A fast maintenance algorithm of the discovered high-utility itemsets with transaction deletion

Jerry Chun-Wei Lin\textsuperscript{a,b,\ast}, Wensheng Gan\textsuperscript{a} and Tzung-Pei Hong\textsuperscript{c,d}
\textsuperscript{a}Innovative Information Industry Research Center, Shenzhen, Guangdong, China
\textsuperscript{b}School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, HIT Campus Shenzhen University Town, Shenzhen, Guangdong, China
\textsuperscript{c}Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan
\textsuperscript{d}Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan

Abstract. High-utility itemset mining (HUIM) has been recently studied to mine high-utility itemsets (HUIs) from the transactional database by considering more factors such as profit and quantity. Many approaches have been proposed for HUIM from a static database. Fewer studies have been developed to maintain the discovered HUIs in dynamic environment whether transaction insertion or transaction deletion. In the past, the FUP-HUI-DEL and PRE-HUI-DEL algorithms were respectively proposed to effectively maintain the discovered high transaction-weighted utilization itemsets (HTWUIs) and high-utility itemsets (HUIs) when the transactions are consequentially deleted from the original database. The original database is still, however, required to be rescanned when small transaction-weighted utilization itemsets in the original database are necessary to be maintained. In this paper, an efficient algorithm namely HUI-list-DEL is presented to discover HUIs by maintaining the built utility-list structure for transaction deletion in dynamic databases. Based on the designed algorithm, the HUIs can be directly produced without candidate generation or the numerous database scans. Two pruning strategies are also designed to speed up the maintenance approach of HUIs. Substantial experiments show that the proposed maintenance approach for transaction deletion significantly outperforms the previous approaches in terms of execution time, memory consumption and scalability.

Keywords: High-utility itemsets, transaction deletion, utility-list, maintenance, dynamic database

1. Introduction

The main purpose of Knowledge Discovery in Database (KDD) is to discover meaningful and useful information from a collection of data. Association Rule Mining (ARM) \cite{2,3,7,16} is the fundamental way to discover the occurrence frequency of the itemsets from the binary databases. However, only the occurrence frequency of the itemsets are considered to mine frequent itemsets or association rules but other significant factors such as profit, weight, and quantity are not concerned in ARM, which is

\textsuperscript{\ast}Corresponding author: Jerry Chun-Wei Lin, School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, Guangdong 518055, China. E-mail: jerrylin@ieee.org.

1088-467X/16/$35.00 \copyright \ 2016 – IOS Press and the authors. All rights reserved
insufficient to identify high profitable items for retailers. To address the above limitation of ARM, high-utility itemset mining (HUIM) [6,31,32] was thus proposed to reveal high-utility itemsets (HUIs). Chan et al. proposed the approach to find the top-$k$ closed utility patterns based on business purpose [6]. Yao et al. designed the utility model to concern both quantities (internal utility) and profits (external utility) of the items for producing HUIs [31]. Since the downward closure (DC) property of ARM is not held for HUIM, Liu et al. then presented the Two-Phase model [27] to discover HUIs based on the designed transaction-weighted downward closure (TWDC) property. Several algorithms have been proposed to efficiently mine HUIs based on Two-Phase model [27] or tree-based model and most of them are developed for mining HUIs from the static databases [5,23,29].

When database is changed whether transaction insertion [9,18,22] or deletion [10,19], the above approaches process the updated database in batch mode, thus requiring the multiple database scans and discarding the already discovered information. In the past, several algorithms have been proposed to maintain and update the discovered frequent itemsets of ARM while transactions are deleted [10,19]. For the HUIM, Lin et al. then developed the FUP-HUI-DEL [25] and the PRE-HUI-DEL [21] algorithms to maintain and update the discovered high transaction-weighted utilization itemsets (HTWUIs) and HUIs with transaction deletion. Original database is, however, required to be rescanned if the small itemsets (not existing in HTWUIs) are required to be maintained and determined for the updated database. Besides, since the FUP-HUI-DEL and PRE-HUI-DEL algorithms are processed in a level-wise way based on the Apriori-like approach, numerous computations are still required to handle the discovered HTWUIs level-by-level.

In this paper, a HUI-list-DEL algorithm with transaction deletion is proposed to efficiently maintain and update the built utility-list structure for directly mining HUIs without candidate generation or extra database scans. Two pruning strategies are also developed in this paper to speed up the computations for mining HUIs. Based on the designed model, key contributions are stated as follows.

1. Only the built utility-list structure is required to be maintained and updated for transaction deletion whenever necessary. Besides, the proposed algorithm is unnecessary to rescan the original database for producing the updated HUIs.
2. Two situations that some transactions are completely deleted from the original database, or some transactions may be partially deleted from the original database, can be successfully handled by the designed algorithm.
3. The designed early pruning strategy can be used to terminate the depth-search procedure, thus significantly reducing the search space for mining HUIs. Besides, the estimated utility co-occurrence structure (EUCS) strategy is also applied in the proposed maintenance algorithm to avoid the numerous computations for the generation-and-test approach of 2-itemsets.
4. Experimental results showed that the proposed HUI-list-DEL algorithm has better performance than the state-of-the-art batch-mode algorithms and the previous maintenance algorithms for transaction deletion.

The rest of this paper is organized as follows. Related works are briefly reviewed in Section 2. Preliminaries and problem statement of HUIM with transaction deletion are given in Section 3. The proposed HUI-list-DEL maintenance algorithm is stated in Section 4. An example to illustrate the proposed algorithm is described in Section 5. Experimental evaluation is provided and summarized in Section 6. Conclusions are finally described in Section 7.
2. Review of related works

In this section, some related studies of high-utility itemset mining (HUIM) and the maintenance algorithms in dynamic databases are briefly reviewed.

2.1. High-utility itemset mining

High-utility itemset mining (HUIM) considers both quantities and profits of itemsets as the interesting factors to derive valuable and meaningful information, which is an emerging issue in KDD. Chan et al. first presented top-\(k\) high utility closed patterns for producing both positive and negative utilities in 2003 [6]. Yao et al. then developed an algorithm to efficiently mine HUIs based on the mathematical properties of utility constraints. The proposed algorithm consists of two pruning strategies to reduce the search space based on the utility upper bounds and expected utility upper bounds [31,32]. Liu et al. developed a Two-Phase model [27] for discovering HUIs by the designed transaction-weighted downward closure (TWDC) property. Ahmed et al. then designed an incremental high-utility pattern (IHUP)-tree structure [5] to efficiently mine HUIs for transaction insertion. Lin et al. also developed a high-utility pattern (HUP)-tree algorithm [23] to compress the original database into a tree structure and use the similar FP-growth algorithm to derive HUIs without candidate generation. Tseng et al. designed two UP-growth and UP-growth\(^+\) algorithms [29] to mine HUIs based on the designed UP-tree structure. Song et al. also proposed the CHUI-Mine (Concurrent High Utility Itemsets Mine) algorithm for mining HUIs by dynamically pruning the designed CHUI-Tree structure [28]. It is, however, still a tree-based approach and the HTWUIs are necessary to be discovered.

