An Algorithm for Mining Fixed-Length High Utility Itemsets

Le Wang

Ningbo University of Finance & Economics, Ningbo Zhejiang 315175, China
wangleboro@gmail.com

Abstract. High utility pattern/itemset mining is a hotspot of data mining. Different from the traditional frequent pattern, high utility pattern takes into consideration not only the number of items in the transaction, but also the weight of these items, such as profit and price. Hence the computational complexity of this mining algorithm is higher than the traditional frequent pattern mining. Thus, one essential topic of this field is to reduce the search space and improve the mining efficiency. Constraint on pattern length can effectively reduce algorithm search space while fulfill a certain kind of actual requirement. Addressing fixed length high utility pattern mining, we propose a novel algorithm, called HUIK (High Utility Itemsets with K-length Miner), that first compresses transaction data into a tree, then recursively searches high utility patterns with designated length using a pattern growth approach. An effective pruning strategy is also proposed to reduce the number of candidate items on the compressed tree, to further reduce the search space and improve algorithm efficiency. The performance of the algorithm HUIK is verified on six classical datasets. Experimental results verify that the proposed algorithm has a significant improvement in time efficiency, especially for long datasets and dense datasets.

Keywords: Data mining, High utility itemsets, Pattern Growth, Frequent pattern.

1 Introduction

High Utility Pattern/Itemset (HUP / HUI) mining is a hot topic in data mining [1-5]. It is derived from frequent pattern mining, but it is different from frequent pattern mining. Frequent pattern mining takes each item of a transaction itemset as binary, i.e., it does not consider the internal utility value (quantity) and external utility value (such as importance, profit, price, etc.) of each item in a transaction itemset [6]. HUP mining introduces the internal utility value and external utility value of items into pattern mining. HUP has been applied to many fields, and its commercial value has been reflected in many applications, including website click stream analysis [7, 8], mobile commerce environment [9], retail store cross-marketing commercial value [10], genetic recombination and other applications [11].

Yao et al. [12] proposed the related definitions and mathematical model of HUP mining. The task of mining HUPs is to find all patterns whose utility value is not less

than a user-specified minimum utility value (threshold). The pruning strategy of traditional frequent pattern mining algorithms is not applicable in HUP mining because a superset of a low utility pattern may be a HUP, which makes search space of mining algorithms larger than frequent pattern mining, which makes HUP mining is much more difficult than frequent pattern mining.

Although there has been a lot of research on HUP mining [8, 13-17], there is still a relatively large search space. In order to provide the useful HUPs for users, Fournier-Viger et al. proposed a length-constrained HUPs mining algorithm FHM+ based on FHM [2, 18] , which can mine specified length HUPs, and it can effectively remove out some patterns that users are not interested in. When this algorithm calculates the estimated utility value of the patterns, the utility value of the transaction with an unsuitable length can be excluded, and the number of candidates can be reduced. Thus, this algorithm effectively improves its efficiency of mining HUPs.

Aiming at this kind of HUPs mining with length constraints, we propose a new mining algorithm HUIKM (High Utility Itemsets with K-length Miner) for improving the performance of this kind algorithms in this paper. This algorithm firstly maps a data to a tree, and then mines HUPs from the tree based on the method of pattern-growth; at the same time, an effective pruning strategy is given to reduce the search space. In the experiment, classical sparse and dense datasets are used to evaluate the performance of HUIKM. The experimental results show that the time efficiency of the algorithm has been greatly improved.

The contributions of this paper include:

- We designed a tree structure to maintain data information, from which the utility value of any specified-length HUP can be retrieved.
- We designed a new algorithm based on tree and pattern-growth for mining the specified-length HUPs.
- We performed an extensive experiment on classical datasets under different situations, and compared HUIKM with FHM+, EFIM and ULBMiner.

The structure of this paper is as follows: The second section gives the problem description & related definitions, and related works. The third section describes the proposed algorithm HUIKM. The fourth section conducts experimental tests. The fifth section gives the conclusions.

2 Background

In this section, we give the related definitions and work of the HUP mining.

2.1 Problem Description and Definitions

In this paper, we adopt definitions similar to those presented in the previous works [2, 7, 10, 13, 14, 18]. Given a *utility-valued transaction dataset* $D = \{T_1, T_2, T_3, \ldots, T_n\}$, which contains n transactions and m unique items $I = \{i_1, i_2, \ldots, i_m\}$. A transaction T_d ($d = 1, 2, 3, \ldots, n$) contains one or more unique items in I, and is also called a transaction itemset; e.g., $T_1 = \{(A,4)\ (C,3)\ (F,1)\}$. Each item i_j in a transaction T_d is attached with a quantity which is called *internal utility* (denoted as $q(i_j,T_d)$); e.g., $q(A,T_1) = 4$, $q(C,T_1) = 3$ and $q(F,T_1) = 1$ in table 1. An item i_j has a unit profit $p(i_j)$, which is called

external utility, e.g. p(A) = 4 in table 2. $|D|$ represents the size of the dataset D, i.e., the number of transactions in a dataset D; and $|T_d|$ represents the number of items in a transaction T_d, i.e., the length of transaction.

Table 1. An example of a transaction dataset

TID	Transaction	*TU*
T_1	(A,4) (C,3) (F,1)	47
T_2	(C,1)(D,4)(E,10)	58
T_3	(A,4)(B,4)(D,2) (E,6)	54
T_4	(A,1) (E,1)	6
T_5	(A,6)(B,2)(D,2) (E,1)	46
T_6	(A,3)(B,3) (D,1) (G,1)	30
T_7	(B,2) (G,2)	10
T_8	(A,3)(B,7) (C,1) (E,3)	49

Table 2. Profits

Item	Profit
A	4
B	3
C	10
D	7
E	2
F	1
G	2

Definition 1 The utility value of the item i_j in a transaction T_d is denoted as $U(i_j, T_d)$, and is defined as:

$$U(i_j, T_d) = \mathrm{p}(i_j) \times \mathrm{q}(i_j, T_d) \tag{1}$$

For example, in Table 1 and 2, $U(\mathrm{A}, T_1) = 4 \times 4 = 16$, $U(\mathrm{C}, T_1) = 10 \times 3 = 30$, and $U(\mathrm{F}, T_1) = 1 \times 1 = 1$.

