Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets

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

Abstract—*Mining high utility itemsets* (*HUIs*) from databases is an important data mining task, which refers to the discovery of itemsets with high utilities (e.g. high profits). However, it may present too many HUIs to users, which also degrades the efficiency of the mining process. To achieve high efficiency for the mining task and provide a concise mining result to users, we propose a novel framework in this paper for mining *closed*$^+$ *high utility itemsets* (*CHUIs*), which serves as a compact and lossless representation of HUIs. We propose three efficient algorithms named *AprioriCH* (*Apriori-based algorithm for mining High utility Closed*$^+$ *itemsets*), *AprioriHC-D* (*AprioriHC algorithm with Discarding unpromising and isolated items*) and *CHUD* (*Closed*$^+$ *High Utility Itemset Discovery*) to find this representation. Further, a method called *DAHU* (*Derive All High Utility Itemsets*) is proposed to recover all HUIs from the set of CHUIs without accessing the original database. Results on real and synthetic datasets show that the proposed algorithms are very efficient and that our approaches achieve a massive reduction in the number of HUIs. In addition, when all HUIs can be recovered by DAHU, the combination of CHUD and DAHU outperforms the state-of-the-art algorithms for mining HUIs.

Index Terms—Frequent itemset, closed$^+$ high utility itemset, lossless and concise representation, utility mining, data mining

1 Introduction

Frequent itemset mining (*FIM*) [1], [3], [4], [5], [8], [10], [18], [20], [20], [27], [31], [32] is a fundamental research topic in data mining. One of its popular applications is *market basket analysis*, which refers to the discovery of sets of items (itemsets) that are frequently purchased together by customers. However, in this application, the traditional model of FIM may discover a large amount of frequent but low revenue itemsets and lose the information on valuable itemsets having low selling frequencies. These problems are caused by the facts that (1) FIM treats all items as having the same importance/unit profit/weight and (2) it assumes that every item in a transaction appears in a binary form, i.e., an item can be either present or absent in a transaction, which does not indicate its purchase quantity in the transaction. Hence, FIM cannot satisfy the requirement of users who desire to discover itemsets with high utilities such as high profits.

To address these issues, *utility mining* [2], [6], [7], [12], [13], [14], [16], [17], [19], [22], [23], [24], [25], [26], [30] emerges as an important topic in data mining. In utility mining, each item has a weight (e.g. unit profit) and can appear more than once in each transaction (e.g. purchase quantity). The utility of an itemset represents its importance, which can be measured in terms of weight, profit, cost, quantity or other information depending on the user preference. An itemset is called a *high utility itemset* (HUI) if its utility is no less than a user-specified *minimum utility threshold*; otherwise, it is called a *low utility itemset*. Utility mining is an important task and has a wide range of applications such as *website click stream analysis* [2], [12], *cross-marketing in retail stores* [24], [26], *mobile commerce environment* [22] and *biomedical applications* [6].

However, HUI mining is not an easy task since the *downward closure property* [1], [10], [21] in FIM does not hold in utility mining. In other words, the search space for mining HUIs cannot be directly reduced as it is done in FIM because a superset of a low utility itemset can be a high utility itemset. Many studies [2], [6], [13], [14], [17], [19], [25], [26], [30] were proposed for mining HUIs, but they often present a large number of high utility itemsets to users. A very large number of high utility itemsets makes it difficult for the users to comprehend the results. It may also cause the algorithms to become inefficient in terms of time and memory requirement, or even run out of memory. It is widely recognized that the more high utility itemsets the algorithms generate, the more processing they consume. The performance of the mining task decreases greatly for low minimum utility thresholds or when dealing with dense databases [8], [18], [20], [31], [32].