The above approaches are processed to overestimate the upper-bound utilities of high transaction-weighted utilization itemsets (HTWUIs) based on Two-Phase mode. An additional database scan is thus necessary to find actual HUIs. To solve the limitation of Two-Phase model, Liu et al. presented the HUI-Miner algorithm [26] and a vertical utility-list structure to directly produce HUIs without candidate generation. Fournier-Viger et al. also extended HUI-Miner algorithm to design a novel strategy for efficiently mining HUIs based on the co-occurrences among 2-itemsets [12]. Other algorithms for mining variants of HUIs such as mining top-\(k\) HUIs [30], discovering HUIs with multiple minimum supports [15], mining on-shelf high utility itemsets [14,20], mining HUIs with negative unit profits [11], and mining weighted erasable patterns by considering item weights [13] are still developed in progress.

2.2. Maintenance algorithms in dynamic databases

In the past, Cheung et al. first designed the fast updated (FUP) algorithm [9] for handling transaction insertion to maintain and update the discovered frequent itemsets in a level-wise way. Hong et al. also developed fast updated frequent pattern (FUFP)-tree algorithm to efficiently update the built tree structure for mining the updated frequent itemsets [18]. Several incremental algorithm have been also extensively addressed to maintain the frequent itemsets or association rules [8,17,22]. For example, the PRE-HUI-INS algorithm was proposed to incrementally mine HUIs based on pre-large concept with transaction insertion [24].

In practical situation, transaction deletion is also an important issue in KDD [10]. Lin et al. respectively designed FUP-HUI-DEL [25] and PRE-HUI-DEL [21] maintenance algorithms for handling transaction deletion to maintain and update the discovered HUIs and HTWUIs based on the Two-Phase model. For the FUP-HUI-DEL algorithm, it divides the discovered HTWUIs into two parts with four cases and processes the maintenance procedures in a level-wise way. For the PRE-HUI-DEL algorithm, it divides the
discovered HTWUIs and PTWUIs into three parts with nine cases and processes the similar maintenance procedures as the FUP-HUI-DEL algorithm. Based on the developed PRE-HUI-DEL algorithm, the updated database is unnecessary to be rescanned until the number of accumulative total utilities achieves the designed safety bound. Both the FUP-HUI-DEL and PRE-HUI-DEL algorithms are required numerous computations to scan the original databases for discovering HTWUIs and an additional database scan is needed for determining the remaining HTWUIs to find the actual HUIs.

3. Preliminaries and problem statement

In this section, the preliminaries and problem statement of HUIM for transaction deletion are given below.

3.1. Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite set of $m$ distinct items in a quantitative database $D = \{T_1, T_2, \ldots, T_n\}$, where each transaction $T_q \in D$ is a subset of $I$ containing several items with its purchased quantities $q(i_j, T_q)$ and has an unique identifier, called TID. The corresponding profit table, $ptable = \{pr_1, pr_2, \ldots, pr_m\}$, in which $pr_j$ is the profit value of an item $i_j$. An itemset $X$ is a set of $k$ distinct items $\{i_1, i_2, \ldots, i_k\}$, where $k$ is the length of an itemset called $k$-itemset. An itemset $X$ is said to be contained in a transaction $T_q$ if $X \subseteq T_q$. An user-specified minimum utility threshold is set as $\varepsilon$. A quantitative transactional database is shown in Table 1. It consists of 12 transactions and 5 items, respectively donated from $A$ to $E$.

Assume the minimum utility threshold is set as 25% and the predefined corresponding profit values for the items are defined in a profit table, which is shown in Table 2.

**Definition 1.** The utility of an item $i_j$ in $T_q$ is defined as:

$$u(i_j, T_q) = q(i_j, T_q) \times pr(i_j).$$

For example, the utility of an item $\{C\}$ in transaction 1 (TID = 1) is calculated as: $u(C, T_1) = q(C, T_1) \times pr(C) (= 3 \times 12) (= 36)$.
**Definition 2.** The utility of an itemset \(X\) in transaction \(T_q\) is denoted as \(u(X, T_q)\), which can be defined as:

\[
u(X, T_q) = \sum_{i_j \in X \land X \subseteq T_q} u(i_j, T_q).
\]

For example, the utility of an itemset \(\{AC\}\) in \(T_1\) is calculated as: \(u(AC, T_1) = u(A, T_1) + u(C, T_1) = 2 \times 4 + 3 \times 12 = 44\).

**Definition 3.** The utility of an itemset \(X\) in \(D\) is denoted as \(u(X)\), which can be defined as:

\[
u(X) = \sum_{X \subseteq T_q \land T_q \in D} u(X, T_q).
\]

For example, \(u(AC) = u(AC, T_1) + u(AC, T_3) + u(AC, T_6) + u(AC, T_9) = 44 + 16 + 32 + 48 = 140\).

**Definition 4.** The transaction utility of transaction \(T_q\) is denoted as \(tu(T_q)\), which can be defined as:

\[
tu(T_q) = \sum_{j=1}^{m} u(i_j, T_q),
\]

in which \(m\) is the number of items in \(T_q\).

For example, \(tu(T_1) = u(A, T_1) + u(C, T_1) + u(E, T_1) = 8 + 36 + 30 = 74\).

**Definition 5.** The total utility of \(D\) is denoted as \(TU^D\), which can be defined as:

\[
TU^D = \sum_{T_q \in D} tu(T_q).
\]

For example, the transaction utilities for \(T_1\) to \(T_{12}\) are respectively calculated as \(tu(T_1) = 74\), \(tu(T_2) = 13\), \(tu(T_3) = 63\), \(tu(T_4) = 24\), \(tu(T_5) = 30\), \(tu(T_6) = 62\), \(tu(T_7) = 44\), \(tu(T_8) = 19\), \(tu(T_9) = 60\), \(tu(T_{10}) = 53\), \(tu(T_{11}) = 59\) and \(tu(T_{12}) = 35\). The total utility in \(D\) is the sum of all transaction utilities, which is calculated as: \((74 + 13 + 63 + 24 + 30 + 62 + 44 + 19 + 60 + 53 + 59 + 35) = 536\).

