Advanced Engineering Informatics xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Advanced Engineering Informatics

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

Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases ☆

Chun-Wei Lin [a,b,*], Tzung-Pei Hong [c,d], Guo-Cheng Lan [e], Jia-Wei Wong [d], Wen-Yang Lin [c]

[a] Innovative Information Industry Research Center (IIIRC), School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen Graduate School, HIT Campus Shenzhen University Town Xili, Shenzhen, PR China
[b] Shenzhen Key Laboratory of Internet Information Collaboration, School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen Graduate School, HIT Campus Shenzhen University Town Xili, Shenzhen, PR China
[c] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan, ROC
[d] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC
[e] Department of Mathematics and Computer Sciences, Fuqing Branch of Fujian Normal University, Fuzhou, Fujian, PR China

ARTICLE INFO

Article history:
Received 14 March 2014
Received in revised form 21 August 2014
Accepted 26 August 2014
Available online xxxx

Keywords:
Utility mining
Pre-large concept
Transaction deletion
Two-phase approach
Dynamic databases

ABSTRACT

Most algorithms related to association rule mining are designed to discover frequent itemsets from a binary database. Other factors such as profit, cost, or quantity are not concerned in binary databases. Utility mining was thus proposed to measure the utility values of purchased items for finding high-utility itemsets from a static database. In real-world applications, transactions are changed whether insertion or deletion in a dynamic database. An existing maintenance approach for handling high-utility itemsets in dynamic databases with transaction deletion must rescan the database when necessary. In this paper, an efficient algorithm, called PRE-HUI-DEL, for updating high-utility itemsets based on the pre-large concept for transaction deletion is proposed. The pre-large concept is used to partition transaction-weighted utilization itemsets into three sets with nine cases according to whether they have large (high), pre-large, or small transaction-weighted utilization in the original database and in the deleted transactions. Specific procedures are then applied to each case for maintaining and updating the discovered high-utility itemsets. Experimental results show that the proposed PRE-HUI-DEL algorithm outperforms a batch two-phase algorithm and a FUP2-based algorithm in maintaining high-utility itemsets.

1. Introduction

In various mining techniques [1–8], association rule mining is the most popular method used for knowledge discovery to find the relationships among items or products. The most common implementations use the Apriori algorithm [9] for generating and testing the candidate itemsets level by level. All frequent itemsets are first found based on a user-defined minimum support threshold and then the association rules are derived from the discovered frequent itemsets based on the user-defined minimum confidence threshold. In association rule mining, each item is treated as a binary variable to discover relationships among itemsets or products. The frequency of an itemset is, however, insufficient for identifying highly profitable itemsets with small sold quantities.

Utility mining [7,10,11] was thus proposed to solve the limitations of frequent itemsets. It may be thought of as frequent itemset mining with sold quantities and item profits concerned. In practice, the utility value of an itemset can be cost, profit, or some other term defined by the user. For example, itemsets with good profits or those with low pollution during manufacturing may be of interest. Liu et al. designed a two-phase algorithm [12] for efficiently extracting high-utility itemsets based on transaction-weighted utilization (TWU) to keep the downward closure property. The TWU is used as an effective upper bound to reduce the generation of candidates for later processing. An additional database scan is performed to determine the real utility values of the remaining candidates to identify high-utility itemsets. Most approaches [7,10,13,11] for handling high-utility itemsets are, however, processed in batch mode with a static database.

In real-world applications, databases tend to be large and dynamic since their contents are frequently changed whether the number of transactions are inserted or deleted. Previous

☆ Handled by C.-H. Chen.

* Corresponding author at: Innovative Information Industry Research Center (IIIRC), School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen Graduate School, HIT Campus Shenzhen University Town Xili, Shenzhen, PR China.

E-mail addresses: jerrylin@ieee.org (C.-W. Lin), tphong@nuk.edu.tw (T.-P. Hong), rrfoheiay@gmail.com (G.-C. Lan), jwwong.alex@gmail.com (J.-W. Wong), wylin@nuk.edu.tw (W.-Y. Lin).

http://dx.doi.org/10.1016/j.aei.2014.08.003
1474-0346/

discovered information may become invalid, or some new information may emerge in the updated database. Dynamic data mining can be referred as an updating technique to find the up-to-date information without re-scanning the original database and re-mining the desired information each time. Hong and Lin et al. proposed maintenance approaches for respectively maintaining the discovered frequent itemsets in dynamic databases based on Fast UPdated (FUP) [14,15] and pre-large concepts [16,17]. Lin et al. then extended the FUP concept to handle transaction insertion [18] for maintaining discovered high-utility itemsets.

Transactions are also frequently deleted in real-world applications. The FUP2 concept [19,20] for transaction deletion has been adopted for updating discovered high-utility itemsets [21]. The database, however, must be rescanned if a high-utility itemset was not large or pre-large both in the original database and in the deleted transactions. In this paper, a maintenance algorithm, called PRE-HUI-DEL, for transaction deletion based on the pre-large concept [22] and the TWU model [12] is proposed to maintain and update discovered high-utility itemsets in dynamic databases. The proposed PRE-HUI-DEL algorithm first partitions transaction-weighted utilization itemsets into three sets with nine cases according to whether they have large (high), pre-large, or small transaction-weighted utilization in the original database and in the deleted transactions. Specific procedures are then applied to each case to maintain and update the discovered transaction-weighted utilization itemsets. The major contributions of this paper are as follows:

1. Traditional utility mining processes a database in batch mode no matter whether transactions are deleted. For the original database, information was already discovered by data mining approaches. It is thus not efficient to waste the discovered information for updating the whole database with a small number of deleted transactions. In this paper, an efficient approach is proposed to handle transaction deletion for maintaining the discovered high-transaction-weighted utilization itemsets.
2. A two-phase model is used as an effective upper bound to reduce the generation of candidates, thus speeding up the processing time for updating the discovered information.
3. The upper bound utility and the lower bound utility are defined as the effective thresholds for respectively deriving high (large) and pre-large transaction-weighted utilization itemsets. Based on the two defined thresholds, the original database must be rescanned only if the transaction-weighted utilization of deleted transactions is more than the safety bound calculated from the two thresholds.
4. In the proposed algorithm, only a small number of itemsets must be rescanned to maintain the transaction-weighted utilization itemsets, reducing the computational load compared to those of the batch approach [12] and the FUP2-based approach [21].

2. Review of related work

In this section, association rule mining, the pre-large concept, and high-utility mining are briefly reviewed.

2.1. Association rule mining

Traditionally, data mining techniques [1,3] are used to derive desired information from a database. The most common approach is to find the association rules [9] from a binary database based on the fact that the presence of certain items in a transaction imply the presence of other items. Agrawal and Srikant proposed the Apriori algorithm [9] to level-wise discover association rules from a static database. Apriori uses the downward closure property to prune unpromising candidates, thus improving the efficiency of discovering frequent itemsets. The Apriori algorithm consists of two main parts for generating association rules. It first uses the generate-and-test approach to find all frequent itemsets, whose counts are larger than or equal to a user-specified threshold (called the minimum support). Each frequent itemset is then level-wise combined to form association rules whose confidence values are larger than or equal to the user-specified threshold (called the minimum confidence).

In real-world applications, databases grow over time and association rules are mined in batch mode. Some new association rules may be generated and some old ones may become invalid when transactions are inserted or deleted. Traditional batch mining algorithms solve this problem by rescanning the updated database when transactions are inserted or deleted, discarding previously discovered knowledge. Cheung et al. thus proposed the FUP [14] and FUP2 [19] algorithms to respectively handle transaction insertion and transaction deletion for maintaining and updating frequent itemsets. Hong et al. then respectively applied the FUP and FUP2 concepts to maintain the FP-tree structure for handling transaction insertion [15] and transaction deletion [20]. More related works for mining association rules for transaction insertion are reviewed elsewhere [23].

2.2. Pre-large concept

Most data mining techniques have been proposed to mine the desired information based on a minimum threshold. An itemset is considered as a frequent itemset if its ratio in database is larger than or equal to the minimum support threshold. When the ratio of an itemset is nearly close to but smaller than the minimum support threshold, it is still considered as an infrequent itemset. For example, a minimum support threshold is set at 50%, an itemset is concerned as an infrequent itemset if its support ratio is 49%, which is lower than 50%. When the transactions are changed in the database whether insertion or deletion, new information may be arisen or discovered information may be missed. It is a trivial way to handle the above situation by keeping all information from the transactional database. However, it is not an efficient mechanism to keep all information from a very large database for dynamic database based on the perspective of data mining. The FUP [14] and FUP2 [19] concepts were respectively proposed to update the discovered information with transaction insertion and transaction deletion. Although FUP2 concept can be used to efficiently update the discovered information, the original database is still required to be rescanned for handling the itemsets in case 4. Pre-large concept was proposed to keep more unpromising information as the buffer to avoid the limitations of database rescan for dynamic data mining with transaction insertion [16] and transaction deletion [22]. A pre-large itemset is not really large (frequent), but has a highly probability of becoming large (frequent) after data insertion [16] or deletion [22]. Two support thresholds are used to respectively find large and pre-large itemsets for reducing the number of database rescans. The pre-large itemset acts like a buffer to reduce the movement of an itemset directly from small to large and vice versa. Lin et al. then adopted the FP-tree structure [24] and the pre-large concept for maintaining discovered frequent itemsets [17,25,26] without candidate generation. Algorithms [16,22] based on this concept rescan the database only when a certain number of transactions are changed (inserted or deleted), thus improving the performance of knowledge maintenance. When transactions are deleted from the database, nine cases arise, as shown in Fig. 1 [22].