Definition 2 The utility value of itemset X in a transaction T_d is denoted as $U(X, T_d)$, and is defined as:

$$U(X, T_d) = \begin{cases} \sum_{i_j \in X} U(i_j, T_d), & if\ X \subseteq T_d \\ 0, & else \end{cases} \tag{2}$$

For example, in Table 1and 2, $U(\{\mathrm{AC}\}, T_1) = 46$, $U(\{\mathrm{AF}\}, T_1) = 17$.

Definition 3 The utility value of itemset X in a dataset D is denoted as $U(X)$, and is defined as:

$$U(X) = \sum_{T_d \in D \wedge X \subseteq T_d} U(X, T_d) \tag{3}$$

For example, in Table 1and 2, $U(\{\mathrm{AC}\}) = U(\{\mathrm{AC}\}, T_1) + U(\{\mathrm{AC}\}, T_6) = 46 + 22 = 68$.

Definition 4 The utility value of transaction T_d is denoted as $TU(T_d)$, and is defined as:

$$TU(T_d) = \sum_{i_j \in T_d} U(i_j, T_d) \tag{4}$$

For example, in Table 1and 2, $TU(T_1) = U(\mathrm{A}, T_1) + U(\mathrm{C}, T_1) + U(\mathrm{F}, T_1) = 16 + 30 + 1 = 47$.

Definition 5 The utility of the dataset D is denoted as TU, and is defined as:

$$TU = \sum_{T_d \in D} TU(T_d) \tag{5}$$

For example, in Table 1 and 2, $TU = 47 + 58 + 54 + 6 + 46 + 30 + 10 + 49 = 300$.

Definition 6 The *transaction-weighted utility value* of itemset X is denoted as $TWU(X)$ (also called *TWU* value), and is defined as:

$$TWU(X) = \sum_{T_d \in D \wedge T_d \supseteq X} TU(T_d) \tag{6}$$

Definition 7 The *minimum utility threshold* δ is a user-specified percentile of total transaction utility values of the given dataset *D*; so the *minimum utility value*, *MinU* (also called a user-specified minimum utility value), is defined as:

$$MinU = TU \times \delta \tag{7}$$

Definition 8 An itemset is called a high utility pattern/itemset (HUP/HUI) if its utility value is not less than the minimum utility value. A HUP is called fixed-length pattern if its length meets a user-specified length *k*, is also denoted as HUPK/HUIK (High Utility Pattern/Itemset with *K*-length).

Given a transaction database *D*, the task of mining HUPK aims at finding all HUPKs from the dataset *D*. Mining HUPKs from a database also refers to finding all itemsets whose utility values are not less than a user-specified minimum utility value and whose lengths meet a user-specified value.

2.2 Related Work

There has been a lot of research on HUP mining [8, 13-17]. The most typical method is to mine HUPs by two phases. The first phase generates candidate itemsets of HUPs by over-estimating the utility value of patterns; in the second phase, the dataset is scanned to calculate the utility value of each candidate. The two-phase method often produces a large number of candidates in the first phase (even including the non-existing itemsets of dataset), which not only requires a lot of space to store candidates, but also causes a huge amount of calculation in the second stage to get the utility value of each candidate. The typical two-phase algorithms mainly include Two-Phase [13], IHUP [8], UP-Growth and UP-Growth+ [19], and MU-Growth [16], etc.

In order to avoid generating candidates, more mining algorithms have been proposed, such as HUI-Miner [14], FHM [2], HUP-Miner [17], ULB-Miner [20], and d2HUP [21]. These algorithms do not maintain a large number of candidates, and are also called one-phase algorithms. It is possible to directly calculate whether each pattern is a HUP. HUI-Miner firstly proposed the utility-list structure for mining HUPs. Then FHM, HUP-Miner, and ULB-Miner have been proposed based on the utility-list structure. The algorithm FHM applied a depth-first search to find HUPs, and was shown to be up to seven times faster than HUI-Miner. D2HUP [21] directly found HUPs using the method of pattern-growth, maintained a database using a hyper structure, and was shown to be up to one order of magnitude faster than the algorithm UP-Growth. Zida et al. [3] proposed a new algorithm EFIM, which applied two new upper bounds of utility values to reduce the search space of the algorithm, and U-List structure to maintain a dataset; the experimental results showed that the performance of EFIM has been greatly improved. ULB-Miner [20] extended FHM and HUI-Mine, reduced the memory and runtime usage of the algorithm by utilizing a utility list buffer structure. The time efficiency of mining HUPs has been continuously improved.

Based on HUPs mining, several variant algorithms have been proposed, e.g., high average-utility patterns mining [22, 23], Top-K high utility patterns mining [4, 24], and HUPs mining from data stream [25]. Most of these studies mainly apply methods of one-phase or two-phase.

Fournier-Viger et al. [18] proposed a length-constrained HUPs mining algorithm FHM+ based on FHM, which can mine specified length HUPs, and it can effectively

remove out some patterns that users are not interested in. When this algorithm calculates the estimated utility value of the patterns, the utility value of the transaction with an unsuitable length can be excluded, and the number of candidates can be reduced. Thus, this algorithm effectively improves its efficiency of mining HUPs.

FP- Growth is a good method for frequent pattern mining, which can compress data into a tree effectively and find all frequent patterns efficiently. Since the utility data contains more information than the data that FP-Growth processes, FP-Growth cannot be used to mine HUPs. However, it is a good method to compress the data into a tree, it can reduce search space and improve search efficiency. In this paper, we propose an algorithm HUIKM based on FP-Growth for mining fix-length HUPs.

3 Algorithm HUIKM

The algorithm HUIKM mainly consists of two steps, as shown in Algorithm 1: (1) the transaction dataset is mapped to a tree, (2)HUPKs are mined from the tree by a pattern-growth method.

Algorithm 1: The algorithm HUIKM

Input : D: transactions data;
$minutil$: the user-specified minimum utility value;
k: the user-specified length of high utility itemsets.