**Definition 6.** An itemset \(X\) in database \(D\) is denoted as a high-utility itemset (HUI) as:

\[
HUI \leftarrow \{X \mid u(X) \geq \varepsilon \times TU^D\},
\]

in which \(\varepsilon\) is the use-specified minimum utility threshold.

For example, an item \(\{A\}\) is not considered as a HUI in the running example database since its utility is calculated as \(u(A) = 42\), which is smaller than \(0.25 \times 536 = 134\). An itemset \(\{AC\}\) is considered as a HUI in database \(D\) since its utility is calculated as \(u(AC) = 140\), which is greater than the minimum utility count (> 134). For the given example, the set of discovered HUIs is shown in Table 3.

From Table 3, it can be seen that the downward closure (DC) property of ARM is not suitable in HUIM. For example, the item \(\{A\}\) is not a HUI but its superset \(\{AC\}\) is considered as a HUI. A large number of candidates can thus be generated without DC property in HUIM. The well-known transaction-weighted downward closure (TWDC) property of Two-Phase model [27] was thus proposed to reduce the number of candidates, thus speeding up the computations for mining HUIs.
Definition 7. The transaction-weighted utility of an itemset \( X \) in \( D \) is the sum of all transaction utilities \( tu(T_q) \) containing itemset \( X \), which is defined as:

\[
TWU^D(X) = \sum_{X \subseteq T_q \wedge T_q \in D} tu(T_q).
\]

Definition 8. An itemset \( X \) is defined as a high transaction-weighted utilization itemset (HTWUI) as:

\[
HTWUI \leftarrow \{X \mid TWU^D(X) \geq \varepsilon \times TU^D\}.
\]

For example, the transaction-weighted utility of an item \( \{E\} \) in Table 1 is calculated as \( TWU(E) = tu(T_1) + tu(T_3) + tu(T_5) + tu(T_7) + tu(T_8) + tu(T_{10}) = (74 + 63 + 30 + 44 + 19 + 53) = 283 \) and it is considered as a HTWUI since \( TWU(E) = 283 > 134 \).

Property 1. The transaction-weighted downward closure (TWDC) property of Two-Phase model is that if an itemset \( X \) is not as a HTWUI, all its supersets are not HUI [27].

Based on TWDC property of Two-Phase model [27], numerous candidates and combinational computations can be greatly reduced.

3.2. Problem statement of HUIM for transaction deletion

Based on the above definitions, the problem statement of HUIM for transaction deletion can be defined as follows:

**Problem statement.** Given a transaction database \( D \), the total utility of \( D \) is \( TU^D \), a user-specified minimum utility threshold is set at \( \varepsilon \), a set of \( d \) transactions are deleted from \( D \), and the total utility of \( d \) is \( TU^d \). The problem of HUIM for transaction deletion from the updated database \( (D - d) \) is to find the completely updated set of the \( k \)-itemsets whose utilities are no less than \( \varepsilon \times (TU^D - TU^d) \).

Thus, the purpose of HUIM for transaction deletion is to efficiently update the discovered HUIs. Based on the current observation of HUIM for transaction deletion, it can be divided into two categories as: (1) the algorithms are processed in batch mode when transactions are changed (deleted) from the original database, and (2) the maintenance strategy to efficiently maintain and update the discovered information of HUIs or HTWUIs.

4. Proposed maintenance algorithm for transaction deletion

Since the HUI-Miner algorithm [26] was not designed for mining HUIs in the dynamic environment, it can only perform in batch-mode process and cannot be used to maintain the discovered HUIs with transaction deletion. Based on the HUI-Miner, less memory was required to store the necessary information based on the compressed utility-list structure. Besides, the multiple database scans in a level-wise mechanism can also be greatly avoided. Hence, a utility-list structure is adopted in the developed maintenance algorithm for handling transaction deletion in the dynamic databases.

For the designed HUI-list-DEL algorithm, the utility-list structure is built in advance for later maintenance process. When some transactions are consequentially deleted from the original database, the proposed maintenance is then processed to maintain and update the utility-list structure by the designed procedures. The utility-list structure of the deleted transactions is also constructed. After that, the utility-list structure from the original database and from the deleted transactions are then merged and updated based on the designed procedures. The complete set of updated HUIs can thus be directly discovered from the developed enumeration tree based on the updated utility-list structure. Details of the proposed algorithm are described below.
4.1. Maintenance operation by utilizing utility-list structure

The utility-list structure inherits the property of HUI-Miner algorithm. Each itemset $X$ in the utility-list structure keeps a set of elements, including $TID$ number of $X$ in $T_q$ ($TID$), the utility of $X$ in $T_q$ ($iu$), and the remaining utility of $X$ in $T_q$ ($ru$).

**Definition 9.** If an itemset $X'$ is an extension of itemset $X$, it denoted as $(X' - X) = (X' / X)$, which indicates that any extension of $X$ is a combination of the itemsets; if there is an itemset $X$ and a transaction $T$ with $X \subseteq T$, then set of all the items after $X$ in $T$ is denoted as $(T - X) = (T / X)$.

For example, an itemset $\{AB\}$ is an extension of an item $\{A\}$, which can be denoted as $(AB - A) = (AB / A)$. In $T_3$, $(T_3 - B)$ can be calculated as: $(T_3 - B) = (T_3 / B) = \{CE\}$.

**Definition 10.** An element of $X$ in the utility-list consisted of three fields, including the set TIDs for $X$ in of $T_q$ ($X \subseteq T_q \in D$), the set of utility for $X$ in $T_q$ ($iu$), and the set of remaining utility for $X$ in $T_q$ ($ru$), in which $iu$ and $ru$ are defined as:

$$X.iu(T_q) = \sum_{i_j \in X} u(i_j, T_q),$$

$$X.ru(T_q) = \sum_{i_j \in (T_q / X)} u(i_j, T_q).$$

The construction procedure of utility-list structure is recursively processed for $k$-itemsets if it is necessary to determine the itemsets in the search space based on the designed procedure. Details of construction algorithm for utility-list structure can be found in HUI-Miner [26]. Based on the construction of utility-list structure, two observations can be obtained as follows.