In Fig. 1, **cases 2**, **3**, **4**, **7**, and **8** do not change the final frequent itemsets. **Case 1** may remove some existing frequent itemsets, and

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

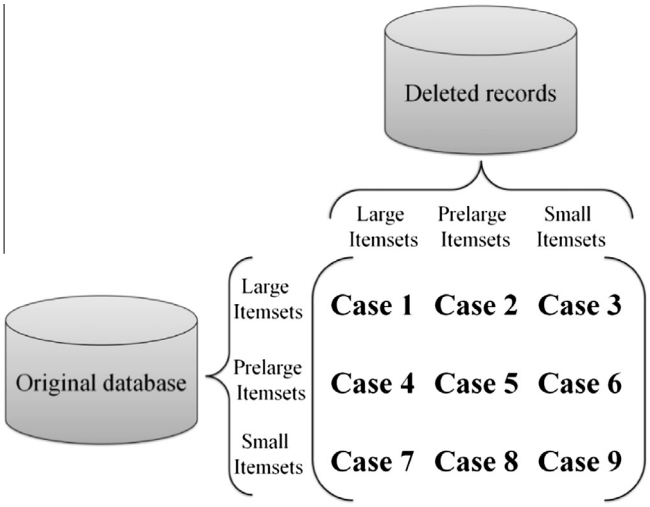

Fig. 1. Nine cases that arise due to transaction deletion with pre-large concept.

cases 5, **6**, and **9** may generate new frequent itemsets. If all large and pre-large itemsets are already mined from the original database, then **cases 1**, **5**, and **6** can be handled easily. In the maintenance process, the ratio of deleted transactions to all transactions in the database is usually very small. **Case 9** cannot possible be large for the updated databases as long as the number of deleted transactions is smaller than the safety number f:

$$f = \left\lfloor \frac{(S_u - S_l) \times d}{S_u} \right\rfloor, \tag{1}$$

where S_u is the upper threshold, S_l is the lower threshold, and d is the number of transactions in the original database. A summary of the nine cases and their merged results are shown in Table 1.

2.3. High utility mining

In this sub-section, high-utility itemsets and their algorithms are described.

2.3.1. Problem definition

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items in database D, with each item i_j having corresponding profit $p(i_j)$. An itemset $X \in I$ with k distinct items has length k and is referred to as a k-itemset. The transitional database is denoted as $D = \{T_1, T_2, \ldots, T_n\}$, where $T_d \in D$. A quantity $q(i_j, T_d)$ is the sold quantity of item i_j in transaction T_d.

Table 1
Summary of nine cases for transaction deletion.

Case	Original/Deleted	Results
Case 1	Large/Large	Large, pre-large, or small, decided from existing information
Case 2	Large/Pre-large	Large
Case 3	Large/Small	Large
Case 4	Pre-large/Large	Pre-large or small, decided from existing information
Case 5	Pre-large/Pre-large	Large, pre-large, or small, decided from existing information
Case 6	Pre-large/Small	Large or pre-large, decided from existing information
Case 7	Small/Large	Small
Case 8	Small/Pre-large	Small
Case 9	Small/Small	Pre-large or small when the number of deleted transactions is small

Definition 1. The utility of item i_j in T_d is defined as:

$$u(i_j, T_d) = q(i_j, T_d) \times p(i_j). \tag{2}$$

Definition 2. The utility of itemset X in transaction T_d is denoted as $u(X, T_d)$, which can be defined as:

$$u(X, T_d) = \sum_{i_j \in X \wedge X \subseteq T_d} u(i_j, T_d). \tag{3}$$

Definition 3. The transaction utility of transaction T_d is denoted as $TU(T_d)$, which can be defined as:

$$TU(T_d) = \sum_{j=1}^{m} u(i_j, T_d), \tag{4}$$

where m is the number of items in T_d.

Definition 4. Total utility TU^D is the sum of all transaction utilities in D, which can be defined as:

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

Definition 5. A high-utility itemset X is denoted as $HUI(X)$, which can be defined as:

$$HUI(X) \geqslant \sigma \times TU^D, \tag{6}$$

where σ is the minimum utility threshold.

Traditional utility mining [11] does not apply the downward closure property to reduce the number of candidates. Transaction-weighted utilization in the two-phase model [12] was thus proposed to keep the downward closure property for reducing the number of candidates.

Definition 6. The transaction-weighted utilization of an itemset X is the sum of all transaction utilities $TU(T_d)$ containing itemset X, which is defined as:

$$TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d). \tag{7}$$

Definition 7. A high-transaction-weighted utilization itemset X, denoted as $HTWUI(X)$, is defined as:

$$HTWUI(X) \geqslant \sigma \times TU^D. \tag{8}$$

2.3.2. High-utility itemset algorithms

Utility mining, an extension of frequent itemset mining, is based on the measurement of local transaction utility and external utility [10,11,27]. Chan et al. proposed top-k high utility closed patterns for deriving both positive and negative utilities [27]. Yao et al. proposed an algorithm for efficiently mining high-utility itemsets based on the mathematical properties of utility constraints [11]. Two pruning strategies were respectively used to reduce the search space based on the utility upper bounds and expected utility upper bounds. Liu et al. proposed a two-phase model [12] for efficiently discovering high-utility itemsets based on the downward closure property. In the first phase, the transaction-weighted utility is used as an effective upper bound for preserving the downward closure property and in the second phase, the real utility values of the remaining candidate itemsets are calculated for discovering

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

high-utility itemsets. Lin et al. proposed the high utility pattern (HUP)-tree algorithm [7] for discovering high-utility itemsets without candidate generation. The algorithm keeps the transaction-weighted utilization 1-itemsets and uses the occurrence frequency to sort the itemsets to form a condensed tree structure. Vincent et al. proposed the UP-tree structure and UP-growth and UP-growth + mining algorithms to efficiently derive high-utility itemsets from a static database [13]. Liu et al. proposed the HUI-Miner algorithm [28] that stores both the utility information of an itemset and heuristic information for pruning the search space, thus reducing the computations required for candidate generation. Lan et al. proposed an efficient utility mining approach based on the indexing mechanism to reduce the memory consumption and speed up the performance for mining high-utility itemsets. A pruning strategy is also adopted to reduce the number of unpromising itemsets in mining process [29]. Fournier-Viger et al. proposed a novel strategy to analysis the co-occurrences among items to speed up the performance by reducing the numerous join operations. For mining high-utility itemsets. Based on the designed mechanism, it outperforms the HUI-Miner with six times faster [30]. Lin et al. proposed several approaches to handle transaction insertion [18] and transaction deletion [21] for updating discovered high-utility itemsets in dynamic databases. Asha et al. also designed a Utility Pattern Tree without examining the entire database to generate the itemsets [31]. An incremental mechanism is also proposed to update the discovered HUIs in dynamic databases, which is similar to Lin's approach [18].

For transaction insertion and transaction deletion, the discovered high-utility itemsets (HUIs) and the high transaction-weighted utilization itemsets (HTWUIs) are then divided into four cases based on FUP [14] and FUP2 [19] concepts. The HUIs and HTWUIs are easily maintained and updated for cases 1, 2, and 4 except case 3 since the database is required to be rescanned for determining the transaction-weighted utility (TWU) of an itemset when it was small in the original database but large in the inserted transactions [18]. For transaction deletion [21], the HUIs and HTWUIs for cases 1, 2 and 3 are easily maintained and updated except the HTWUIS in case 4 since the database is required to be rescanned for determining the TWU of an itemset when it is small both in the original database and in the deleted transactions. Few studies have focused on mining high-utility itemsets in dynamic databases. In this paper, the pre-large concept and the TWU model are adopted to respectively reduce the number of original database rescans and the number of candidates of high-utility itemsets after transaction deletion.