In FIM, to reduce the computational cost of the mining task and present fewer but more important patterns to users, many studies focused on developing concise representations, such as *free sets* [3], *non-derivable sets* [4], *odds ratio patterns* [15], *disjunctive closed itemsets* [11], *maximal*

- *V. S. Tseng and C.-W. Wu are with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan City, Taiwan, ROC. E-mail: tsengsm@mail.ncku.edu.tw, silvemoonfox@hotmail.com.*
- *P. Fournier-Viger is with the Department of Computer Science, University of Moncton, Moncton, Canada. E-mail: philippe.fournierer@umoncton.ca.*
- *P.S. Yu is with the Department of Computer Science, University of Illinois at Chicago IL 60607. E- mail: psyu@cs.uic.edu.*

Manuscript received 8 Jan. 2014; revised 11 July 2014; accepted 11 July 2014. Date of publication 4 Aug. 2014; date of current version 28 Jan. 2015. Recommended for acceptance by H. Xiong.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2345377

itemsets [8] and *closed itemsets* [20]. These representations successfully reduce the number of itemsets found, but they are developed for FIM instead of HUI mining. Therefore, an important research question is "*Is it possible to conceive a compact and lossless representation of high utility itemsets inspired by these representations to address the aforementioned issues in HUI mining?*"

Answering this question positively is not easy. Developing a concise and complete representation of HUIs poses several challenges:

1) Integrating concepts of concise representations from FIM into HUI mining may produce a lossy representation of all HUIs or a representation that is not meaningful to the users.
2) The representation may not achieve a significant reduction in the number of extracted patterns to justify using the representation.
3) Algorithms for extracting the representation may not be efficient. They may be slower than the best algorithms for mining all HUIs.
4) It may be hard to develop an efficient method for recovering all HUIs from the representation.

In this paper, we address all of these challenges by proposing a condensed and meaningful representation of HUIs named *closed*$^+$ *high utility itemsets* (CHUIs), which integrates the concept of closed itemset into high utility itemset mining. Our contributions are four-fold and correspond to resolving the previous four challenges:

1) The proposed representation is *lossless* due to a new structure named *utility unit array* that allows recovering all HUIs and their utilities efficiently.
2) The proposed representation is also *compact*. Experiments show that it reduces the number of itemsets by several orders of magnitude, especially for datasets containing long high utility itemsets (up to 800 times).
3) We propose three efficient algorithms named *AprioriHC* (*Apriori-based algorithm for mining High utility Closed*$^+$ *itemset*), *AprioriHC-D* (*AprioriHC algorithm with Discarding unpromising and isolated items*) and *CHUD* (*Closed*$^+$ *High Utility itemset Discovery*) to find this representation. The AprioriHC and AprioriHC-D algorithms employs breadth-first search to find CHUIs and inherits some nice properties from the well-known Apriori algorithm [1]. The CHUD algorithm includes three novel strategies named *REG*, *RML* and *DCM* that greatly enhance its performance. Results show that CHUD is much faster than the state-of-the-art algorithms [2], [17], [24] for mining all HUIs.
4) We propose a top-down method named *DAHU* (*Derive All High Utility itemsets*) for efficiently recovering all HUIs from the set of CHUIs. The combination of CHUD and DAHU provides a new way to obtain all HUIs and outperforms UP-Growth [24], one of the currently best methods for mining HUIs.

The remainder of this paper is organized as follows. In Section 2, we introduce the background for compact representations and utility mining. Section 3 defines the representation of closed$^+$ high utility itemsets and presents our methods. Experiments and discussion are shown in Section 4 and Section 5. Conclusion and future works are given in Sections 6 and 7.

2 BACKGROUND

In this section, we introduce the preliminaries associated with high utility itemset mining, closed itemset mining and compact representations of high utility itemsets.

2.1 High Utility Itemset Mining

Let $I = \{a_1, a_2, \ldots, a_M\}$ be a finite set of distinct items. A *transactional database* $D = \{T_1, T_2, \ldots, T_N\}$ is a set of transactions, where each transaction $T_R \in D (1 \leq R \leq N)$ is a subset of I and has an unique identifier *R*, called *Tid*. Each item $a_i \in I$ is associated with a positive real number $p(a_i, D)$, called its *external utility*. Every item a_i in the transaction T_R has a real number $q(a_i, T_R)$, called its *internal utility*. An *itemset* $X = \{a_1, a_2, \ldots, a_K\}$ is a set of K distinct items, where $a_i \in I$, $1 \leq i \leq K$, and K is called *the length of X*. A *K-itemset* is an itemset of length K. An itemset X is said to be contained in a transaction T_R if $X \subseteq T_R$.