**Observation 1.** In the construction algorithm of utility-list structure, all 1-itemsets (items) should be sorted in ascending order of their transaction-weighted utilization utilities (TWU values). Moreover, the designed algorithm necessary keep all 1-itemsets for constructing the initially utility-list structure since the small utility itemsets (not existing in HUIs) may become HUIs after the database is updated. Thus, in order to guarantee the completeness and correctness of the updated HUIs which derived by the designed maintenance algorithm, all 1-itemsets should be kept for constructing the initially utility-list structure.

**Observation 2.** The ascending order of TWU values in the updated database is close to the original TWU ascending order. The reason is that the size of the deleted transactions is always quite smaller to those of original database. Thus, the ascending order of TWU values after transaction deletion will not be seriously changed.

Since all 1-itemsets are completely kept to construct the initially utility-list structure, the designed maintenance algorithm can quickly find the updated HUIs without losing information. From the given example in Table 1, the TWU values of 1-itemsets are $\{A: 338, B: 316, C: 395, D: 303, E: 283\}$, and the construction process of utility-list structure for 1-itemsets are sorted in ascending order of their TWU values as $\{E < D < B < A < C\}$. The results of the utility-list structures of all 1-itemsets are shown in Fig. 1.

**Definition 11.** The $X.IU$ is to sum the utilities of an itemset $X$ in $D$, which is defined as:

$$X.IU = \sum_{X \subseteq T_q \land T_q \in D} X.iu(T_q).$$
**Definition 12.** The \( X.RU \) is to sum the remaining utilities of an itemset \( X \) in \( D \), which is defined as:

\[
X.RU = \sum_{X \subseteq T_q, T_q \in D} X.ru(T_q)
\]

For example, the itemset \{A\} exists in TIDs \{1, 3, 6, 7, 9, 12\}. The \( A.IU \) is calculated as \( A.IU \) \((= 8 + 4 + 8 + 12 + 16) (= 52)\), and \( A.RU \) is calculated as \( A.RU \) \((= 36 + 12 + 24 + 0 + 36 + 0) (= 108)\). In addition, an itemset \{AE\} exists in TIDs \{1, 3, 7\}. \( AE.IU \) is calculated as \( AE.IU \) \((= A.IU + E.IU) (= 106)\), and \( AE.RU \) \((= A.RU + E.RU) (= 139)\).

### 4.2. Two pruning strategies

The search space of HUI-list-DEL algorithm is concerned in an enumeration tree, which is the same as the HUI-Miner. For the designed HUI-list-DEL algorithm, two early pruning strategies \([12, 26]\) can be developed, thus reducing the size of search space for mining HUIs. The related properties of the two strategies are described below.

**Property 2.** In the enumeration tree, given the constructed utility-list structure of an itemset (node) \( X \), it can be obtained that the summation of \( X.IU \) and \( X.RU \) is always no less than any extension \( X' \) of \( X \) (i.e. any superset) \([26]\).

**Proof.** For \( \forall \) transaction \( T_q \supseteq X' \):

\[
\therefore X' \text{ is an extension of } X \Rightarrow (X' - X) = (X'/X).
\]

\[
X \subset X' \subseteq T_q \Rightarrow (X'/X) \subseteq (T_q/X).
\]

\[
\therefore X'.iu(T_q) = X.iu(T_q) + (X'/X).iu(T_q);
\]

\[
= X.iu(T_q) + \sum_{i \in (X'/X)} i.iu(T_q);
\]

\[
\leq X.iu(T_q) + \sum_{i \in (T_q/X)} i.iu(T_q);
\]

\[
= X.iu(T_q) + X.ru(T_q).
\]
\[ X'.iu(T_q) \leq X.iu(T_q) + X.ru(T_q). \]

\[ X \subset X' \Rightarrow X'.TIDs \subseteq X.TIDs. \]

\[ X'.IU = \sum_{T_q \in X'.TIDs} X'.iu(T_q); \]

\[ \leq \sum_{T_q \in X'.TIDs} (X.iu(T_q) + X.ru(T_q)); \]

\[ \leq \sum_{T_q \in X.TIDs} (X.iu(T_q) + X.ru(T_q)); \]

\[ = \sum_{T_q \in X.TIDs} (X.iu(T_q)) + \sum_{T_q \in X.TIDs} (X.ru(T_q)); \]

\[ = X.IU + X.RU. \]

\[ X'.IU \leq X.IU + X.RU. \]

**Pruning strategy 1** (Early termination). When performing the depth-first strategy in the enumeration tree, if the summation of \( X.IU \) and \( X.RU \) is smaller than the user-specified minimum utility count, any extension \( X' \) of \( X \) is not a HUI; all of them can be directly pruned.

In addition, we further adopt the estimated utility co-occurrence pruning (EUCP) strategy [12] to efficiently reduce the number of join operations for utility-list structures. The EUCP concept is used to build estimated utility co-occurrence structure (EUCS) for directly eliminating the extensions with lower utilities and all its transitive extensions is unnecessary to re-construct their utility-list structures.

**Property 3.** If a 2-itemset \( X \) is not a HTWUI, any \( k \)-itemset \( (k \geq 3) \) extension of \( X \) will not be a HTWUI or HUI.

**Proof** Note that a 2-itemset is defined as \( X \), assume a \( k \)-itemsets \( X^k \ (k \geq 3) \). From property 1, it can be get that \( TWU(X^k) \leq TWU(X^{k-1}) \). Thus, if a 2-itemset \( X \) is not a HTWUI, any extension of \( X \) as \( k \)-itemset \( X^k \ (k \geq 3) \) will not be a HTWUI or HUI.

**Pruning strategy 2** (EUCP). When a 2-itemset \( X \) is not a HTWUI, it indicates that any \( k \)-itemset \( (k \geq 3) \) extension of \( X \) will not be a HTWUI or HUI; all of them can be directly pruned, and stop performing the depth-first search.

Based on the EUCP pruning strategy, a huge number of unpromising \( k \)-itemset \( (k \geq 3) \) can be significantly pruned in the proposed maintenance algorithm. The above two pruning strategies can successfully apply to the developed HUI-list-DEL for handling variants of data characteristics since the pruning strategies are designed based on the measure criterion of utility factor, which is independent to data characteristics. The EUCS structure of the given example is shown in Table 4, which is used to keep the TWU values between two itemsets.

For example, in Table 4, it can be calculated that \( TWU(AB) = tu(T_3) + tu(T_7) + tu(T_{12}) = 63 + 44 + 35 \) (= 142). Thus, the \( TWU(AB) \) in Table 4 is set as 142.