3. Proposed maintenance algorithm for transaction deletion based on pre-large concept

Based on the FUP and FUP2 concepts, an additional database rescan is required for respectively handling case 3 with transaction insertion [18] and case 4 with transaction deletion [21]. This situation may frequently occur especially when the small number of transactions are processed. In this paper, the proposed algorithm is based on the pre-large concept to divide the itemsets into nine case. Each case can be easily handled to process the updating procedure. The updated results of nine cases in high-utility mining is the same as the cases in association-rule mining, which was shown in Table 1. The proof of each case of the proposed approach in high-utility mining is given below.

Lemma 1. *An itemset X will be large, pre-large or small in the updated database (U) if* $twu(X)^D \geqslant S_u \times TU^D$ *in the original database (D) and* $twu(X)^d \geqslant S_u \times TU^d$ *in the deleted transactions (d) (**Case 1**).*

Proof. Three situations of large, pre-large, and small in case 1 are proved by the contradiction proof method.

(1) The updated result is large, in which $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. Assume that $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d < twu(X)^d + S_u \times TU^D$. Since $S_u \times TU^d \leqslant twu(X)^d$, the inequality in $twu(X)^D < S_u \times TU^D$ still holds. However, it contradicts to $twu(X)^D \geqslant S_u \times TU^D$. Thus, $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$.

(2) The updated result is pre-large, in which $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$. To prove this situation, two parts can be divided as:

(i) For $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$, assume that $S_l \times (TU^D - TU^d) > twu(X)^D - twu(X)$, $S_l \times TU^D + twu(X)^d > twu(X)^D + S_l \times TU^d$. Since $twu(X)^d \geqslant S_u \times TU^d > S_l \times TU^d$ for $S_u > S_l$, the inequality in $S_l \times TU^D > twu(X)^D$ still holds. However, it contradicts to $twu(X)^D \geqslant S_l \times TU^D$. Thus, $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$.

(ii) For $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$, assume that $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d \geqslant twu(X)^d + S_u \times TU^D$. Since $twu(X)^D \geqslant S_u \times TU^D > S_l \times TU^D$ for $S_u > S_l$, the inequality in $S_u \times TU^d \geqslant twu(X)^d$ still holds. However, it contradicts to $twu(X)^d \geqslant S_u \times TU^d$. Thus, $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

By (i) and (ii), $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

(3) The updated result is small, in which $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. Assume that $twu(X)^D - twu(X)^d \geqslant S_l \times (TU^D - TU^d)$, $twu(X)^D + S_l \times TU^d \geqslant twu(X)^d + S_l \times TU^D$. Since $twu(X)^D \geqslant S_l \times TU^D$, the inequality in $S_l \times TU^d \geqslant twu(X)^d$ still holds. However, it contradicts to $S_l \times TU^d \leqslant twu(X)^d$. Thus, $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. □

Lemma 2. *An itemset X will be large in the updated database (U) if* $twu(X)^D \geqslant S_u \times TU^D$ *in the original database (D) and* $S_l \times TU^d \leqslant twu(X)^d < S_u \times TU^d$ *in the deleted transactions (d) (**Case 2**).*

Proof. The updated result is large, in which $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. By the assumption of $twu(X)^D \geqslant S_u \times TU^D$ and $S_u \times TU^d > twu(X)^d$, it can be obtained that $twu(X)^D + S_u \times TU^d \geqslant S_u \times TU^D + twu(X)^d$. Thus, $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. □

Lemma 3. *An itemset X will be large in the updated database (U) if* $twu(X)^D \geqslant S_u \times TU^D$ *in the original database (D) and* $twu(X)^d < S_l \times TU^d$ *in the deleted transactions (d) (**Case 3**).*

Proof. The updated result is large, in which $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. By the assumption of $twu(X)^D \geqslant S_u \times TU^D$ and $S_u \times TU^d > S_l \times TU^d > twu(X)^d$, it can be obtained that $twu(X)^D + S_u \times TU^d \geqslant S_u \times TU^D + twu(X)^d$. Thus, $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. □

Lemma 4. *An itemset X will be pre-large or small in the updated database (U) if* $S_l \times TU^D \leqslant twu(X)^D < S_u \times TU^D$ *in the original database (D) and* $twu(X)^d \geqslant S_l \times TU^d$ *in the deleted transactions (d) (**Case 4**).*

Proof. Two situations of pre-large or small in case 4 are proved as follows.

(1) The updated result is pre-large, in which $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$. To prove this situation, two parts can be divided as:

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

(i) For $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$, since $twu(X)^D - S_l \times TU^D \geqslant 0$, the inequality in $twu(X)^D - twu(X)^d + twu(X)^d - S_l \times TU^D \geqslant 0$ can be thus obtained. Since $twu(X)^d \text{v} S_u \times TU^d > S_l \times TU^d$ for $S_u > S_l$, the inequality in $twu(X)^D - twu(X)^d - S_l \times (TU^D - TU^d) \geqslant 0$ still holds. Thus, $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$.

(ii) For $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$, since $twu(X)^D < S_u \times TU^D$ and $S_u \times TU^d \leqslant twu(X)^d$, $twu(X)^D + S_u \times TU^d < S_u \times TU^D + twu(X)^d$. Thus, $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

By (i) and (ii), $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

(2) The updated result is small, in which $S_l \times (TU^D - TU^d) > twu(X)^D - twu(X)^d$. By the contradiction proof method, assume that $twu(X)^D - twu(X)^d \geqslant S_l \times (TU^D - TU^d)$, $twu(X)^D + S_l \times TU^d \geqslant twu(X)^d + S_l \times TU^D$. Since $twu(X)^D \geqslant S_l \times TU^D$, the inequality in $S_u \times TU^d > S_l \times TU^d \geqslant twu(X)^d$ still holds. However, it contradicts to $twu(X)^d \geqslant S_u \times TU^d$. Thus, $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. □

Lemma 5. *An itemset X will be large, pre-large or small in the updated database (U) if $S_l \times TU^D \leqslant twu(X)^D < S_u \times TU^D$ in the original database (D) and $S_l \times TU^d \leqslant twu(X)^d < S_u \times TU^d$ in the deleted transactions (d) (**Case 5**).*

Proof. Three situations of large, pre-large, and small in case 5 are proved by the contradiction proof method.

(1) The updated result is large, in which $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. Assume that $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d < twu(X)^d + S_u \times TU^D$. Since $twu(X)^D < S_u \times TU^D$, the inequality in $S_u \times TU^d < twu(X)^d$ still holds. However, it contradicts to $S_u \times TU^d > twu(X)^d$. Thus, $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$.

(2) The updated result is pre-large, in which $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$. To prove this situation, two parts can be divided as:

(i) For $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$, assume that $S_l \times (TU^D - TU^d) > twu(X)^D - twu(X)^d$, $twu(X)^d + S_l \times TU^D > twu(X)^D + S_l \times TU^d$. Since $twu(X)^d > S_l \times TU^d$, the inequality in $S_l \times TU^D > twu(X)^D$ still holds. However, it contradicts to $S_l \times TU^D \leqslant twu(X)^D$. Thus, $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$.

(ii) For $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$, assume that $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d \geqslant twu(X)^d + S_u \times TU^D$. Since $twu(X)^d < S_u \times TU^d$, the inequality in $twu(X)^D \geqslant S_u \times TU^D$ still holds. However, it contradicts to $twu(X)^D < S_u \times TU^D$. Thus, $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

By (i) and (ii), the updated result is pre-large.

(3) The updated result is small, in which $S_l \times (TU^D - TU^d) < twu(X)^D - twu(X)^d$. Assume that $twu(X)^D - twu(X)^d \geqslant S_l \times (TU^D - TU^d)$, $twu(X)^D + S_l \times TU^d \geqslant twu(X)^d + S_l \times TU^D$. Since $twu(X)^D \geqslant S_l \times TU^D$, the inequality in $S_l \times TU^d \geqslant twu(X)^d$ still holds. However, it contradicts to $S_l \times TU^d < twu(X)^d$. Thus, $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. □

Lemma 6. *An itemset X will be large or pre-large in the updated database (U) if $S_l \times TU^D \leqslant twu(X)^D < S_u \times TU^D$ in the original database (D) and $S_l \times TU^d > twu(X)^d$ in the deleted transactions (d) (**Case 6**).*

Proof. Two situations of large or pre-large in case 6 are proved as follows.

(1) The updated result is large, in which $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$. By the contradiction proof method, assume that $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d < twu(X)^d + S_u \times TU^D$. Since $twu(X)^D < S_u \times TU^D$, the inequality in $S_u \times TU^d < twu(X)^d$ still holds. However, it contradicts to $twu(X)^d < S_l \times TU^d < S_u \times TU^d$ for $S_u > S_l$. Thus, $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$.

(2) The updated result is pre-large, in which $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$. To prove this condition, it can divided into two parts as:

(i) For $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$, $S_l \times TU^D + twu(X)^d \leqslant twu(X)^D + S_l \times TU^d$. Thus, $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$.