Output : HUIs

// create a Tree T and a header Table H

1 CreateGTree(D,$minutil$,k) ;

// find all HUIs,which lenth is k, from the Tree T

2 MHUIK(T,H,base-itemset,k) ;

3.1 Create a Tree and a Header Table

In the process of mining pattern, the algorithm needs to repeatedly scan the dataset to calculate the utility value of patterns. If the identical transactions in the dataset can be compressed together as much as possible, this can greatly reduce the search space of the algorithm. In order to be able to effectively compress transaction itemsets (as much as possible to compress the identical transactions or items to an identical node or branch on the tree), we propose an algorithm, called HUIKM. HUIKM streamlines each transaction itemset, deletes the non-candidate items of each transaction. The same transactions are compressed into a branch, and itemsets with the same preorders are also compressed together, which can effectively reduce the search space.

In order to further describe the algorithm HUIK and ensure the algorithm accuracy, we begin by giving the following definitions:

Definition 9 The maximum utility sum of k items in a transaction itemset T_d is called maximum transaction utility (denoted as $TUK(T_d, k)$), and is defined by

$$TUK(T_d,k)=\begin{cases}\max(\sum_{j=1}^{k}U(x_{i_j},T_d)\,|\,(1\le i_1<i_2<\cdots<i_k\le|T_d|)\wedge(x_{i_j}\in T_d)) & if\ |T_d|\ge k\\ 0, & else\end{cases} \quad (8)$$

For example, in Table 1 and 2, ***TUK*(*T*$_1$,1)** = max(*U*(A, *T*$_1$), *U*(C, *T*$_1$), *U*(F, *T*$_1$)) = max(16, 30, 1) = 30, ***TUK*(*T*$_1$,2)** = max(*U*(AC, *T*$_1$), *U*(AF, *T*$_1$), *U*(CF, *T*$_1$)) = max(46, 17, 31) = 46, ***TUK*(*T*$_1$,3)** = max(*U*(ACF, *T*$_1$,)) = max(47) = 47.

Definition 10 The maximum transaction utility of an itemset X in a transaction T_d is denoted as $TUK(X,T_d,k)$, and is defined by

$$TUK(X,T_d,k)=\begin{cases}U(X,T_d)+TUK(T_d-X,k-|X|) & if\ X\subseteq T_d\wedge|T_d|\geq k\\ 0, & else\end{cases} \quad (9)$$

For example, in Table 1 and 2, ***TUK*({C},*T*$_1$,2)** = *U*(C, *T*$_1$) + *TUK*({*T*$_1$ - {C},1) = 30 + max(*U*(A, *T*$_1$), *U*(F, *T*$_1$)) = max(16,1) = 46, ***TUK*({F},*T*$_1$,2)** = *U*(F, *T*$_1$) + *TUK*({*T*$_1$ - {F},1) = 1 + max(16,30) = 31, ***TUK*({C},*T*$_2$,2)** = *U*(C, *T*$_2$) + *TUK*({*T*$_2$ - {C},1) = 10 + max(28,20) = 38, ***TUK*({C},*T*$_8$,2)** = *U*(C, *T*$_8$) + *TUK*({*T*$_8$ - {C},1) = 10 + max(*U*(A, *T*$_8$), *U*(B, *T*$_8$), *U*(E, *T*$_8$)) = 10 + max(12, 21, 6) = 31.

Definition 11 The maximum transaction utility of an itemset X in a dataset D is called maximum transaction weight utility (denoted as $TWUK(X,k)$), and is defined by

$$TWUK(X,k)=\sum_{T_d\in D\wedge T_d\supseteq X} TUK(X,T_d,k) \quad (10)$$

For example, in Table 1 and 2, ***TWUK*({C},2)** = *TUK*({C},*T*$_1$,2) + *TUK*({C},*T*$_2$,2) + *TUK*({C},*T*$_8$,2) = 46 + 38 + 31 =115.

Definition 12 An itemset/item X is called a candidate (or promising itemset/item) for HUPK if its TWUK value is not less than a user-specified minimum utility value, otherwise it is an non-candidate (or unpromising itemset/item).

Definition 13 Given a transaction itemset $T_d=\{x_1,x_2,\cdots,x_i,\cdots\}$, and an ordered subset $X=\{x_i,x_{i_1},x_{i_2},\cdots,x_{i_j}\}$ of itemset T_d, then itemset $\{x_1,x_2,\cdots,x_{i\text{-}1}\}$ is named remain transaction-itemset of X in T_d; the maximum remain-transaction utility of an itemset X in a remain transaction-itemset T_d is denoted as ***RTWUK*(*X*,*T*$_d$,*k*)**, and is defined by

$$RTWUK(X,T_d,k)=\begin{cases}U(X,T_d)+TUK(\{x_1,x_2,...,x_{i-1}\},k-|X|) & if\ X\subseteq T_d\wedge|T_d|\geq k\\ 0, & else\end{cases} \quad (11)$$

The maximum remain-transaction utility of an itemset X in a dataset D is denoted as ***RTWUK*(*X*,*T*$_d$,*k*)**, and is defined by

$$RTWUK(X,k)=\sum_{T_d\in D\wedge X\in T_d} RTWUK(x_i,T_d,k) \quad (12)$$

For example, in Table 1 and 2, ***RTWUK*({C},*T*$_1$,2)** = *U*({C}, *T*$_1$)+ *TUK*({*T*$_1$ - {AC},1) = 30 + max(*U*(F, *T*$_1$)) = 31, ***RTWUK*({C},*T*$_2$,2)** = *U*({C}, *T*$_2$)+ *TUK*({*T*$_2$ - {C},1) = 10 + max(*U*(D, *T*$_2$), *U*(E, *T*$_2$)) = 10 + 28 = 38, ***RTWUK*({C},*T*$_8$,2)** = *U*({C}, *T*$_8$)+ *TUK*({*T*$_8$ - {ABC},1) = 10 + max(*U*(E, *T*$_8$)) = 10 + 6 = 16. ***RTWUK*({C}, 2) =** *RTWUK*({C},*T*$_1$,2) + *RTWUK*({C},*T*$_2$,2) + *RTWUK*({C},*T*$_8$,2) = 31+38+16 = 85.

Definition 14 An itemset/item X is called a candidate (or promising itemset/item) for HUPK if its RTWUK value is not less than a user-specified minimum utility value, otherwise it is an non-candidate (or unpromising itemset/item).