**4.3. Proposed HUI-list-DEL algorithm**

Based on the above definitions and properties, the pseudo code of the proposed maintenance HUI-list-DEL algorithm for transaction deletion is described in Algorithm 1, and the HUI-DEL-Mine procedure is in described in Algorithm 2.
Table 4
The EUCS structure of the given database

<table>
<thead>
<tr>
<th>Item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>142</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C</td>
<td>259</td>
<td>175</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>D</td>
<td>201</td>
<td>181</td>
<td>119</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>E</td>
<td>181</td>
<td>209</td>
<td>190</td>
<td>74</td>
<td>–</td>
</tr>
</tbody>
</table>

Algorithm 1: HUI-list-DEL Algorithm

INPUT: 
- $D$, the original database; 
- $d$, the deleted database; 
- $TU^D$, the total utility of $D$; 
- $TU^d$, the total utility of $d$; 
- $ptable$, the profit table; 
- $\varepsilon$, the use-specified minimum utility threshold; 
- $EUCS$, the estimated utility co-occurrence structure; 
- $D.UL$, the utility-list of $D$.

OUTPUT: 
- $U$, the updated database ($U = D - d$); 
- $TU^U$, the total utility of $U$ ($TU^U = TU^D - TU^d$); 
- $U.UL$, the utility-list of $U$; 
- $EUCS$, the updated estimated utility co-occurrence structure; 
- $HUIs$, the complete set of updated high-utility itemsets in $U$.

/* scan $d$ to construct the utility-list structure for updating $EUCS$ */
1. for each 1-itemset $X \in T_d \land T_q \subseteq d$ do
2.   $X.UL \leftarrow X.UL \cup \{T_q, iu, ru\}$;  // $X.UL$, the utility-list of 1-itemset $X$
3.   update TWU ($X$) in $EUCS$;
4. end for
5. $d.UL \leftarrow \cup X.UL$  /* updated the utility-list structure */
6. for each $X \in D \land X.UL \in D.UL$ do
7.   if $X.UL \neq \emptyset$ then
8.     search itemset $X \in D.UL$ in $d.UL$;
9.     if $\exists (X \in D.UL \land X \in d.UL)$ then
10.    for each element $E_i \in X.UL \land X.UL \in b.UL$ do
11.       $X.iu \leftarrow X.iu - E_i.\text{iu}$;
12.       $X.ru \leftarrow X.ru - E_i.\text{ru}$;
13.       $X.UL \leftarrow X.UL - \{E_i\}$;  // completely or partially deleted
14.   end for
15. end if
16. $U.UL \leftarrow U.UL \cup \{X.UL\}$.
17. end if
18. end for
/* recursively mining HUIs by the depth-first search */
19. call HUI-DEL-Mine($\emptyset$, $U.UL$, $EUCS$, $\varepsilon$).
20. return $U$, $TU^U$, $U.UL$, $EUCS$, $HUIs$

For the proposed HUI-list-DEL algorithm, it first scans the deleted transactions to calculate the relevant information, and construct the utility-list structure of each item in the deleted transactions, and the TWU values between 2-itemsets in the built EUCS structure is then updated (Lines 1 to 5 in Algorithm 1). The built utility-list structure is also updated based on the original and deleted utility-list structures (Lines 6 to 18 in Algorithm 1). After that, the set of all the 1-extensions of itemset $X$ is recursively
Algorithm 2: HUI-DEL-Mine \((X, \text{extULsOf}X, \text{EUCS}, \varepsilon)\)

**INPUT:** \(X,\) an itemset; \(\text{extULsOf}X,\) the set of utility-list of all 1-extensions of \(X;\) \(\text{EUCS},\) the updated estimated utility co-occurrence structure; \(\varepsilon,\) the use-specified minimum utility threshold.

**OUTPUT:** \(\text{HUIs},\) the updated high-utility itemsets.

1. for each \(X_a \in \text{extULsOf}X\) do
2. if \(X_a.IU \geq (TU^D - TU^d) \times \varepsilon\) then
3. \(\text{HUIs} \leftarrow \text{HUIs} \cup X_a.\)
4. end if
5. if \((X_a.IU + X_a.RU) \geq (TU^D - TU^d) \times \varepsilon\) then
6. \(\text{extULsOf}X_a \leftarrow \emptyset.\)
7. for each \(X_b\) after \(X_a\) in \(\text{extULsOf}X\) do
8. if \(\exists \text{TWU}(X_a, X_b) \in \text{EUCS} \land \text{TWU}(X_a, X_b) \geq (TU^D - TU^d) \times \varepsilon\) then
9. \(\text{extULsOf}X_a \leftarrow \text{extULsOf}X_a \cup \text{Construct}(X, X_a, X_b).\)
10. end if
11. end for
12. end if
13. call HUI-DEL-Mine\((X_a, \text{extULsOf}X_a, \text{EUCS}, \varepsilon)\).
14. end for
15. return \(\text{HUIs}\)

processed (Line 19 in Algorithm 1; Lines 1 to 15 in Algorithm 2) based on depth-first search procedure HUI-DEL-Mine. Each itemset \(X\) is first determined to check whether it is a HUIs (Lines 2 to 4 in Algorithm 2). If \(X\) satisfies the condition of HUI, it is then produced and putted into the set of HUI; otherwise, the pruning strategy 1 is then performed to check whether the supersets of \(X\) are required to be determined (Line 5 in Algorithm 2). In addition, the pruning strategy 2 of the updated EUCS structure can also be used to prune the unpromising itemsets, thus reducing the search space for mining HUIs (Lines 7 to 11 in Algorithm 2). If \(X\) satisfies the condition, the \(\text{extULsOf}X\) was construct by the Construct\((X, X_a, X_b)\) function (details of this construction function can refer to HUI-Miner [26]), and the depth-first search is then recursively performed to HUI-DEL-Mine\((X_a, \text{extULsOf}X_a, \text{EUCS}, \varepsilon)\) function (Lines 5 to 13 in Algorithm 2); otherwise, the next itemset after \(X\) in the enumeration tree will be processed and determined. The updated \(U, TU^U, U.UL, \text{EUCS},\) and \(\text{HUIs}\) will be returned and kept for next iteration (Line 20 in Algorithm 1) when transactions are deleted from the original database. Based on the developed algorithm, the updated HUIs can be directly produced without candidate generation or multiple database scans in a level-wise way. Hence, the proposed HUI-list-DEL algorithm is more practical in real-world applications.

5. An illustrated example

Assume that the original database and the profit table were respectively shown in Tables 1 and 2. Also assume that the last four transactions shown in Table 5 are deleted from the original database.