Definition 1 (Support of an itemset). *The support count of an itemset X is defined as the number of transactions containing X in D and denoted as SC(X). The support of X is defined as the ratio of SC(X) to* $|D|$. *The complete set of all the itemsets in D is defined as* $L = \{X \mid X \subseteq I, SC(X) > 0\}$.

Definition 2 (Absolute utility of an item in a transaction). *The absolute utility of an item* a_i *in a transaction* T_R *is denoted as* $au(a_i, T_R)$ *and defined as* $p(a_i, D) \times q(a_i, T_R)$.

Definition 3 (Absolute utility of an itemset in a transaction). *The absolute utility of an itemset X in a transaction* T_R *is defined as* $au(X, T_R) = \sum_{a_i \in X} au(a_i, T_R)$.

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

Definition 5 (Absolute utility of an itemset in a database). *The absolute utility of an itemset X in D is defined as* $au(X) = \sum_{X \subseteq T_R \wedge T_R \in D} u(X, T_R)$. *The (relative) utility of X is defined as* $u(X) = au(X)/TotalU$.

Definition 6 (High utility itemset). *An itemset X is called high utility itemset iff u(X) is no less than a user-specified minimum utility threshold min_utility* ($0\% < min\_util \leq 100\%$). *Otherwise, X is a low utility itemset. An equivalent definition is that X is high utility iff* $au(X) \geq abs\_min\_util$, *where abs_min_util is defined as* $min\_util \times TotalU$.

Definition 7 (Complete set of HUIs in the database). *Let S be a set of itemsets and a function* $f_H(S) = \{X \mid X \in S, u(X) \geq min\_utility\}$. *The complete set of HUIs in D is denoted as* $H (H \subseteq L)$ *and defined as* $f_H(L)$. *The problem of mining HUIs is to find the set H in D.*

Example 1 (High Utility Itemsets). Let Table 1 be a database containing five transactions. Each row in Table 1 represents a transaction, in which each letter represents an item and has a purchase quantity (internal utility).

TABLE 1
An Example Database

TID	Transaction	Transaction Utility (TU)
T_1	A(1), B(1), E(1), W(1)	5
T_2	A(1), B(1), E(3)	8
T_3	A(1), B(1), F(2)	8
T_4	E(2), G(1)	5
T_5	A(1), B(1), F(3)	11

The unit profit of each item is shown in Table 2 (external utility). In Table 1, the absolute utility of the item {F} in the transaction T_3 is $au(\{\mathrm{F}\}, T_3) = p(\{\mathrm{F}\}, D) \times q(\{\mathrm{F}\}, T_3) = 3 \times 2 = 6$. The absolute utility of {BF} in T_3 is $au(\{\mathrm{BF}\}, T_3) = au(\{\mathrm{B}\}, T_3) + au(\{\mathrm{F}\}, T_3) = 1 + 6 = 7$. The absolute utility of {BF} is $au(\{\mathrm{BF}\}) = u(\{\mathrm{BF}\}, T_3) + u(\{\mathrm{BF}\}, T_5) = 17$. If $abs\_min\_utility = 10$, the set of HUIs in Table 1 is $H = \{\{\mathrm{E}\}: 12, \{F\}: 15, \{\mathrm{AE}\}: 10, \{\mathrm{AF}\}: 7, \{\mathrm{BE}\}: 10, \{\mathrm{BF}\}: 17, \{\mathrm{ABE}\}: 12, \{\mathrm{ABF}\}: 19\}$, where the number beside each itemset is its absolute utility.

Note that the utility constraint is neither *monotone* nor *anti-monotone* [1], [10], [21]. In other words, a superset of a low utility itemset can be high utility and a subset of a HUI can be low utility. Hence, we cannot directly use the *anti-monotone* property (also known as *downward closure property*) to prune the search space. To facilitate the mining task, Liu et al. introduced the concept of *transaction-weighted downward closure (TWDC)* [17], which is based on the following definitions.