(ii) For $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$. By the contradiction proof method, assume that $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d \geqslant twu(X)^d + S_u \times TU^D$. Since $twu(X)^d < S_l \times TU^d < S_u \times TU^d$ for $S_u > S_l$, the inequality in $twu(X)^D \geqslant S_u \times TU^D$ still holds. However, it contradicts to $twu(X)^D < S_u \times TU^D$. Thus, $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

By (i) and (ii), the updated result is pre-large. □

Lemma 7. *An itemset X will be small in the updated database (U) if its $twu(X)^D < S_l \times TU^D$ in the original database (D) and its $twu(X)^d \geqslant S_u \times TU^d$ in the deleted transactions (d) (**Case 7**).*

Proof. By the assumption, it can be obtained that $twu(X)^d \geqslant S_u \times TU^d > S_l \times TU^d$ for $S_u \geqslant S_l$. $twu(X)^d + S_l \times TU^D > S_l \times TU^d + twu(X)^D$. Thus, $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. □

Lemma 8. *An itemset X will be small in the updated database (U) if $twu(X)^D < S_l \times TU^D$ in the original database (D) and $S_l \times TU^d \leqslant twu(X)^d < S_u \times TU^d$ in the deleted transactions (d) (**Case 8**).*

Proof. By the assumption, it can be obtained that $twu(X)^D + S_l \times TU^d < S_l \times TU^D + twu(X)^d$ except $S_l \times TU^d = twu(X)^d$. Thus, $twu(X)^D - twu(X)^d < S_l \times TU^D - TU^d)$. □

Lemma 9. *An itemset X will be small in the updated database (U) if $twu(X)^D < S_l \times TU^D$ in the original database (D) and $twu(X)^d < S_l \times TU^d$ in the deleted transactions (d) (**Case 9**).*

Proof. Two situations of pre-large or small in case 9 are proved by using the contradiction proof method.

(1) The updated result is pre-large in which $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d < \times (TU^D - TU^d)$. It can divide into two parts as:

(i) For $S_l \times (TU^D - TU^d) \leqslant twu(X)^D - twu(X)^d$, assume that $S_l \times (TU^D - TU^d) > twu(X)^D - twu(X)^d$, $S_l \times TU^D + twu(X)^d > twu(X)^D + S_l \times TU^d$. Since $S_l \times TU^D > twu(X)^D$, the inequality in $twu(X)^d > S_l \times TU^d$ still holds. However, it contradicts to $twu(X)^d < S_l \times TU^d$. Thus, $twu(X)^D - twu(X)^d \leqslant S_u \times (TU^D - TU^d)$.

(ii) For $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$. Assume that $twu(X)^D - twu(X)^d \geqslant S_u \times (TU^D - TU^d)$, $twu(X)^D + S_u \times TU^d \geqslant twu(X)^d + S_u \times TU^D$. Since $twu(X)^d < S_l \times TU^d < S_u \times TU^d$ for $S_l < S_u$, $twu(X)^D \geqslant S_u \times TU^D$. However, it contradicts to $twu(X)^D < S_l \times TU^D < S_u \times TU^D$. Thus, $twu(X)^D - twu(X)^d < S_u \times (TU^D - TU^d)$.

By (i) and (ii), the updated result is pre-large.

(2) The updated result is small in which $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. Assume that $twu(X)^D - twu(X)^d \geqslant S_l \times (TU^D - TU^d)$, $twu(X)^D + S_l \times TU^d \geqslant S_l \times TU^D + twu(X)^d$. Since $S_l \times TU^d \geqslant twu(X)^d$, the inequality in $twu(X)^D \geqslant S_l \times TU^D$ still holds. However, it contradicts to $twu(X)^D < S_l \times TU^D$. Thus, $twu(X)^D - twu(X)^d < S_l \times (TU^D - TU^d)$. □

When transactions are deleted from the original database, if the total utility of deleted transactions is small compared to that of the transactions in the original database, the transaction-weighted utilization of an itemset that is small (neither large nor pre-large) both in the original database and in the deleted transactions cannot become large for the updated database (in Case 9). This is proven in Theorem 1.

Theorem 1. *Let S_u and S_l be respectively the upper and lower utility thresholds, and let TU^D and TU^d be respectively the total utilities of the original database and the deleted transactions. If $TU^d \leqslant \frac{(S_u - S_l)}{S_u} \times TU^D$, the transaction-weighted utilization of an itemset that is small (neither large nor pre-large) both in the original database and the deleted transactions does not become large for the updated database.*

Proof. From $TU^d \leqslant \frac{(S_u - S_l)}{S_u} \times TU^D$, the following derivations can be obtained:

$$TU^d \leqslant \frac{(S_u - S_l)}{S_u} \times TU^D \Rightarrow S_u \times TU^d \leqslant (S_u - S_l) \times TU^D$$
$$\Rightarrow S_u \times TU^d \leqslant S_u \times TU^D - S_l \times TU^D \Rightarrow S_l \times TU^D$$
$$\leqslant S_u \times (TU^D - TU^d) \Rightarrow \frac{S_l \times TU^D}{(TU^D - TU^d)} \leqslant S_u. \quad \square \tag{9}$$

Then, if the transaction-weighted utilization of an itemset X is small (neither large nor pre-large) in the original database D, its transaction-weighted utilization $twu^D(X)$ in the original database D will be less than $S_l \times TU^D$; therefore:

$$twu^D(X) < S_l \times TU^D. \tag{10}$$

If the transaction-weighted utilization of an itemset X is small in the deleted transactions d, its transaction-weighted utilization $twu^d(X)$ in the deleted transactions d will be less than $S_l \times TU^d$; therefore:

$$twu^d(X) < S_l \times TU^d. \tag{11}$$

The ratio of an itemset X in the updated database U is calculated as $\frac{twu^U(X)}{TU^D + TU^d}$, thus the updated ratio of an itemset X is always small as follows:

$$\frac{twu^D(X) - twu^d(X)}{TU^D - TU^d} \leqslant \frac{twu^D(X)}{TU^D - TU^d} < \frac{S_l \times TU^D}{TU^D - TU^d} \leqslant S_u. \tag{12}$$

It can be found that the transaction-weighted utilization of itemset X does not become large for the entire updated database when the total utility TU^d of the deleted transactions is smaller than or equal to $\frac{(S_u - S_l) \times TU^D}{S_u}$. Thus, the following definition can be given.

Definition 8. An itemset X cannot be large in the updated database if the total utility in the deleted transactions is less than the safety bound, which can be denoted as:

$$TU^d < f = \frac{(S_u - S_l) \times TU^D}{S_u}. \tag{13}$$

Example. Assume that the total utility of the original database is 100 ($TU^D = 100$), and that the lower and upper utility thresholds are respectively set at 20% and 40% ($S_l = 0.2$, $S_u = 0.4$). The total utility of the original database does not need to be scanned for rule maintenance as:

$$\frac{(S_u - S_l) \times TU^D}{S_u} = \frac{(0.4 - 0.2) \times 100}{0.4} = 50.$$

Thus, if the total utility of deleted transactions is less than or equal to 50, an itemset X is definitely not large for the entire updated database. From Theorem 1, the itemsets in case 9 can be efficiently handled using the total utility of the deleted transactions, the upper and lower utility thresholds based on the pre-large concept, and the total utility of the original database.

When transactions are deleted from the original database, the proposed PRE-HUI-DEL algorithm is executed to maintain the discovered high-transaction-weighted utilization itemsets. Firstly, the candidate 1-itemsets in the deleted transactions are obtained with their transaction-weighted utilization values and actual utility values. Three parts with nine cases are then arisen compared to the original database and the deleted transactions. Each part is then processed with its own procedure to update the large (high) and pre-large transaction-weighted utilization 1-itemsets in the updated database. Note that both the transaction-weighted utilizations and the actual utility values of the generated candidate 1-itemsets are updated at the same time. An Apriori-like approach [9] is then used to generate the next candidate itemsets from the discovered large (high) and pre-large transaction-weighted utilization itemsets level by level until all high-utility itemsets have been maintained and updated. The notation and the details of the designed algorithm are illustrated below. The variable *buf* is designed to preserve the total utility value of the deleted transactions at the last rescan of the original database.