The constructed utility-list structure of the original database was already shown in Fig. 1. Assume the minimum utility threshold is also set as 25%. The proposed HUI-list-DEL algorithm is then processed as follows. Firstly, the utility-list structure for the deleted transactions is then constructed for recording.
Table 5: Four deleted transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6: Transaction utilities of the deleted transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>tu</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>53</td>
</tr>
<tr>
<td>11</td>
<td>59</td>
</tr>
<tr>
<td>12</td>
<td>35</td>
</tr>
</tbody>
</table>

Fig. 2. Constructed utility-list structure for the deleted transactions.

The related information of utility values of the items, TWU values of the items. The items are sorted in ascending order of their TWU values to construct the utility-list structures. Take the first transaction \(T_9\) in Table 5 as an example to illustrate the procedure. For the first transaction in Table 5, the sorted items will be \(\{D\} < \{A\} < \{C\}\); In this example, for item \(D\), \(iu(D) = (2 \times 6) = (12)\) and \(ru(D) = (3 \times 12 + 3 \times 4) = (48)\); for item \(A\), \(iu(A) = (3 \times 4) = (12)\) and \(ru(A) = (3 \times 12) = (36)\); for item \(C\), \(iu(C) = (3 \times 12) = (36)\) and \(ru(C) = (0)\). Thus, \(UL(D) = \{TID, iu(D), ru(D)\} = \{9, 12, 48\}, UL(A) = \{TID, iu(A), ru(A)\} = \{9, 12, 36\}, and UL(C) = \{TID, iu(C), ru(C)\} = \{9, 36, 0\}\). The transaction utility for the first transaction is also calculated as \(tu(T_9) = iu(A) + iu(C) + iu(D) = (2 + 36 + 12) = (60)\). The other three transactions in Table 5 is also processed in the same way. After that, the utility-list structure of each item in the four deleted transactions is summed up together. For example, item \(\{D\}\) appears in transactions 9, 11 and 12 in Table 5, its utility-list set is \(UL(D) = \{TID, iu(D), ru(D)\} = \{\{9, 12, 48\}; \{11, 30, 29\}; \{12, 18, 17\}\}. The final results of the constructed utility-list structure and the transaction utilities of four deleted transactions are respectively shown in Fig. 2 and Table 6. The total utility of the four deleted transactions is calculated as \((tu_9 + tu_{10} + tu_{11} + tu_{12}) = (60 + 53 + 59 + 35) = (207)\). The updated total utility for the updated database is calculated as \(TU^U = (TU^D - TU^U) = (536 - 207) = (329)\), and the minimum utility count is calculated as \((TU^D - TU^U) \times \varepsilon = (329 \times 0.25) = (82.25)\).

The utility-list structure for the original database and the deleted transactions are then merged and updated. For example, the utility-list set of an item \(A\) in the original database is \(UL(A) = \{TID, iu(A), ru(A)\} = \{\{1, 8, 36\}; \{3, 4, 12\}; \{6, 8, 24\}; \{7, 4, 0\}; \{9, 12, 36\}; \{12, 16, 0\}\}; the utility-list set of item \(A\) in the deleted transactions is \(UL(A)' = \{TID, iu(A), ru(A)\} = \{9, 12, 36\}; \{12, 16, 0\}\}. The utility-list set of item \(A\) is updated as \(UL(A) = UL(A) - UL(A)' = \{TID, iu(A), ru(A)\} = \{\{1, 8, 36\}; \{3, 4, 12\}; \{6, 8, 24\}; \{7, 4, 0\}\}. The same process is also performed for the other items. After that, the updated utility-list structure of each item in this example is then shown in Fig. 3.

After the utility-list structure is maintained for transaction deletion, each node in the enumeration tree is then top-down determined for producing HUIs. In this example, the ascending order of the items is \(\{E < D < B < A < C\}\). The item \(\{E\}\) is first processed. The total \(iu\) and \(ru\) of an item \(\{E\}\) is calculated as \(E.RU = (30 + 45 + 15 + 15 + 15) = (120)\), and \(E.RU = (48 + 18 + 15 + 29 + 4) = (110)\). Since \(E.IU\) is larger than the minimum utility count \((110 > 82.25)\), an item \(\{E\}\) is directly
produced as a HUI. Since $E.IU + E.RU = 110 + 120 = 230 > 82.25$, the depth-first search is then performed to construct the supersets of \{E\} by \textit{HUI-DEL-Mine} function. In this example, the utility-list structures for the supersets of an item \{E\} are shown in Fig. 4.

The similar procedure is then recursively performed for all itemsets. The final results are shown in Table 7.

### 6. Experimental evaluation

In this section, several experiments were conducted to evaluate the performance of the proposed maintenance HUI-list-DEL algorithm compared to those of the other algorithms whether the well-known batch-mode algorithms such as HUI-Miner [26] and FHM [12], or the maintenance approaches such as FUP-HUI-DEL [25] and PRE-HUI-DEL [21] algorithms for transaction deletion. Since HUI-Miner and FHM are the state-of-the-art algorithms for HUI-M in static databases, the other related algorithms, such as UP-growth$^+$ [29], CHUI-Mine [28], and MHU-Growth [15], as well as the PRE-HUI-INS algorithm for transaction insertion [24] are unnecessary compared in the experiments.

The proposed HUI-list-DEL algorithm is then divided into two different approaches with two different pruning strategies in the experiments. Note that the HUI-list-DEL1 algorithm only adopts the early pruning strategy 1, and the HUI-list-DEL2 algorithm adopts both of the designed two pruning strategies. Since the HUI-Miner and FHM algorithms were not designed for mining HUIs with transaction deletion in dynamic databases, both of them are performed in batch mode which indicates that the multiple database scans are required to re-mine the HUIs when transactions are deleted from the original databases.

#### 6.1. Experimental setup and dataset descriptions

The experiments are implemented in the Java language and executed on a PC with an Intel Core i5-3470 at 3.2 GHz CPU and 4 GB of memory, running on the Windows 7 platform. Both real-life [1]
and synthetic datasets [4] were used in the experiments to evaluate the effectiveness of the proposed maintenance HUI-list-DEL algorithm. A simulation model [27] was developed to generate the quantities of items in the transactions for the database except foodmart dataset. The range of quantities was set from 1 to 5, and the profit was randomly set from 1 to 1000 in the utility table. The same as previous studies, there was no noise model in the real-life and synthetic datasets. The parameters of the datasets are defined as follows: \(|D|\) indicates the total number of transactions; \(\text{AvgLen}\) is the average transaction length; the number of distinct items is set as \(|I|\); and \(\text{Type}\) is defined as the dataset type (sparse or dense). The characteristics of the used datasets are shown in Table 8.