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

Definition 9 (HTWUI). *An itemset X is a high transaction-weighted utilization itemset (HTWUI) iff* $TWU(X) \geq abs\_min\_utility$. *An HTWUI of length K is abbreviated as K-HTWUI.*

Property 1 (TWDC Property). *The transaction-weighted downward closure property states that for any itemset X that is not a HTWUI, all its supersets are low utility itemsets [2], [17], [19], [24].*

Example 2 (TWDC Property). The transaction utilities of T_1 and T_3 are $TU(T_1) = au(\{\mathrm{ABE}\}, T_1) = 5$ and $TU(T_3) = 8$. If $abs\_min\_utility = 10$, {AB} is a HTWUI since $TWU(\{\mathrm{AB}\}) = TU(T_1) + TU(T_3) = 13$ is no less than *abs_min_utility*. In contrast, the itemset {W} is not a HTWUI, and therefore all the supersets of {W} are low utility (Property 1).

Many studies have been proposed for mining HUIs, including *Two-Phase* [17], *IHUP* [2], *TWU-Mining* [25], *IIDS* [19] and *HUP-Tree* [13], *PB* [14], *UP-Growth* [24]. The former three algorithms use TWDC property to find HUIs. They consist of two phases. In Phase I, they find all HTWUIs from the database. In Phase II, HUIs are identified from the set of HTWUIs by calculating the exact utilities of HTWUIs. Although these methods capture the complete set of HUIs, they may generate too many candidates in Phase I, i.e., HTWUIs, which degrades the performance of Phase II and the overall performance. To reduce the number of candidates in Phase I, various methods have been proposed (e.g.

TABLE 2
Profit Table

Item	A	B	E	F	G	W
Unit Profit	1	1	2	3	1	1

[2], [13], [14], [19]). Recently, Tseng et al. proposed UP-Growth [24] with four strategies *DGU*, *DGN*, *DLU* and *DLN*, for mining HUIs. Experiments in [24] show that the number of candidates generated by UP-Growth in Phase I can be order of magnitudes smaller than that of HTWUIs.

Though the above methods perform well in some cases, their performance degrades quickly when there are many HUIs in the databases. A large number of HUIs and candidates cause these methods to suffer from long execution time and huge memory consumption. When the system resources are limited (e.g. the memory space and processing power), it is often impractical to generate the entire set of HUIs. Besides, a large amount of HUIs is hard to be comprehended or analyzed by users. In FIM, to reduce the number of patterns, many studies were conducted to develop compact representations of frequent itemsets (FIs) that eliminate redundancy, such as *free sets* [3], *non-derivable sets* [4], *odds ratio patterns* [15], *disjunctive closed itemsets* [11], *maximal itemsets* [8] and *closed itemsets* [20]. Although these representations achieve a significant reduction in the number of extracted frequent itemsets, some of them lead to loss of information (e.g. [8]). To provide not only compact but also complete information about frequent itemsets to users, many studies were conducted on closed itemset mining.

2.2 Closed Itemset Mining

In this section, we introduce definitions and properties related to closed itemsets and mention relevant methods. For more details about closed itemsets, readers can refer to [5], [9], [11], [18], [20], [27], [31].

Definition 10 (Tidset of an itemset). *The Tidset of an itemset X is denoted as g(X) and defined as the set of Tids of transactions containing X. The support count of X is expressed in terms of g(X) as* $SC(X) = |g(X)|$.

Property 2. *For itemsets* $X, Y \in L, SC(X \cup Y) = |g(X) \cap g(Y)|$.

Definition 11 (Closure of an itemset). *The closure of an itemset* $X \in L$, *denoted as C(X), is the largest set* $Y \in L$ *such that* $X \subseteq Y$ *and* $SC(X) = SC(Y)$. *Alternatively, it is defined as* $C(X) = \bigcap_{R \in g(X)} T_R$.

Property 3. $\forall X \in L, SC(X) = SC(C(X)) \Leftrightarrow g(X) = g(C(X))$.

Definition 12 (Closed itemset). *An itemset* $X \in L$ *is a closed itemset iff there exists no itemset* $Y \in L$ *such that (1)* $X \subset Y$ *and (2)* $SC(X) = SC(Y)$. *Otherwise, X is non-closed itemset. An equivalent definition is that X is closed iff* $C(X) = X$.

For example, in the database of Table 1, {B} is *non-closed* because $C(\{\mathrm{B}\}) = T_1 \cap T_2 \cap T_3 \cap T_5 = \{\mathrm{AB}\}$.