3.1. Notations

I	A set of m items, $I = \{i_1, i_2, \ldots, i_m\}$, in which each item i_j has its own profit value $p(i_j)$
D	Original quantitative database, $D = \{T_1, T_2, \ldots, T_n\}$, in which each transaction T_d contains several items with sold quantities
d	Deleted transactions extracted from the original database
U	Entire updated database, i.e., $D - d$
TU^D	Total utility of the transactions in D
TU^d	Total utility of the deleted transactions in d
TU^U	Total utility of the transactions in U
$q(i_j, T_d)$	Quantity of item i_j in transaction T_d
$u(i_j, T_d)$	Utility of item i_j in transaction T_d calculated as $q(i_j, T_d) \times p(i_j)$
$TU(T_d)$	Transaction utility of transaction T_d
buf	Stores the total utility of the last processed transactions for transaction deletion. It is set to 0 when the database is rescanned
X	An itemset containing k distinct items
S_u	Upper utility threshold for large (high) transaction-weighted utilization and high-utility itemsets. It is the same as the high utility threshold in traditional utility mining

S_l	Lower utility threshold for pre-large transaction-weighted utilization and pre-large utility itemsets, $S_u > S_l$
f	Safety transaction utility bound for deleted transactions
C_r	Set of candidate r-itemsets
$HTWUI_r^D$	Set of large (high) transaction-weighted utilization r-itemsets in the original database
$PTWUI_r^D$	Set of pre-large transaction-weighted utilization r-itemsets in the original database
$HTWUI^D$	Set of large (high) transaction-weighted utilization itemsets in the original database
$PTWUI^D$	Set of pre-large transaction-weighted utilization itemsets in the original database
$HTWUI_r^U$	Set of large (high) transaction-weighted utilization r-itemsets in the updated database
$PTWUI_r^U$	Set of pre-large transaction-weighted utilization r-itemsets in the updated database
$HTWUI^U$	Set of large (high) transaction-weighted utilization itemsets in the updated database
$PTWUI^U$	Set of pre-large transaction-weighted utilization itemsets in the updated database
HUI^U	Set of high-utility itemsets in the updated database
$twu^D(X)$	Transaction-weighted utilization of itemset X in the original database
$twu^d(X)$	Transaction-weighted utilization of itemset X in the deleted transactions
$twu^U(X)$	Transaction-weighted utilization of itemset X in the updated database
$au^D(X)$	Actual utility of itemset X in the original database
$au^d(X)$	Actual utility of itemset X in the deleted transactions
$au^U(X)$	Actual utility of itemset X in the updated database

3.2. Proposed algorithm

INPUT: A profit table, a quantitative database D, an upper support threshold S_u, a lower support threshold S_l, total utility TU^D in D, a set of large (high) transaction-weighted utilization itemsets $HTWUI^D$ with their actual utilities, a set of pre-large transaction-weighted utilization itemsets $PTWUI^D$ with their actual utilities, safety buffer buf, a set of deleted transactions d from D.

OUTPUT: Updated $HTWUI^U$, $PTWUI^U$, and HUI^U.

1. Set $buf = 0$;
2. Calculate safety bound $f := \frac{(S_u - S_l) \times TU^D}{S_u}$.
3. Calculate TU^d in d.
4. Calculate updated $TU^U := TU^D - TU^d$.
5. ***IF*** $(buf + TU^d) \leqslant f$, ***THEN***
   $buf += TU^d$;
   Find $TWUI_1^d$ as C_1.
   OTHERWISE
   Rescan U to find updated $HTWUI^U$ and $PTWUI^U$;
   Set $HTWUI^D := HTWUI^U$, $PTWUI^D := PTWUI^U$;
   Set $HUI^D := HUI^U$;
   Set $buf := 0$;
   Terminate.
6. Scan d to find the transaction-weighted utilization 1-itemsets $TWUI_1^d$.
7. Set $r := 1$.
8. ***FOR*** each r-itemset X in C_r ***DO***
9. Find $twu^d(X)$, $au^d(X)$.
10. ***END FOR***
11. ***FOR*** each r-itemset in $HTWUI_r^D$ ***DO*** //Cases 1, 2, and 3
12. $twu^U(X) := twu^D(X) - twu^d(X)$;
13. $au^U(X) := au^D(X) - au^d(X)$.
14. ***IF*** $\frac{twu^U(X)}{TU^U} \geqslant S_u$ ***THEN***
15. Put X in $HTWUI_r^U$.
16. ***OTHERWISE IF*** $S_l \leqslant \frac{twu^U(X)}{TU^U} \leqslant S_u$ ***THEN***
17. Put X in $PTWUI_r^U$.
18. ***END IF***
19. ***END FOR***
20. ***FOR*** each r-itemset in $PTWUI_r^D$ ***DO*** //Cases 4, 5, and 6
21. $twu^U(X) := twu^D(X) - twu^d(X)$;
22. $au^U(X) := au^D(X) - au^d(X)$.
23. ***IF*** $\frac{twu^U}{TU^U} \geq S_U$ ***THEN***
24. Put X in $HTWUI_r^U$.
25. ***OTHERWISE IF*** $S_l \leqslant \frac{twu^U(X)}{TU^U} \leqslant S_u$ ***THEN***
26. Put X in $PTWUI_r^U$.
27. ***END IF***
28. ***END FOR***
29. Set r += 1.
30. ***REPEAT*** STEPS 8 to 29 until no updated large (high) or pre-large transaction-weighted utilization itemsets are found.
31. Calculate $au^U(X)$ in $HTWUI^U$.
32. ***IF*** $au^U(X) \geqslant S_u$ ***DO***
33. Put X in HUI^U.
34. ***END IF***
35. Set $HTWUI^D := HTWUI^U$.
36. Set $PTWUI^D := PTWUI^U$.

4. An example

In this section, an example is provided to illustrate the proposed PRE-HUI-DEL algorithm step by step. The original database is shown in Table 2. It consists of 12 transactions with 5 items, denoted by A to E, respectively.

The upper and lower utility thresholds are respectively set to 30% and 15%. Note that the upper utility threshold is the same as the minimum high utility threshold in traditional utility mining. The upper and lower thresholds can be adjusted by use's preference. The gap between upper and lower thresholds should not be large since only few unpromising itemsets with high probabilities to be large in the nearly future. The profit table for the items is given in Table 3.

Firstly, the two-phase algorithm [12] is performed to find large (high) transaction-weighted utilization itemsets and pre-large transaction-weighted utilization itemsets with their actual utilities using the two utility thresholds. The results are respectively shown in Tables 4 and 5.

Suppose that the last four transactions from Table 2 are chosen as the deleted transactions from the original database. The execution process is as follows. Since there were no last processed transactions in the past, buf is initially set at 0. The safety transaction utility bound is calculated to evaluate whether the original database should be rescanned. It is calculated as $\frac{S_u - S_l}{S_u} \times TU^D = \frac{0.3-0.15}{0.3} \times 560 (= 280)$, where S_u and S_l are respectively the upper and lower utility thresholds, and TU^D is the total utility in the original database D.

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

The utility of each item in the last four deleted transactions in Table 2 is first found. Take transaction 9 as an example. The items and their quantities in transaction 9 are (B: 2, C: 3, D: 7). The profits for the three 1-itemsets (B), (C), and (D) are 2, 15, and 7, respectively, from Table 3. The transaction utility for transaction 9 is calculated as $TU(9) = (2 \times 2) + (3 \times 15) + (7 \times 7)$ (= 98). The other transactions are calculated in the same way. The results are shown in Table 6. The total utility of the four deleted transactions TU^d is then calculated as 98 + 10 + 75 + 27 = 210.

The total utility of the updated database is updated as (560 − 210 = 350). Since there were no last processed transactions in the past, *buf* is initially set at 0. The total utility in the deleted transactions TU^d was calculated as 210, which is smaller than the safety transaction utility bound (f = 280). The original database is thus not rescanned. The following steps are used to find the high-utility itemsets.

First, the items that appear in the deleted transactions are found as the candidate 1-itemsets, which are (B, C, D, E). r is initially set at 1. The transaction-weighted utilization and the actual utility of each candidate 1-itemset in the deleted transactions are thus calculated. Take 1-itemset (B) as an example. 1-itemset (B) appears in transactions 9, 11, and 12. $twu^d(B)$ is the sum of the transaction utilities of the above three transactions (98 + 75 + 27 = 200). The actual utility of $au^d(B)$ is also calculated at the same time, which is $(2 \times 2) + (5 \times 2) + (3 \times 2) = 20$. The other items are calculated in the same way. The results are shown in Table 7.

Each 1-itemset in the set of large (high) transaction-weighted utilization 1-itemsets $HTWUI_1^D$ in the original database is then processed. In this example, three 1-itemsets (B, C, D) are next processed. The transaction-weighted utilization of 1-itemset (B) in the original database is ($twu^D(B)$ = 287), as shown in Table 4. The transaction-weighted utilization of 1-itemset (B) in the deleted transactions is 200. The updated transaction-weighted utilization of 1-itemset (B) is then updated as (287 − 200 = 87). The actual utility of 1-itemset (B) in the original database is ($au^D(B)$ = 44), as shown in Table 4. Its actual utility in the deleted transactions is ($au^d(B)$ = 20). The updated actual utility of 1-itemset (B) was thus calculated as ($au^U(B) = 44 - 20 = 24$). 1-itemsets (C) and (D) are processed in the same way. The results are shown in Table 8.

