high utility sequential patterns,” in *Proc. IEEE Int. Conf. Data Mining*, 2013, pp. 1259–1264.
[37] M. Zihayat and A. An, “Mining top-*k* high utility itemsets over data streams,” *Inf. Sci.*, vol. 285, no. 20, pp. 138–161, 2014.
[38] S. Zhu, J. Wu, H. Xiong, and G. Xia, “Scaling up top-*k* cosine similarity search,” *Data Knowl. Eng.*, vol. 70, no. 1, pp. 60–83, 2011.
[39] Frequent Itemset Mining Implementations Repository [Online]. Available: http://fimi.cs.helsinki.fi/
[40] FoodMart2000, Microsoft Developer Network (MSDN) [Online]. Available: https://technet.microsoft.com/en-us/library/aa217032(v = sql.80).aspx

Vincent S. Tseng is currently a professor in the Department of Computer Science, National Chiao Tung University. Currently, he also serves as the chair for IEEE Computational Intelligence Society Tainan Chapter. He served as the president in Taiwanese Association for Artificial Intelligence during 2011-2012 and acted as the director in the Institute of Medical Informatics of National Cheng Kung University (NCKU) during August 2008 and July 2011. During 2004 and 2007, he also served as the director of the Informatics Center, NCKU Hospital. He has a wide variety of research interests covering data mining, big data, biomedical informatics, multimedia databases, and mobile and web technologies. He has published more than 300 research papers in referred journals and international conferences as well as 15 patents held. He has been on the editorial board of a number of journals including the *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Journal on Biomedical and Health Informatics*, *ACM Transactions on Knowledge Discovery from Data*, etc. He has also served as chairs/program committee members for a number of premier international conferences related to data engineering artificial computational intelligence including KDD, ICDM, SDM, PAKDD, ICDE, CIKM, IJCAI, etc. He also received the 2014 K.T. Li Breakthrough Award. He is a senior member of the IEEE.

Cheng-Wei Wu received the PhD degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, in 2015. Currently, he is working as a postdoctoral researcher in the College of Computer Science, National Chiao Tung University, Taiwan. His research interests include data mining, utility mining, pattern discovery, machine learning, and big data analytics.

Philippe Fournier-Viger received the PhD degree in cognitive computer science from the University of Quebec in Montreal in 2010. He is an assistant professor at the University of Moncton, Canada. His research interests include data mining, e-learning, intelligent tutoring systems, knowledge representation, and cognitive modeling. He is the author of the popular SPMF data mining software.

Philip S. Yu received the BS degree in electrical engineering from National Taiwan University, the MS and PhD degrees in electrical engineering from Stanford University, and the MBA degree from New York University. He is a professor in the Department of Computer Science, University of Illinois at Chicago and also holds the Wexler chair in Information Technology. He spent most of his career at the IBM Thomas J. Watson Research Center and was a manager of the Software Tools and Techniques group. His research interests include data mining, database systems, and privacy. He has published more than 560 papers in refereed journals and conferences. He holds or has applied for more than 300 US patents. He is an associate editor of the *ACM Transactions on the Internet Technology* and *ACM Transactions on Knowledge Discovery from Data*. He is on the steering committee of the IEEE Conference on Data Mining and was a member of the IEEE Data Engineering steering committee. He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* (2001–2004). He had received several IBM honors including two IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, two Research Division Awards, and the 94th plateau of Invention Achievement Awards. He was an IBM Master Inventor. He received a Research Contributions Award from the IEEE International Conference on Data Mining in 2003 and also an IEEE Region 1 Award for “promoting and perpetuating numerous new electrical engineering concepts” in 1999. He is a fellow of the ACM and IEEE.