Definition 13 (Complete set of closed itemset in the database). *Let S be a set of itemsets and a function* $f_C(S) = \{X \mid X \in S, \neg \exists Y \in S$ *such that* $X \subset Y$ *and* $SC(X) = SC(Y)\}$. *The complete set of closed itemsets in D is denoted as* $C (C \subseteq L)$ *and defined as* $f_C(L)$.

Property 4 (Recovery of Support). $\forall X \in L, SC(X) = max\{SC(Y) \,|\, Y \in f_C(L) \wedge X \subseteq Y\}$.

For example, the set of closed itemsets in Table 1 is $f_C(L) = \{\{\text{E}\}:3, \{\text{EG}\}:1, \{\text{AB}\}:4, \{\text{ABE}\}:2, \{\text{ABF}\}:2, \{\text{ABEW}\}:1\}$, in which the number beside each itemset is its support count. The supersets of {B} in $f_C(L)$ are $\{\text{AB}\}:4$, $\{\text{ABE}\}:2$, {ABF}:2 and $\{\text{ABEW}\}:1$. Thus, $SC(\{\text{B}\}) = max\{4, 2, 2, 1\} = 4$.

Mining *frequent closed itemset* (FCI) refers to the discovery of all the closed itemsets having a support no less than a user-specified threshold. It is widely recognized that the number of FCIs can be much smaller than the set of frequent itemsets for real-life databases and that mining FCIs can also be much faster and memory efficient than mining FIs [5], [18], [20], [27], [31]. The set of closed itemsets is lossless since all FIs and their supports can be easily derived from it by Property 4 without scanning the original database [20]. Many efficient methods were proposed for mining FCIs, such as *A-Close* [20], *CLOSET+* [27], *CHARM* [31] and *DCI-Closed* [18]. However, these methods do not consider the utility of itemsets. Therefore, they may present lots of closed itemsets with low utilities to users and omit several high utility itemsets.

2.3 Compact Representations of High Utility Itemset Mining

To present representative HUIs to users, some concise representations of HUIs were proposed. Chan et al. introduced the concept of utility frequent closed patterns [6]. However, it is based on a definition of high utility itemset that is different from [2], [7], [12], [13], [14], [16], [17], [24], [25], [26] and our work. Shie et al. proposed a compact representation of HUIs, called *maximal high utility itemset* and the *GUIDE* algorithm for mining it [23]. A HUI is said to be *maximal* if it is not a subset of any other HUI. For example, if *abs_min_utility* = 10, the set of maximal HUIs in Table 1 is {{ABE}, {ABF}}. Although this representation reduces the number of extracted HUIs, it is not lossless. The reason is that the utilities of the subsets of a maximal HUI cannot be known without scanning the database. Besides, recovering all HUIs from maximal HUIs can be very inefficient because many subsets of a maximal HUI can be low utility. Another problem is that the GUIDE algorithm cannot capture the complete set of maximal HUIs.

3 CLOSED+ HIGH UTILITY ITEMSET MINING

In this section, we incorporate the concept of closed itemset with high utility itemset mining to develop a representation named *closed*$^+$ *high utility itemset*. We theoretically prove that this new representation is *meaningful*, *lossless* and *concise* (i.e., *not larger than the set of all HUIs*).

3.1 Push Closed Property into High Utility Itemset Mining

The first point that we should discuss is how to incorporate the closed constraint into high utility itemset mining. There are several possibilities. First, we can define the closure on the utility of itemsets. In this case, a high utility itemset is said to be closed if it has no proper superset having the same utility. However, this definition is unlikely to achieve a high reduction of the number of extracted itemsets since not many itemsets have exactly the same utility as their supersets in real datasets. For example, there are seven HUIs in Example 1 and only one itemset {E} is non-closed, since $\{\text{E}\} \subseteq \{\text{ABE}\}$ and $au(\{\text{E}\}) = au(\{\text{ABE}\}) = 12$. A second possibility is to define the closure on the supports of itemsets. In this case, there are two possible definitions depending on the join order between the closed constraint and the utility constraint:

— Mine all the high utility itemsets first and then apply the closed constraint. We formally define this set as $H' = f_C(f_H(L))$. It follows that $H' \subseteq H$.
— Mine all the closed itemsets first and then apply the utility constraint. We formally define this set as $C' = f_H(f_C(L))$. It follows that $C' \subseteq C$.