Property 1 Let *TWUK* or *RTWUK* value of an itemset X be not less than a user-specified minimum utility value, then any non-empty subset of X is also a promising

itemset; let *TWUK* or *RTWUK* value of an itemset *X* be less than a user-specified minimum utility value, then any superset of *X* is also an unpromising itemset.

Proof: An itemset Y_1 is any non-empty subset of *X*, and an itemset Y_2, whose length is less than *k*, is any superset of *X*. According to definition 11 and 13, $TWUK(Y_1,k) > TWUK(X,k) > TWUK(Y_2,k)$ and $RTWUK(Y_1,k) > RTWUK(X,k) > RTWUK(Y_2,k)$; thus, any non-empty subset of *X* is a promising itemset if TWUK or RTWUK value of itemset *X* is not less than a user-specified minimum utility value; any superset of *X* is an unpromising itemset if *TWUK* or *RTWUK* value of itemset *X* is less than a user-specified minimum utility value.

When the HUIKM algorithm creates the header table, it uses property 1 to find all candidates and save them in the header table. The *TWUK* value of property 1 is less than the *TWU* value, so property 1 can effectively reduce the number of candidates, i.e., reduce the subsequent search space. At the same time, the HUIKM algorithm uses property 1 to reduce the number of subsequent processing items when adding the transaction itemset to the tree, and saves the *RTWUK* value to the original *TWUK* value of the header table. The steps of HUIKM are as follows:

Step 1: Create a header table: calculate the TWUK value of each item by one scan of a dataset, maintain the candidates into a header table H, and sort them by the ascending order of TWUK value;

Step 2: The TWUK value of each item in the header table H is re-assigned to 0;

Step 3: Remove the non-candidates from each transaction itemset, sort remain items of each transaction in the order of the header table;

Step 4: If the length of the processed itemset is less than the user-specified length *k*, go back the above step to process next transaction itemset; otherwise, add this processed transaction itemset to a tree, and save the utility information to last node of the itemset. The *RTWUK* value and utility value of each item of this itemset is accumulated into the header table H.

The structure of a header table: "Item" records item name; "twuk/rtwuk" records the TWUK value of each item when creating the header table, and records the *RTWUK* value of each item when creating a tree; "utililty" records the utility value sum of each item and a base-itemset; "link" records the nodes of each item in a tree.

The tree node structure of the algorithm HUIKM is as follows: each node records the item name and child nodes, but the last node of each itemset on the tree also includes: "piu" is a list that records the utility of each item of the itemset that is from the node to the root node; "bu" records the utility value of the base itemset (see Example 2 in Section 3.2. The base item set is empty when the first tree is created. Thus, this value is not recorded when the first tree is created).

Example 1: We take the data in Table 1 and Table 2 as an example to illustrate the tree creation process (set the minimum utility threshold to 60 and k to 3):

(1) Create the header table as shown in Fig. 1(a).

(2) Process transaction T_1: After deleting the non-candidate "F" from the transaction itemset, the length of remaining itemset of the transaction (2) is less than the value of *k*(3). Thus, there is no need to add the processed itemset to the tree.

(3) Process transaction T_2: delete non-candidate items, and sort them to get the itemset {E, D, C} and their utility values, then add the itemset to the tree *T*, and save

the utility value of each item into the last node of this itemset (*piu*), the result is shown in Fig.1(b). When adding each item to the tree, the utility value of each item and the utility value of the base-itemset (*bu*) are accumulated to the utility of the header table (the initial value of the base-itemset is empty). When adding item "C" to the tree, its *RTWUK* value (58) is added to the *twuk* of the header table, and its node is added to the link of the header table; the result is shown in Fig.1(b).

(4) The transaction itemset T_3 is handled in the same way: the tree is shown in Fig. 1(c) after adding T_3 to the tree. When adding items "D" and "B", their *RTWUK* value (42) is added to the header table, and their nodes are also added to the link of the header table. The result is shown in the header table in Fig.1(c).

(5) The remaining transactions are added into the tree by the same way. When adding T_5 to the tree, only the utility values of the items are added to node "B", and the *RTWUK* values of "D" and "B" are accumulated to the header table.

(6) The result is shown in Fig.1(d) after adding all transactions to the tree.

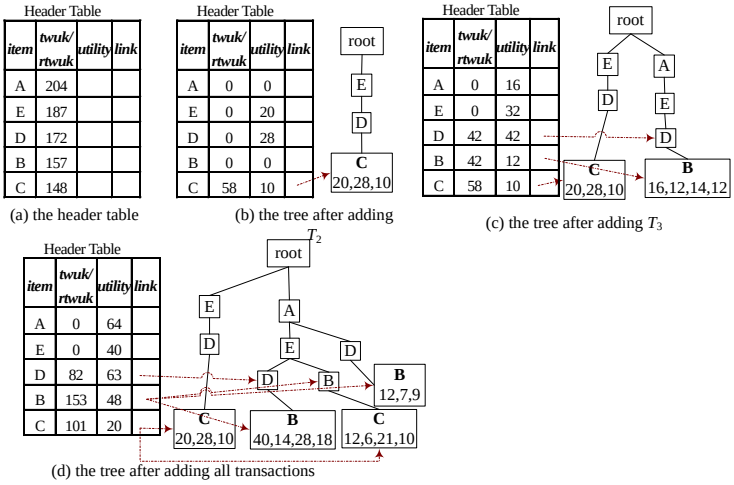


Fig.1 A case of creating a tree and a header table

The algorithm of creating the header table and tree is shown in Algorithm 2. First, create a header table via one scan of dataset. The processing steps are as follows:

(1) Calculate the TWUK value of each item (lines 2-6) and store it into a header table *H*;

(2) Remove these items, whose TWUK values are less than the minimum utility value, from the header table (line 7);

(3) Sort the items of the header table H by the descending order of TWUK values(line 8);

(4) Re-assign the TWUK value of each item in *H* to 0 (lines 9-11).

Second, add each transaction T_d to a tree. The processing steps are as follows:

(1) Remove non-candidates from a transaction itemset T_d (line 14);

(2) Process next transaction if the length of itemset T_d is less than *k* (line 15);

(3) Sort items of T_d by the order of H (line 16);

(4) Insert T_d to the tree, accumulate the *RTWUK* value of item in T_d into the header table H (line 19), accumulate the sum of utility values of item and base-itemset into H (line 20), and add the link of new node to H (line 21).