In the experiments, two parameters such as minimum utility threshold (MU) and deletion ratio (DR) are respectively set for different datasets to evaluate the performance of the proposed algorithm. The transactions for deletion are selected bottom-up from the original dataset and the number of transactions for deletion is set as \((|D| \times DR)\). For example, assume a dataset size is set as 100, DR is set as 5%, the bottom-up \((100 \times 0.05) = 5\) transactions are selected as the transactions for deletion, and the updated database contains \((100 - 5) = 95\) transactions after transaction deletion.

6.2. Performance under various MUs

The execution time, number of patterns and memory consumption are compared under various minimum utility thresholds. The execution time includes both computational time and I/O time. For the conducted experiments under different minimum utility thresholds, the DR is set as a fixed number to evaluate the performance of the compared algorithms in four different datasets. The results are shown in Fig. 5.

From Fig. 5, it can be observed that both the HUI-list-DEL1 and HUI-list-DEL2 algorithms outperform the other algorithms in four datasets under various MUs, and HUI-list-DEL2 has the best performance among them. For example in Fig. 5(d), when DR is fixed set as 5% and MUs are respectively set from 0.05% to 0.25%, with 0.05% increment each time, the runtime of HUI-list-DEL1 algorithm is slightly faster than HUI-Miner algorithm, but worse than FHM and HUI-list-DEL2 algorithms. The HUI-list-DEL2 algorithm has almost one or two orders of magnitude faster than the other algorithms. The reason is that when MU is set lower, the FUP-HUI-DEL and PRE-HUI-DEL algorithms are required to maintain and update a huge number of HTWUIs. This procedure takes numerous computations to find the actual HUIs, which is not suitable in real-world applications. Since the HUI-list-DEL1 and HUI-list-DEL2 algorithms applied the developed pruning strategies to early prune the unpromising itemsets, the search space and the computations can thus be greatly reduced.

Besides, it also can be found that FHM algorithm has poor performance than HUI-Miner and HUI-list-DEL2 algorithms in foodmart and mushroom datasets but has slightly better results than the proposed HUI-list-DEL1 algorithm. The reason is that the foodmart is a sparse dataset with 4.4 average transaction length, many unpromising candidates can be directly pruned by the TWU model. The constructed EUCS structure is thus inefficient for early pruning the unpromising itemsets but the numerous computations are required for constructing and maintaining EUCS structure. For the densely mushroom dataset with 23 average transaction length, most produced candidate itemsets have highly TWU values. Two pruning strategies are inefficient to prune the candidate itemsets. Thus, the FHM algorithm has no good performance either in sparsely or densely datasets.

The number of high transaction-weighted utilization itemsets (HTWUIs), the pre-large transaction-weighted utilization itemsets (PTWUIs) and the high-utility itemsets (HUIs) are then compared under various MUs and a fixed DR. Note that FUP-HUI-DEL algorithm generates the HTWUIs and HUIs,
and PRE-HUI-DEL generates HTWUIs, PTWUIs and HUIs. The generated HTWUIs of the PRE-HUI-DEL algorithm is the same as those of the FUP-HUI-DEL algorithm. For the HUI-Miner, FHM and the proposed algorithms, only HUIs are directly produced without candidate generation. The results are then shown in Fig. 6.

From Fig. 6, it can be observed that the huge numbers of HTWUIs and PTWUIs are generated but rare of them belong to the required HUIs. The FUP-HUI-DEL and PRE-HUI-DEL algorithms are performed in a level-wise approach to necessary generate the huge number of HTWUIs for deriving the actual HUIs. Besides, the pre-large concept is adopted in the PRE-HUI-DEL algorithm, thus keeping more PTWUIs to reduce the computations of multiple database scans. Although the TWDC property is adopted in the proposed algorithm to prune the unpromising itemsets, it still, however, requires numerous computations to generate amount of candidates in a level-wise way. The same results are also obtained in other datasets. The experiments of memory consumption are then conducted to show the performance under various MUs with a fixed DR. The results are shown in Fig. 7.

From Fig. 7, it can be observed that the proposed HUI-list-DEL1 and HUI-list-DEL2 algorithms required slightly more memory than the other algorithms in Figs 7(a), (b) and (d) except in Fig. 7(c). It
is reasonable since when MU is increased, fewer candidate itemsets were processed. Besides, the proposed two HUI-list-DEL1 and HUI-list-DEL2 algorithms have better performance compared to previous two maintenance algorithms, FUP-HUI-DEL and PRE-HUI-DEL, for transaction deletion and require slightly more memory than the HUI-Miner and FHM algorithms. Among them, the HUI-list-DEL2 algorithm requires slightly more memory than HUI-list-DEL1 algorithm since the EUCP strategy is necessary to build the EUCS for 2-itemsets. When MU is set lower, the FUP-HUI-DEL and PRE-HUI-DLE algorithms require more memory than our proposed algorithm with two pruning strategies. It also can be observed that the gap between two maintenance algorithms (FUP-HUI-DEL and PRE-HUI-DEL) and the proposed HUI-list-DEL1 and HUI-list-DEL2 algorithms is larger along with the decreasing of MUs.

6.3. Performance under various DRs

In this section, the performance of the proposed algorithm with two pruning strategies compared to the
other approaches is evaluated under various DRs with a fixed MU. The results of runtime under various DRs in four datasets are shown in Fig. 8.

From Fig. 8, it can be observed that the proposed HUI-list-DEL2 algorithm has lesser runtime compared to those of the other algorithms in four datasets. In Figs 8(a) and (b), the proposed algorithm with two pruning strategies almost requires the same runtime. As the previous discussion of FHM algorithm, it is inefficient to handle both sparsely or densely datasets. Thus, two proposed approaches require more computations in foodmart and mushroom datasets but still faster than the other state-of-the-art batch-mode HUI-Miner and FHM algorithms. In addition, most algorithms have stable results under various DRs with a fixed MU except the FUP-HUI-DEL and PRE-HUI-DEL algorithms in four datasets. It can be concluded that different DRs would not seriously influence the runtime for transaction deletion compared to the other algorithms in four datasets. The results for number of patterns are then shown in Fig. 9.