In this example, the updated transaction-weighted utilization ratio of 1-itemset (B) is calculated as (87/350 = 24.8%), which is larger than the lower utility threshold (15%) but smaller than the upper utility threshold (30%). 1-itemset (B) is thus concerned as a pre-large transaction-weighted utilization itemset after the database is updated. 1-itemset (B) is put into the set $PTWUI_1^U$. The updated transaction-weighted utilization ratio of 1-itemset (C) is calculated as (147/350 = 42%), which is larger than the upper utility threshold. 1-itemset (C) is put into the set $HTWUI_1^U$. The updated transaction-weighted utilization ratio of 1-itemset (D) is calculated as (67/350 = 19.1%), which is larger than the lower utility threshold but smaller than the upper utility threshold. 1-itemset (D) is put into the set $PTWUI_1^U$. Then, $HTWUI_1^U$ = {C} and $PTWUI_1^U$ = {B, D}.

Each 1-itemset in the set of pre-large transaction-weighted utilization 1-itemsets $PTWUI_1^D$ in the original database is then processed. In this example, 1-itemsets (A) and (E) are processed. The transaction-weighted utilization of 1-itemset (A) in the original database is ($twu^D(A)$ = 108), as shown in Table 5. The transaction-weighted utilization of 1-itemset (A) in the deleted transactions is 0. The updated transaction-weighted utilization of 1-itemset (A) is updated as (108 − 0 = 108). The actual utility of 1-itemset (A) in the original database is ($au^D(A)$ = 48), as shown in Table 5. Its actual utility in the deleted transactions is 0. The updated actual utility of 1-itemset (A) is thus calculated as ($au^U(A) = 48 - 0 = 48$). The results are shown in Table 9.

In this example, the updated transaction-weighted utilization ratio of 1-itemset (A) is calculated as (108/350 = 30.8%), which is larger than the upper utility threshold (30%). The updated transaction-weighted utilization ratio of 1-itemset (E) is calculated as (152/350 = 43.4%), which is larger than the upper utility threshold.

Table 2
Original database in the example.

TID	A	B	C	D	E
1	6	0	0	0	0
2	0	1	0	5	0
3	0	6	0	0	0
4	0	0	5	0	0
5	0	0	0	0	8
6	2	0	2	0	3
7	0	1	0	4	0
8	0	4	0	0	0
9	0	2	3	7	0
10	0	0	0	0	1
11	0	5	2	5	0
12	0	3	0	3	0

Table 3
Profit table.

Item	Profit ($)
A	6
B	2
C	15
D	7
E	10

Table 4
$HTWUI^D$ and their actual utilities.

Itemset	*twu*	*au*
B	287	44
C	320	180
D	267	168
BC	173	89
BD	267	192
CD	173	159
BCD	173	173

Table 5
$PTWUI^D$ and their actual utilities.

Itemset	*twu*	*au*
A	108	48
E	162	120

Table 6
Transaction utilities for the last four deleted transactions.

TID	A	B	C	D	E	TU
9	0	2	3	7	0	98
10	0	0	0	0	1	10
11	0	5	2	5	0	75
12	0	3	0	3	0	27

Table 7
twu values for deleted transactions of 1-itemsets.

1-Itemset	*twu*	*au*
B	200	20
C	173	75
D	200	105
E	10	10

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

1-itemset (E) is put into the set $HTWUI_1^U$. Then, $HTWUI_1^U = \{A, C, E\}$ and $PTWUI_1^U = \{B, D\}$.

The large (high) transaction-weighted utilization and the pre-large transaction-weighted utilization 1-itemsets and their actual utilities in the updated database are shown in Table 10.

The candidate 2-itemsets are then formed from Table 10 using an Apriori-like approach [9]. The generated results are (AB, AC, AD, AE, BC, BD, BE, CD, CE, DE). The variable r is set at 2. The above steps are repeated until no candidate itemsets are generated. The final results are shown in Table 11.

Finally, the large (high) transaction-weighted utilization itemsets in Table 11 are determined to evaluate whether they are high-utility itemsets in the updated database. Take 1-itemset (A) as an example. The updated actual utility for 1-itemset (A) is 48; its ratio in the updated database is calculated as (48/350 = 13.7%), which is smaller than the lower utility threshold. 1-itemset (A) is thus not a high-utility itemset after the database is updated. The other large (high) transaction-weighted utilization itemsets in Table 11 are processed in the same way. The results are shown in Table 12.

In this example, the set $HTWUI^U = \{A, C, E\}$ and the set $PTWUI^U = \{B, D, AB, AC, BD, CE, ACE\}$. They are considered as the set of large (high) transaction-weighted utilization $HTWUI^D$ and the set of pre-large transaction-weighted utilization $PTWUI^D$, respectively, for the next transaction deletion. The final results for the high-utility itemsets are thus $\{C, E\}$.

5. Experimental results

Experiments were implemented in the Java language and executed on a PC with a 3.0-GHz CPU and 4 GB of memory. Two databases were used in the experiments, namely a simulation database created using the IBM data generator [35] and the real-world *foodmart* database [36]. A simulation model was developed to generate the quantities of the items in the transactions for the generated database by IBM data generator, which was similar to that used in Liu et al. [12]. The model is extended to generate the quantity values of the items in the sequences. Each quantity ranged among 1–5 following the way described in [12]. In addition, for each database generated, a corresponding utility table was also produced in which a profit value in the range from 0.01 to 10.00 was randomly assigned to an item. In real-world databases, most items are in the low profit range, the log normal distribution is used to generate the utility values. The *foodmart* database is a quantitative database of products sold by an anonymous chain store. There are 21,556 transactions and 1559 items in the database. In the experiments, the two-phase high utility mining (TP-HUI) algorithm [12], the high utility mining algorithm based on the FUP concept for transaction deletion (FUP-HUI-DEL) [21], and the proposed PRE-HUI-DEL algorithm were compared. When transactions are deleted from the original database, the TP-HUI algorithm has to rescan the updated database to extract the updated high-utility itemsets in batch mode. The FUP-HUI-DEL algorithm divides the itemsets into four sets according to whether their transaction-weighted utilizations are large or small in the original database and in the deleted transactions. Each set is then processed separately to update the discovered knowledge. The itemset which has small transaction-weighted utilization both in the deleted transactions and in the original databases is required to re-scan the whole database for maintaining the transactions-weighted utilization of the discovered rules.

It is a non-trivial task to set the appropriate thresholds for different characteristics of databases, several top-k [27,32] or bio-inspired algorithms [33,34] have been developed to mine the required information without minimum support threshold. Since it is another research issue in data mining, the upper and lower utility thresholds are manually set in the conducted experiments. To show the performance of the proposed PRE-HUI-DEL algorithm, the gap is set small between the upper and lower utility thresholds. Only fewer promising itemsets with extremely high probability will be kept, thus avoiding the memory consumption of the proposed approach.

5.1. Experimental results for simulation database

The IBM data generator was used to generate a simulation database called T10I4N4KD200K (T is the average length of items in a transaction, I is the average length of maximal potentially frequent itemsets, N is the total number of items, and D is the total number of transactions). Firstly, 200,000 transactions were used to initially mine the large (high) and pre-large transaction-weighted utilization itemsets with their actual utility values. Each 2000 transac-

Table 8
Updated transaction-weighted utilization 1-itemsets in HTWUI1 in D.

1-Itemset	*twu*	*au*	Ratio (%)	Updated result
B	87	24	24.8	Pre-large
C	147	105	42.0	Large
D	67	63	19.1	Pre-large

Table 9
Updated transaction-weighted utilization 1-itemsets in PTWUI1 in D.

1-Itemset	*twu*	*au*	Ratio (%)	Updated result
A	108	48	30.8	Large
E	152	110	43.4	Large

Table 10
Results for $HTWUI_1^U$ and $PTWUI_1^U$.

	Itemset	*twu*	*au*
$HTWUI_1^U$	{A}	108	48
	{C}	147	105
	{E}	152	110
$PTWUI_1^U$	{B}	87	24
	{D}	67	63

Table 11
Final results for $HTWUI^U$ and $PTWUI^U$ and their actual utilities.

	Itemset	*twu*	*au*
$HTWUI^U$	A	108	48
	C	147	105
	E	152	110
$PTWUI^U$	B	87	24
	D	67	63
	AC	72	42
	AE	72	42
	BD	67	67
	CE	72	60
	ACE	72	72

Table 12
Final results.