Algorithm 2: The CreateGTree Procedure

Input : D: transactions data, k: the length of high utility itemsets.
Output : a tree T

// First scan of the database D
1 Initiate a header table H containing the fields of item, $twuk$, and links;
2 **for** *each transaction* T_d *of* D **do**
3 **for** *each item* P *in* T_d **do**
4 $H.P.twuk+ = twuk(P, T_d, k)$;
5 **end**
6 **end**
7 Delete unpromising items from H based on threshold η;
8 Sort H according in the descending order of $twuk$ of H;
// Second scan of the database D
9 **for** *each item* P *in* H **do**
10 $H.P.twuk = 0$;
11 **end**
12 Initialize a Tree T with an empty root node;
13 **for** *each transaction* T_d *of* D **do**
14 Delete unpromising items from T_d;
15 if count of promising items in T_d is less than k then continue;;
16 Sort items of T_d according to H, with utility values, to X;
17 Insert X to T;
18 **for** *each item* P *in* X **do**
19 $H.P.twuk+ = rtwuk(P, X, k)$;
20 $H.P.utility+ = (U(P, X) + bu)$;
// bu: N.bu
21 New node is added to $H.P.link$;
22 **end**
23 **end**

3.2 Ming HUIK from a Tree

Algorithm HUIKM will process each item of the header table after creating a header table and a tree in section 3.1. It is necessary to determine whether the base-itemset of each item in the header table is a HUIK, and it is necessary to determine whether there is a HUI in its superset. If the superset may include a HUIK, it is necessary to create a sub tree and a sub header table, and then recursively process the new header table and tree. The steps of processing each item *P* in the header table are as follows:

Step 1: If the *RTWUK* value of item *P* is less than the minimum utility value, skip to step 4 otherwise, add item *P* to the base-itemset. The base-itemset is a HUIK if its length and utility values are not less than the user-specified values.

Step 2: If the length of the base-itemset is less than the user-specified value, create a new sub tree and a sub header table by scanning the tree via the link of item *P* in the header table, and recursively process the new header table and tree if the new header table is not empty.

Step 3: Remove the item *P* from the base-itemset;

Step 4: Process the utility information of item *P* as follows: accumulate or copy the utility of the base-itemset and list *piu* to parent node, and delete the nodes of item *P* from the tree.

Algorithm 3: MHUIK

```
   Input  : T: a tree, H: a header table, base-itemset, k: the length
            of high utility itemsets.
   Output : HUIs
 1 for each item P in H (with a bottom-up sequence) do
       // Step 1: Generate HUIs and create sub TN-tree
 2     if H.P.rtwuk ≥ min_util then
 3         base-itemset = base-itemset ∪{P};
 4         if |base − itemset| == k and H.P.utility ≥ min_util then
               // Calculate BU and NU
 5             Copy base-itemset to HUIs ; // generate one HUI
 6         end
 7         if |base − itemset| < k then
 8             Create a sub tree subT and a header table subH for
                base-itemset;
 9             MHUIK(subT, subH, base-itemset,k) ; // recursive call
10         end
11         Remove item P from itemset base-itemset;
12     end
       // Step 2: modify the utility information of the last item of itemset
13     foreach node N for item H.P in T do
14         remove utility value of item P from list N.piu;
15         if N.parent.bu==NULL then
16             N.parent.bu = N.bu;
17             N.parent.piu = N.piu;
18         else
19             N.parent.bu = N.parent.bu + N.bu;
20             N.parent.piu = N.parent.piu + N.piu;
21         end
22         Remove node N from T;
23     end
24 end
25 return HUIs;
```

Algorithm 3 is to mine HUIKs from a tree. The algorithm handles each item of a header table from the last item. The processing steps of each item consist mostly of two steps. (1) Determine whether each item will be a candidate for HUIKs (lines 2-12); (2) Move the utility information of nodes to the parent node (lines 13-23).

The first step consists mostly of the following sub-steps: (1) If the *RTWUK* value of an item is less than the minimum utility value *MinU*, move the utility information of nodes named this item and process the next item; otherwise, proceed to the next step. (2) Add the current item to the *base-itemset* (line 3； known as base itemset or prefix itemset). (3) If the length of the *base-itemset* equals the user specified minimum length value *k* and the utility of this *base-itemset* is not less than *MinU*, this *base-itemset* is a HUIK(lines 4-6). (4) If he length of the *base-itemset* is less than *k*, create a sub header table and a sub tree for *base-itemset* (or the current item), and recursively process the sub header table and the sub tree (lines 7-10). (5) Remove the current item from the *base-itemset*.

The second step consists the following sub-steps: (1) Delete the utility value of the current item from the list *piu* of nodes whose name is the current item. (2) If the

parent of the current nodes does not include the utility information, move the utility information of the current nodes to the parent node (line 16-17; refer to Example 2 for details); otherwise, accumulate the utility information of the current nodes to the parent node (lines 19-20). (3) Remove the current nodes from the tree (line 22).