From Fig. 9, it can be observed that huge numbers of HTWUIs and PTWUIs are generated, but rare of them belong to HUIs. The proposed HUI-list-DEL algorithm reduces considerable redundancy and up to 90% compared to the previous two maintenance FUP-HUI-DEL and PRE-HUI-DEL algorithms. The reason is that the TWU model keeps HTWUIs based on the highly upper-bound utility (TWU) value.
of Two-Phase model. Although PRE-HUI-DEL adopts pre-large concept to reduce multiple database scans by keeping the slight number of PTWUIs, it still has to maintain a huge number of HTWUIs. For the proposed HUI-list-DEL1 and HUI-list-DEL2 algorithms, the HUIs can be directly produced by the updated utility-list structure without candidate generation. From the observed results shown in Fig. 9, it can be concluded that various DRs would not seriously influence the number of patterns. Since rare HUIs are produced from the evaluated datasets, the batch-mode HUI-Miner and FHM algorithms are inefficient to re-scan the original datasets and re-mine HUIs. The results of memory consumption under various DRs with a fixed MU in four different datasets are shown in Fig. 10.

From Fig. 10, it also can be observed that the proposed HUI-list-DEL1 and HUI-list-DEL2 algorithms require slightly more memory than the other compared algorithms except in retail dataset. In addition, the proposed HUI-list-DEL2 algorithm requires slightly more memory than proposed HUI-list-DEL1 algorithm since the EUCP pruning strategy is adopted in the HUI-list-DEL2 algorithm. From the results in Fig. 10, the memory consumption remains stable for all compared algorithms under various DRs from 1% to 5%, increments 1% each time. The designed algorithm with two pruning strategies in real-world applications can thus be acceptable.
6.4. Scalability

In this section, the scalability is then evaluated to show the performance of the proposed algorithm compared to those of the other approaches. The experiments are performed on a series of synthetic datasets T10I4 N4KD|X|K under various dataset size. The MU and DR are respectively set at 0.1% and 5%. The results in terms of runtime, memory consumption and number of patterns are respectively shown in Fig. 11.

From Fig. 11, it can be observed that all compared algorithms have good scalability in term of runtime. As shown in Fig. 11(a), the proposed HUI-list-DEL2 algorithm has better runtime under various dataset size from 100 K to 500 K, increments 100 K each time. Both the FUP-HUI-DEL and PRE-HUI-DEL algorithms have more execution time than the proposed HUI-list-DEL2 algorithm. The state-of-the-art FHM algorithm outperforms the HUI-Miner and HUI-list-DEL1 algorithms in the synthetic datasets T10I4 N4KD|X|K, but has worse result than the proposed HUI-list-DEL2 algorithm. The reason is that both FHM and HUI-list-DEL2 algorithms applied EUCP strategy to reduce the unpromising $k$-itemsets by the developed EUCS structure. Furthermore, the proposed HUI-list-DEL2 algorithm has
better performance than the FHM algorithm in term of execution time. When the dataset size is set higher, the gap between the proposed HUI-list-DEL2 algorithm and the others will be larger.

From Fig. 11(b), it can be observed that the FHM and HUI-list-DEL2 algorithms have more memory consumption than the other algorithms and the HUI-list-DEL2 algorithm requires slightly more memory than FHM algorithm along with the increasing of dataset size. The similar results can be obtained between HUI-list-DEL1 and HUI-Miner algorithms. The reason is that both the HUI-list-DEL1 and HUI-list-DEL2 algorithms maintain the built utility-list structure to discover HUIs when transactions are consequentially deleted from the original dataset, slightly more memory is thus required compared to those of the HUI-Miner and FHM algorithms. From Fig. 11(c), it also can be observed that the proposed algorithm can greatly avoid the numerous HTWUIs and PTWUIs for producing the actual HUIs without candidate generation.

6.5. Summary of experimental results

From the conducted experiments, only one time iteration is used to evaluate the performance of the proposed maintenance algorithm. When several iterations of transaction deletion are performed, the
proposed algorithm generally has better results along with the increasing of the number of deleted transactions. In real-world situations, the dataset is usually referred to dynamic environment. When some transactions are consequentially deleted from the original dataset, the batch-mode algorithms necessary perform the entirely mining process to re-scan the updated dataset and re-mine the desired information. The proposed algorithm can thus avoid the multiple database scans and maintain the already built utility-list structure, which is more suitable in real-world applications.

In addition, the proposed method and the designed pruning strategies can be modified and applied to improve the previous works of HUIM, such as the tree-based algorithms of UP-growth\(^+\) [29], CHUI-Mine [28], MHU-Growth [15], and WEP (Weighted Erasable Patterns with a tree structure) [13]. The other maintenance algorithms, such as PRE-HUI-INS [24], can also be improved to filter out the nu-
merous unpromising HTWUIs based on the designed pruning strategies. The reason is that the pruning strategies are designed based on the measure criterion of utility factor. Thus, it is easier to speed up the performance of the previous algorithms used in HUIM, especially in the dynamic environment.

7. Conclusions

In real-world applications, database is frequently changed whether transaction insertion or transaction deletion. The discovered information may become invalid or new information may arise. Traditional batch-mode algorithms are necessary to re-scan the updated database and re-mine the desired information each time when data is changed whether the number of changed transactions is small or large. In the past, two maintenance FUP-HUI-DEL and PRE-HUI-DEL algorithms for HUIM have been proposed to handle the problem of transaction deletion. However, both of them may easily incur the “combination explosion” problem especially when the threshold is set low. Besides, multiple database scans are also required for generating HTWUIs and PTWUIs in a level-wise way.

In this paper, a novel maintenance HUI-list-DEL algorithm for transaction deletion is thus proposed to maintain and update the built utility-list structure for directly producing HUIs without candidate generation. Two pruning strategies are also developed to early prune the unpromising itemsets, thus reducing the search space to speed up the computations for mining HUIs. Experimental results showed that the proposed algorithm with the developed strategies generally outperforms whether batch-mode algorithms or the maintenance ones in terms of execution time, number of discovered patterns, the memory consumption and scalability. The proposed algorithm can thus avoid the multiple database scans and re-use the already built utility-list structure, which is more suitable in real-world applications.

Acknowledgments

This research was partially supported by the Tencent Project under grant CCF-TencentRAGR20140114, by the Shenzhen Peacock Project, China, under grant KQC201109020055A, by the Natural Scientific Research Innovation Foundation in Harbin Institute of Technology under grant HIT.NSRIF.2014100, and by the Shenzhen Strategic Emerging Industries Program under grant ZDSY20120613125016389.

References

[16] J. Han, J. Pei, Y. Yin and R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, Data Mining and Knowledge Discovery 8(1) (2004), 53–87.