	Itemset	*au*	Ratio (%)	HUI
$HTWUI^U$	A	48	13.7	–
	C	105	30.0	Yes
	E	110	31.4	Yes

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

tions were then sequential deleted from the original databases bottom up from the transactions at each time. The minimum high utility threshold (upper utility threshold) was set at 0.2% to evaluate the performance of the TP-HUI and FUP-HUI-DEL algorithms. The lower utility threshold was set at 0.18%. Fig. 2 shows the execution times for the TP-HUI, FUP-HUI, and PRE-HUI-DEL algorithms. The 2000 transactions were then sequentially deleted from the original database to update the database.

As shown in Fig. 2, the TP-HUI algorithm has to process the updated database in batch mode whenever transactions are deleted. The FUP-HUI-DEL algorithm rescans the whole database only if it is necessary to re-calculate the itemsets in case 4. The proposed PRE-HUI-DEL algorithm reduces the rescanning time of original database, and thus ran fastest for transaction deletion. In the last process (accumulated deleted transactions reaches 18 K) of the PRE-HUI-DEL algorithm reaches, however, the number of safety transaction utility bound. The database is rescanned to find the required high-utility itemsets.

Experiments were also made to evaluate the efficiency of the proposed PRE-HUI-DEL algorithm for various minimum high utility threshold values. The results are shown in Fig. 3. The minimum high utility threshold (upper utility threshold) was varied from 0.2% to 0.6% in 0.1% increments. The lower utility threshold for the proposed algorithm was varied from 0.18% to 0.058%, decreases 0.02% each time of the upper utility threshold.

The execution time of the proposed PRE-HUI-DEL algorithm is much less than those of the TP-HUI and FUP-HUI-DEL algorithms for handling transaction deletion with various minimum high utility thresholds. Experiments were then made to evaluate the efficiency of the proposed PRE-HUI-DEL algorithm for various numbers of deleted transactions. The upper utility threshold (minimum high utility threshold) and the lower utility threshold were respectively set at 0.2% and 0.18%. The numbers of deleted transactions were 4000, 8000, 12,000, 16,000, and 20,000. The results are shown in Fig. 4.

The execution time of the proposed PRE-HUI-DEL algorithm is lower than those of the TP-HUI and FUP-HUI-DEL algorithms for handling transaction deletion for various numbers of deleted transactions.

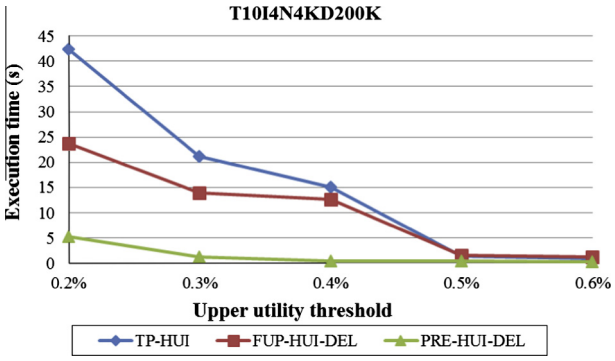

Fig. 3. Execution time for various minimum utility thresholds for simulated database.

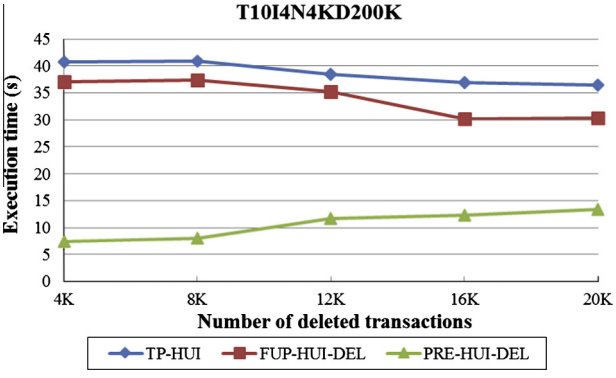

Fig. 4. Execution time for various numbers of deleted transactions in simulated database.

5.2. Experimental results for foodmart database

The *foodmart* database was also used for comparing the three algorithms. The first 21,556 transactions were initially used to mine the large (high) and pre-large transaction-weighted utilization itemsets and their actual utility values. The minimum high utility threshold (upper utility threshold) was set at 0.01%. The lower utility threshold was set at 0.0095%. Fig. 5 shows the execution times of the three algorithms. The 100 transactions were then randomly deleted from the original database.

The proposed PRE-HUI-DEL algorithm is fastest for transaction deletion. In the experiments, the total utility in *foodmart* is calculated as 104,450,739. The total utility for the selected 600 transactions for deletion is calculated as 2,013,024 and the total utility for the selected 700 transactions for deletion is calculated as 2,636,302. The safety of the proposed PRE-HUI-DEL is calculated as:

$$f = \frac{(S_u - S_l) \times TU^D}{S_u} = \frac{(0.01\% - 0.0098\%) \times 104,450,739}{0.01\%} = 2089014.78.$$

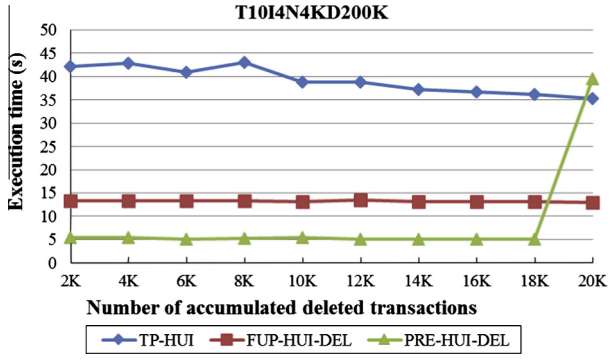

Fig. 2. Execution times for accumulated deleted transactions in simulated database.

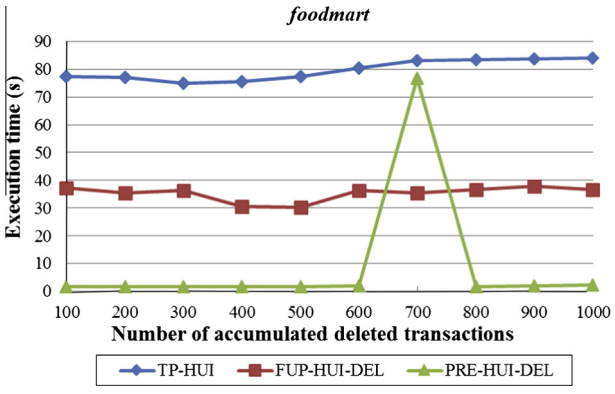

Fig. 5. Execution time for accumulated deleted transactions in *foodmart* database.

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003

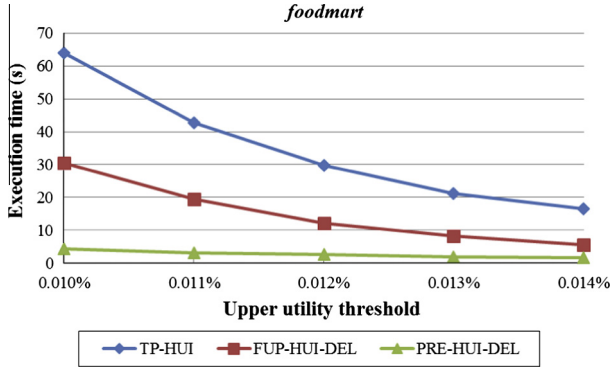

Fig. 6. Execution time for various minimum utility thresholds for *foodmart* database.

From the experiments, it can be observed that the database is required to be rescanned at the seventh process since the total utility for the selected 600 deleted transactions is calculated as 2,013,024, and the total utility for the selected 700 deleted transactions is calculated as 2,636,302, thus:

$$2,013,024 < 2,089,014 < 2,636,302.$$

Experiments were made to evaluate the efficiency of the proposed PRE-HUI-DEL algorithm for various minimum high utility threshold values. The results are shown in Fig. 6. The minimum high utility threshold (upper utility threshold) was set from 0.01% to 0.014% in 0.001% increments. The lower utility threshold for the proposed algorithm was set from 0.0098% to 0.0138%, decreases 0.0002% at each time of the upper utility threshold.

The execution time of the proposed PRE-HUI-DEL algorithm is much lower than those of the TP-HUI and FUP-HUI-DEL algorithms for handling transaction deletion at various minimum high utility thresholds for the *foodmart* database. Experiments were made to evaluate the efficiency of the proposed PRE-HUI-DEL algorithm for various numbers of deleted transactions. The minimum high utility threshold (upper utility threshold) and the lower utility threshold were respectively set at 0.01% and 0.0098%. The numbers of deleted transactions were 200, 400, 600, 800, and 1000. The results are shown in Fig. 7.

The execution time of the proposed PRE-HUI-DEL algorithm is lower than those of the TP-HUI and FUP-HUI-DEL algorithms for handling transaction deletion with various numbers of deleted transactions for the *foodmart* database. For 1000 deleted transactions, the proposed algorithm had to rescan the database to maintain and update the high-utility itemsets.