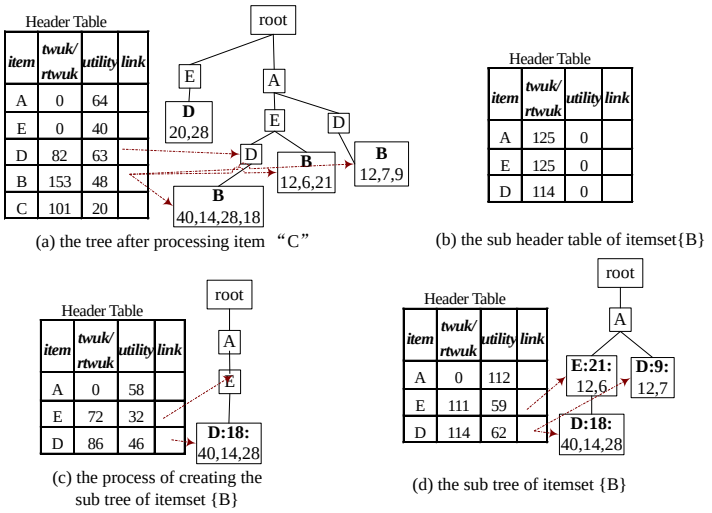


Fig.2 A case of creating a sub-tree and a header table

Example 2: We take the tree and header table in Fig.1(d) as an example to illustrate the steps of mining HUIKs from a tree. We will process each item of the header table from the last item. (1) The *RTWUK* value (101) of the item "C" is not less than 60, so add item "C" into the base-itemset (initial value is empty), i.e., base-itemset ={C}. (2) The base-itemset is not a HUIK because its length (1) is not equal to 3. (3) Since the length of the base-itemset is less than 3, create a sub header table and a tree for this base-itemset by the following: get two itemsets {E,D,C} and {A,E,B,C}, and their utility information by scanning the tree in Fig.1(d), and use these two itemsets and their utility information to create a sub header table. The sub header does not contain any items whose *TWUK* value is not less than 60. Thus, we do not need to create a sub tree. (4) Remove item "C" from the base-itemset. (5) Remove the utility the base-itemset and list *piu* to parent node (nodes "D"and "B"), the result is shown in Fig.2(a). (6) Remove the nodes "C" from the tree.

We process the item "B" of the header table by the same way: (1) Create a sub header table by scanning three branches on the tree, the result is shown in Fig. 2(b). (2) Get the first itemset {A,E,D,B} from the branch in Fig.2(a), sort the itemset by the order of the sub header table, and add this ordered itemset to a sub tree, the result is shown in Fig. 2(c), the utility list of the itemset {40,14,28} and the utility of the base-itiemset (bu) are stored into the node "D" in Fig. 2(c)； we do not modify the *RTWU* value of in the sub header table and modify the utility value (58=40+18) of item "A" in the header table when the item "A" of this itemset is added to the tee; modify the RTWUK value (72=40+14+18) and the utility value (32=14+18) in the header table when the item "E" of this itemset is added to the tee; modify the RTWUK value (86=40+28+18) and the utility value (46=28+18) in the header table when the item

"D" of this itemset is added to the tee. (3) According to Algorithm 3, we recursively process the header table in Fig.2(d), and mine 3 HUIKs ({B,D,E:62}, {B,D,A:114}, {B,E,A:111}). (4)After processing the item "B" in Fig. 2(a), process the next item "D" of the header table in Fig.2(a) and get a HUIK {D,E,A:82}. (5) We do not need to process the other items "A" and "E" in the header table in Fig.2(a) since the *RTWUK* values of these two items are less than 60. Finally, we find four HUIKs from the dataset: {B,D,E:62},{B,D,A:114}, {B,E,A:111} and {D,E,A:82}.

4 Experimental Results

Table 3. Dataset characteristics

Dataset	Distinct items(#)	Avg. trans. length (AS)	Transactions (#)	Type
Chainstore	46,086	7.2	1,112,949	Very sparse, short transactions
Pumsb	2,111	74	49,046	Sparse, very long transactions
Retail	16,470	10.3	88,162	Very sparse, short transactions
Connect	129	43	67,557	Dense, long transactions
Mushroom	119	23	8,124	Dense, moderately long transactions
Chess	76	37	3,196	Dense, long transactions

In order to assess the performance of HUIKM, we compare the performance of HUIKM with FHM+ [18], EFIM [3] and ULBMiner [20]. But EFIM and ULBMiner mine all HUIs; so we firstly mine HUIs using EFIM and ULBMiner, and then find fixed-length HUIs. We call the revised algorithms EFIM_k and ULBMiner_K respectively. All algorithms are implemented in JAVA programming language. FHM+, EFIM and ULBMiner are downloaded from the SPMF website (http://www.philippe-fournier-viger.com/spmf/) [26]. The experimental data are six classical datasets downloaded from the SPMF website. The characteristics of these six datasets are shown in Table 3. Experimental platform: Windows 7 operating system, 16G memory, Intel(R) Core(TM) i7-6500 CPU @ 2.50 GHz.

In each experiment, the four algorithms find the same HUIKs, and the results are shown in Fig. 3. According to Theorem 1, deleting non-candidates from a header table or a sub header table does not result in the loss of HUIKs. Algorithm HUIKM uses pattern-growth for mining HUIKs; when processing header table or sub header table, the items that have been already processed do not appear again in subsequent processing, so the processed items do not need to be considered, that is, the *RTWUK* value is the *TWUK* value that does not include the utility value of processed items. Thus, algorithm HUIKM can find all HUIKs.

The experiment firstly compared the running time of the two algorithms on six datasets. On each dataset, the larger the fixed-length *k* value, the larger the *TWUK* value of each item, the more candidates will be retained in the header table, the greater the amount of computation the algorithm will be and the more time-consuming the algorithm will be. There are generally fewer long patterns on sparse datasets, while there are relatively many long patterns on dense datasets. In order to compare the running time of the algorithm under different *k* values, *k* takes 2 and 3 on two sparse

datasets, and 4 and 6 on dense datasets, respectively. The higher the *k* value, the more items the header table will have, i.e. the greater the search space of the algorithm, the less efficient the algorithm will run, as shown in Fig. 4. But the larger the value of k, the more patterns will not be, as shown in Fig. 3(a). For example, on the dataset Chainstore, the higher *k* value, the fewer HUIKs produced. It is obvious from Fig.4 that the proposed algorithm is more efficient than FHM under various *k* values.

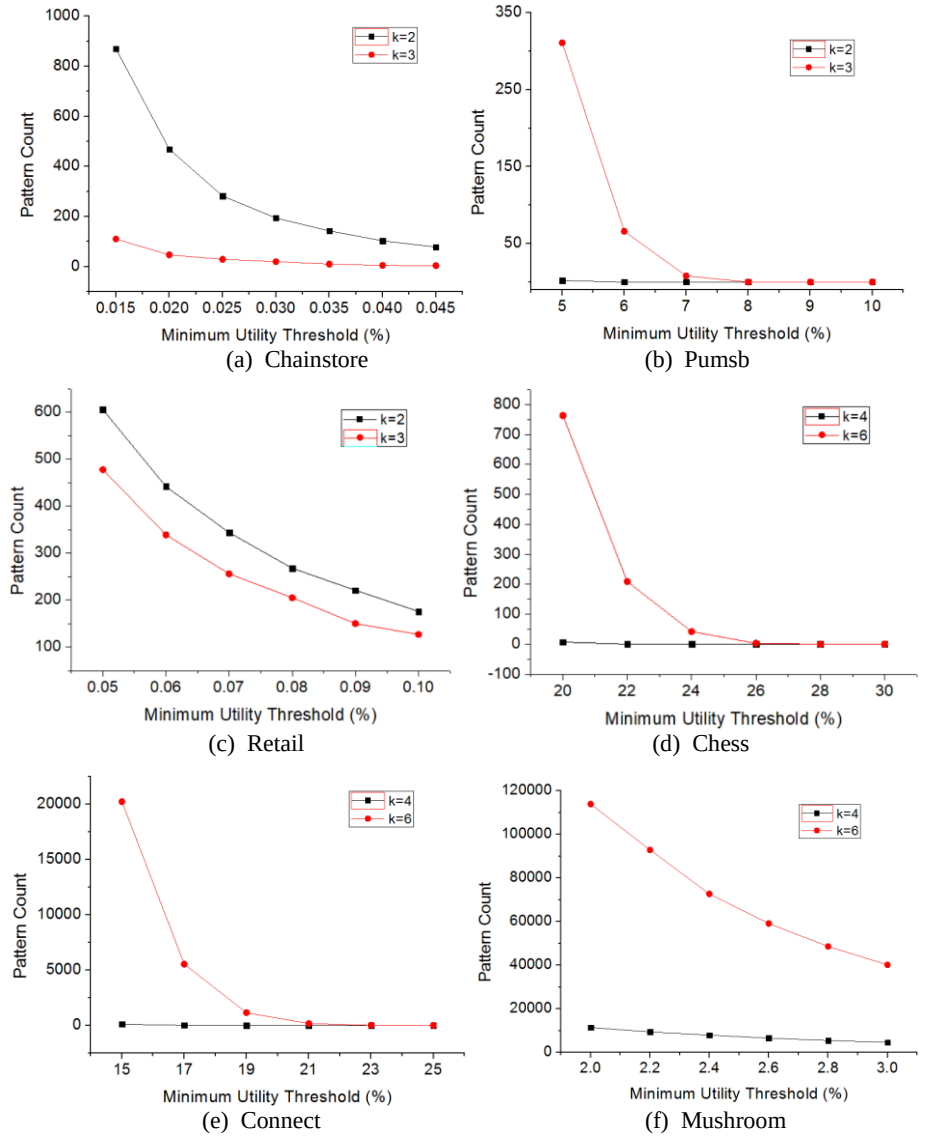
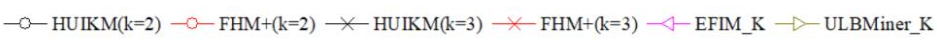


Fig. 3 Number of Patterns



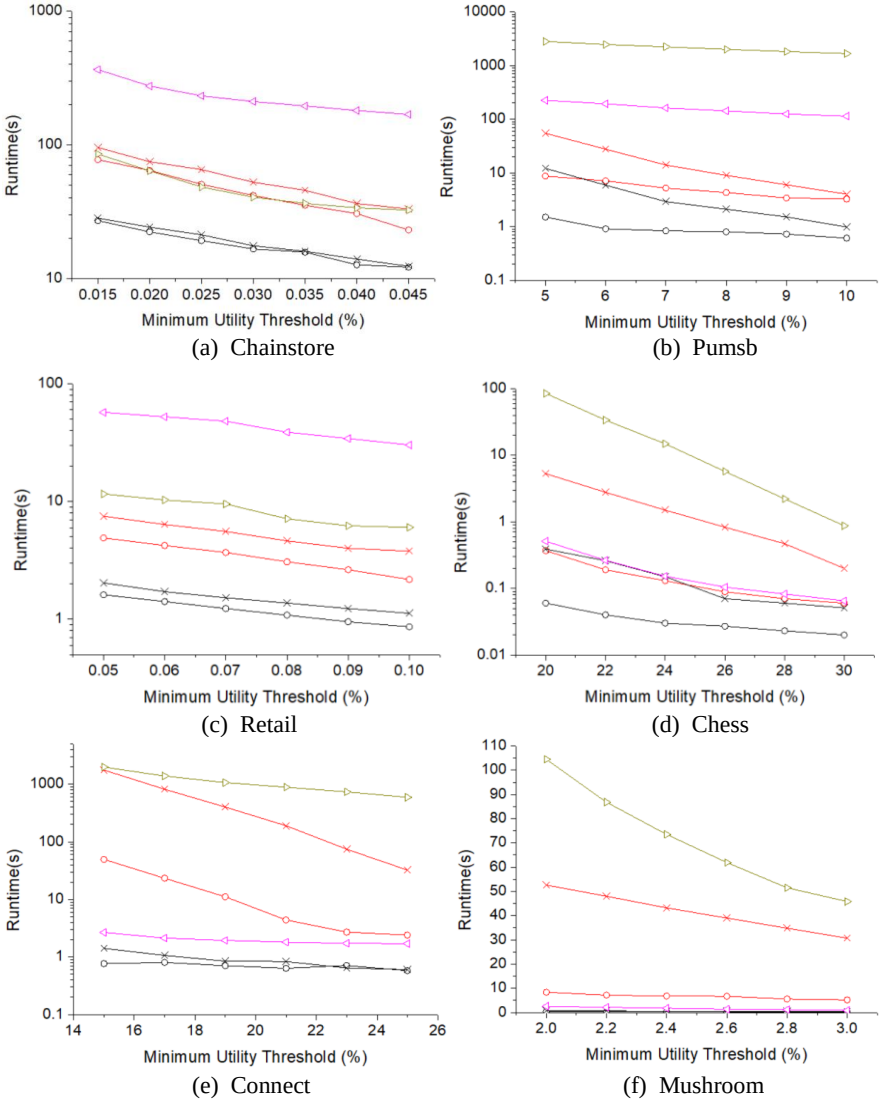


Fig. 4 Running time on different datasets

Fig. 4 shows the running time of two algorithms under minimum utility thresholds and various *k*-values. It is obvious from Fig.4 that HUIKM is more efficient than FHM under various *k* values and various minimum utility thresholds. The efficiency of the algorithm is increased to 2-3 orders of magnitude on dense datasets. HUIKM can reduce the search space and improve the performance by the following methods: (1) it uses the *TWUK* values to create header tables, and the *TWUK* value is not more than the *TWU* value, so this can obviously reduce the number of items of the header table; (2) Items closing to the root node is not considered for mining HUIKs; (3)