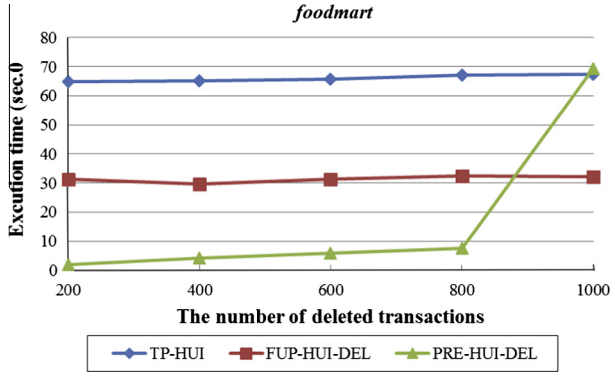

Fig. 7. Execution time for various numbers of deleted transactions for *foodmart* database.

6. Conclusion and future work

This paper proposed a high utility mining algorithm based on pre-large concepts for transaction deletion (PRE-HUI-DEL) for efficiently maintaining and updating discovered high-transaction-weighted utilization itemsets to derive high-utility itemsets. When transactions are removed from the original database, the proposed PRE-HUI-DEL algorithm partitions the itemsets in the deleted transactions into three sets with nine cases according to whether they have large (high), pre-large, or small transaction-weighted utilization in the original database and in the deleted transactions. Each set is then processed separately to maintain the discovered high-utility itemsets. When the total utility value of the inserted transactions is smaller than the safety transaction utility bound, the high-utility itemsets are directly updated without a database rescan, reducing computational time. Experimental results show that the proposed PRE-HUI-DEL algorithm outperforms existing high utility mining algorithms. When the accumulative total utility in the deleted transactions archives the safety bound of pre-large concept, the database is required to be rescanned for determining the TWU values of the itemsets in case 9. From the conducted experiments, the original database is unnecessary to be rescanned each time, thus reducing the computations compared to the traditional two-phase approach and the FUP-HUI-DEL algorithm.

Traditional data mining requires the minimum support threshold to define the number of desired information, which is not practical in real-world applications. Since it is a non-trivial task to set an appropriate threshold for mining the desired information, the maintenance approach for efficiently updating the discovered information without minimum support threshold should be discussed in the nearly future. It can help solve the limitations of the traditional mining approaches especially when the characteristics of the compared databases are widely different. Besides, transaction modification is also considered as another research issue in dynamic data mining. How to efficiently improve the performance with an efficient tree structure for reducing the computations compared to the level-wise approach will be developed in the nearly future work.

Acknowledgment

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

References

[1] R. Agrawal, T. Imielinski, A. Swami, Database mining: a performance perspective, IEEE Trans. Knowl. Data Eng. 5 (1993) 914–925.
[2] P. Berkhin, A survey of clustering data mining techniques, Group. Multidimen. Data (2006) 25–71.
[3] M.S. Chen, J. Han, P.S. Yu, Data mining: an overview from a database perspective, IEEE Trans. Knowl. Data Eng. 8 (1996) 866–883.
[4] S. B. Kotsiantis, Supervised machine learning: a review of classification techniques, in: The Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies, 2007, pp. 3–24.
[5] C.W. Lin, T.P. Hong, A new mining approach for uncertain databases using cufp trees, Expert Syst. Appl. 39 (2012) 4084–4093.
[6] C.W. Lin, T.P. Hong, A survey of fuzzy web mining, Wiley Interdisc. Rev.: Data Min. Knowl. Discov. 3 (2013) 190–199.
[7] C.W. Lin, T.P. Hong, W.H. Lu, An effective tree structure for mining high utility itemsets, Expert Syst. Appl. 38 (2011) 7419–7424.
[8] C.W. Lin, T.P. Hong, W.H. Lu, W.Y. Lin, An incremental fusp-tree maintenance algorithm, Int. Conf. Intell. Syst. Des. Appl. 2008 (2008) 445–449.

[9] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: The International Conference on Very Large Data Bases, 1994, pp. 487–499.
[10] W. Song, Y. Liu, J. Li, Mining high utility itemsets by dynamically pruning the tree structure, Appl. Intell. (2013) 1–15.
[11] H. Yao, H.J. Hamilton, Mining itemset utilities from transaction databases, Data Knowl. Eng. 59 (2006) 603–626.
[12] Y. Liu, W.K. Liao, A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, Advan. Knowl. Discov. Data Min. (2005) 689–695.
[13] V.S. Tseng, S. Bai-En, W. Cheng-Wei, P.S. Yu, Efficient algorithms for mining high utility itemsets from transactional databases, IEEE Trans. Knowl. Data Eng. 25 (2013) 1772–1786.
[14] D. W. L. Cheung, J. Han, V. Ng, C. Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating technique, in: International Conference on Data Engineering, 1996, pp. 106–114.
[15] T.P. Hong, C.W. Lin, Y.L. Wu, Incrementally fast updated frequent pattern trees, Expert Syst. Appl. 34 (2008) 2424–2435.
[16] T.P. Hong, C.Y. Wang, Y.H. Tao, A new incremental data mining algorithm using pre-large itemsets, Intell. Data Anal. 5 (2001) 111–129.
[17] C.W. Lin, T.P. Hong, W.H. Lu, The pre-fufp algorithm for incremental mining, Expert Syst. Appl. 36 (2009) 9498–9505.
[18] C.W. Lin, G.C. Lan, T.P. Hong, An incremental mining algorithm for high utility itemsets, Expert Syst. Appl. 39 (2012) 7173–7180.
[19] D.W.L. Cheung, S.D. Lee, B. Kao, A general incremental technique for maintaining discovered association rules, in: The International Conference on Database Systems for Advanced Applications, 1997, pp. 185–194.
[20] T.P. Hong, C.W. Lin, Y.L. Wu, Maintenance of fast updated frequent pattern trees for record deletion, Comput. Statist. Data Anal. 53 (2009) 2485–2499.
[21] C.W. Lin, G.C. Lan, T.P. Hong, L. Kong, Mining high utility itemsets based on transaction deletion, Lect. Notes Electr. Eng. 260 (2014) 983–990.
[22] T.P. Hong, C.Y. Wang, Maintenance of association rules using pre-large itemsets, Intell. Datab.: Technol. Appl. (2007) 44–60.
[23] B. Nath, D.K. Bhattacharyya, A. Ghosh, Incremental association rule mining: a survey, WIREs Data Min. Knowl. Discov. 3 (2013).
[24] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: a frequent-pattern tree approach, Data Min. Knowl. Disc. 8 (2004) 53–87.
[25] C.W. Lin, T.P. Hong, W.H. Lu, Maintenance of the pre-large trees for record deletion, in: N. Mastorakis, J. Sakellaris (Eds.), Lecture Notes in Electrical Engineering, vol. 11, Springer, US, 2009, pp. 137–148.
[26] C.W. Lin, T.P. Hong, W.H. Lu, Using the structure of prelarge trees to incrementally mine frequent itemsets, New Gener. Comput. 28 (2010) 5–20.
[27] R. Chan, Q. Yang, Y.D. Shen, Mining high utility itemsets, in: IEEE International Conference on Data Mining, 2003, pp. 19–26.
[28] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, in: ACM International Conference on Information and Knowledge Management, 2012, pp. 55–64.
[29] G.C. Lan, T.P. Hong, Vincent S. Tseng, An efficient projection-based indexing approach for mining high utility itemsets, Knowl. Inform. Syst. 38 (2014) 85–107.
[30] Philippe Fournier-Viger, C.W. Wu, S. Zida, Vincent S. Tseng, FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning, in: The International symposium on Methodologies for Intelligent Systems, 2014, pp. 83–92.
[31] P. Asha, T. Jebarajan, G. Saranya, A survey on efficient incremental algorithm for mining high utility itemsets in distributed and dynamic database, Int. J. Emerg. Technol. Advan. Eng. 4 (2014) 146–149.
[32] C.W. Wu, B.E. Shie, Philip S. Yu, Vincent S. Tseng, Mining top-k high utility itemsets, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 78–86.
[33] X. Yan, C. Zhang, S. Zhang, Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support, Expert Syst. Appl. 36 (2009) 3066–3076.
[34] H.R. Qodmanan, M. Nasiri, B. Minaei-Bidgoli, Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence, Expert Syst. Appl. 38 (2011) 288–298.
[35] R. Agrawal, R. Srikant, Quest Synthetic Data Generator, 1994 <http://www.Almaden.ibm.com/cs/quest/syndata.html>.
[36] Microsoft, Example Database Foodmart of Microsoft Analysis Services <http://msdn.microsoft.com/en-us/library/aa217032(SQL.80).aspx>.

Please cite this article in press as: C.-W. Lin et al., Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Informat. (2014), http://dx.doi.org/10.1016/j.aei.2014.08.003