HUIKM uses the *RTWUK* value instead of *TWUK* value to determine if there will be one HUIK or more HUIKs. HUIKM effectively reduces the number of items of the header table using three methods above, and effectively reduces the amount of computation. At the same time, it can also be seen from Fig. 4 that the running time of HUIKM increases with the decrease of threshold, the increase is relatively flat, especially on the dataset containing long transactions, HUIKM reduces the number of items in the header table more obviously in the long dataset, and HUIKM can effectively compress the data into the tree on the dense dataset, thereby reducing the search space of the algorithm, and the running time can be increased to 2-3 orders of magnitude on the dense dataset, as shown in Figure 4 (c).

5 Conclusion

In this paper, we propose an algorithm for mining HUIKs through the study of the HUP mining algorithm. This algorithm firstly compresses a data effectively into a tree, and then uses the method of pattern-growth to recursively find all HUIKs; HUIKM effectively reduces the search space and improves the efficiency of the algorithm since this algorithm uses the tree structure for maintaining the data and uses the *RTWUK* value to reduce the number of header tables. In our experiments, we compare the performance of HUIKM with FHM+, EFIM and ULBMiner on six classical datasets. The experimental results show that the running time of HUIKM is greatly improved on different datasets, especially on the long and dense datasets, which can be increased to 2-3 orders of magnitude.

But HUIKM costs much time in creating sub trees and header table, we hope to further optimize it in the next step. In the future, we will also extend the approach in sequential data to reveal high utility sequential pattern.

Acknowledgement

This work is partially supported by the Zhejiang Philosophy and Social Science Project (19GXSZ49YB).

References

[1]. Lin, C., G. Lan and T. Hong, Mining high utility itemsets for transaction deletion in a dynamic database. Intelligent Data Analysis, 2015. 19(1): p. 43-55.
[2]. Fournier-Viger, P., et al., FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning, in Foundations of Intelligent Systems. 2014, Springer. p. 83-92.
[3]. Zida, S., et al., EFIM: A fast and memory efficient algorithm for high-utility itemset mining. Knowledge and Information Systems, 2017. 51(2): p. 595-625.
[4]. Han, X., et al., Efficient top-k high utility itemset mining on massive data. Information Sciences, 2020. In Press.
[5]. Verma, A., et al., High-utility and diverse itemset mining. Applied Intelligence, 2021: p. 1-15.

[6]. Agrawal, R., T. Imielinski and A. Swami. Mining association rules between sets of items in large databases. in ACM SIGMOD International Conference on Management of Data. 1993. Washington, DC, United states: Publ by ACM.
[7]. Li, H., et al. Fast and memory efficient mining of high utility itemsets in data streams. in 2008 eighth IEEE international conference on data mining. 2008: IEEE.
[8]. Ahmed, C.F., et al., Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering, 2009. 21(12): p. 1708-1721.
[9]. Shie, B., H. Hsiao and V.S. Tseng, Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments. Knowledge and information systems, 2013. 37(2): p. 363-387.
[10]. Li, Y., J. Yeh and C. Chang, Isolated items discarding strategy for discovering high utility itemsets. Data & Knowledge Engineering, 2008. 64(1): p. 198-217.
[11]. Zihayat, M., H. Davoudi and A. An, Mining significant high utility gene regulation sequential patterns. BMC systems biology, 2017. 11(6): p. 109.
[12]. Yao, H., H.J. Hamilton and G.J. Butz. A foundational approach to mining itemset utilities from databases. in 4th SIAM International Conference on Data Mining (ICDM 2004). 2004. Lake Buena Vista, FL, United states.
[13]. Liu, Y., W.K. Liao and A. Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. in 9th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. 2005. Hanoi, Viet nam.
[14]. Liu, M. and J. Qu. Mining high utility itemsets without candidate generation. in 21st ACM International Conference on Information and Knowledge Management (CIKM 2012). 2012. Maui, HI, United states: Association for Computing Machinery.
[15]. Lan, G., et al., Applying the maximum utility measure in high utility sequential pattern mining. Expert Systems with Applications, 2014. 41(11): p. 5071-5081.
[16]. Yun, U., H. Ryang and K.H. Ryu, High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Systems with Applications, 2014. 41(8): p. 3861-3878.
[17]. Krishnamoorthy, S., Pruning strategies for mining high utility itemsets. Expert Systems with Applications, 2015. 42(2015): p. 2371 – 2381.
[18]. Fournier-Viger, P., et al. FHM + : Faster high-utility itemset mining using length upper-bound reduction. in International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. 2016: Springer.
[19]. Tseng, V.S., et al., Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Transactions on Knowledge and Data Engineering, 2013. 25(8): p. 1772-86.
[20]. Duong, Q., et al., Efficient high utility itemset mining using buffered utility-lists. Applied Intelligence, 2018. 48(7): p. 1859-1877.
[21]. Liu, J., K. Wang and B.C. Fung. Direct discovery of high utility itemsets without candidate generation. in 2012 IEEE 12th international conference on Data Mining. 2012: IEEE.
[22]. Kim, J., et al., One scan based high average-utility pattern mining in static and dynamic databases. Future Generation Computer Systems, 2020. 111: p. 143-158.
[23]. Truong, T., et al., Efficient high average-utility itemset mining using novel vertical weak upper-bounds. Knowledge-Based Systems, 2019. 183: p. 104847.
[24]. Krishnamoorthy, S., Mining top-k high utility itemsets with effective threshold raising strategies. Expert Systems with Applications, 2019. 117: p. 148-165.
[25]. Nam, H., et al., Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions. Information Sciences, 2020. 529: p. 1-27.
[26]. Fournier-Viger, P., et al. The SPMF open-source data mining library version 2. in Joint European conference on machine learning and knowledge discovery in databases. 2016: Springer.