As indicated in [29], the join order between two constraints often lead to different results. Therefore, our next step is to analyze the result sets defined based on the above two join orders. We show that they produce the same result set by the following lemmas.

Lemma 1. $H' \subseteq C'$.

Proof. We prove that $H' \subseteq C'$ by proving that $\forall X \in H' \Rightarrow X \in C'$. Since $X \in H$, $X \in H$ and $u(X) \geq min\_utility$. Then, we prove that $\neg\, \exists Y \in H$ such that $X \subset Y$ and $SC(X) = SC(Y)$ yields $X \in C$ by showing that $X \notin C$ contradicts $Y \in H$. If $X \notin C$, there must exists an itemset $Y \in L$ such that $X \subset Y$ and $SC(X) = SC(Y)$. By Definition 5, $u(Y) > u(X) \geq min\_utility$, and therefore $Y \in H$, which is a contradiction. □

Lemma 2. $C' \subseteq H$.

Proof. We prove that $C' \subseteq H$ by proving that $\forall X \in C' \Rightarrow X \in H$. Since $X \in C$ and $u(X) \geq min\_utility$, we have $X \in H$. Then, we prove that $X \in C$ yields $\neg\, \exists Y \in H$ such that $X \subset Y$ and $SC(X) = SC(Y)$ by showing that $\exists Y \in H$ contradicts $X \notin C$. If $Y \in H$, then $Y \in L$. Because $X \subset Y$, $Y \in L$ and $SC(X) = SC(Y)$, it follows that $X \notin C$. □

Theorem 1. $H' = C$.

Proof. This directly follows from Lemmas 1 and 2. Because the two join orders produce the same result, the join order can be removed to obtain a general definition. □

Definition 14 (Closed high utility itemset). *We define the set of closed high utility itemsets as* $HC = \{X \,|\, X \in L, X = C(X), u(X) \geq min\_utility\}$, $HC = H' = C$. *An itemset X is called non-closed high utility itemset iff* $X \in H$ *and* $X \notin C$.

For example, if $abs\_min\_utility = 10$, the complete set of closed HUIs in Table 1 is HC = {{E}, {ABE}, {ABF}}.

Definition 14 gives an alternative solution to incorporate the closed constraint with high utility itemset mining. The advantage of using this definition is that the two constraints can be applied in any order during the mining process. We say that the representation *HC* is *concise* because its size is guaranteed to be no larger than the set of all HUIs (because $HC \subseteq H$).We next show that this representation is *meaningful*.

Property 5. *For any non-closed high utility itemset X,* $\exists Y \in HC$ *such that* $Y = C(X)$ *and* $u(Y) > u(X)$.

Proof. $\forall X \in L$, $\exists Y \in C$ such that $Y = C(X)$ and $SC(X) = SC(Y)$. Since $X \in H$ and $X \notin C$, $u(X) \geq min\_utility$ and $X \subset Y$. $SC(X) = SC(Y)$ and $X \subset Y$ yields $u(Y) > u(X) \geq min\_utility$ by Property 3 and Definition 5. □

We claim that *HC* is a meaningful representation of all HUIs by Property 5. For any non-closed high utility itemset *X*, *X* does not appear in a transaction without its closure *Y*. Moreover, the utility (e.g. profit/user preference) of *Y* is guaranteed to be higher than the utility of *X*. For these reasons, users are more interested in finding *Y* than *X*. Moreover, closed itemsets having high utilities are useful in many applications. For example, in market basket analysis, *Y* is the closure of *X* means that no customer purchases *X* without its closure *Y*. Thus, when a customer purchases *X*, the retailer can recommend $Y - X$ to the customer, to maximize profit.

Although *HC* is based on the concise representation of closed itemsets, the set of closed HUIs is *not lossless*. If an itemset is not included in this representation, there is no way to infer its utility and to know whether it is high utility or not. To tackle this problem, we attach to each closed HUI a special structure named *utility unit array*, which is defined as follows.

Definition 15 (Utility unit array). $\forall X = \{a_1, a_2, \ldots, a_K\} \in L$, *the utility unit array of X is denoted as* $\mathrm{V}(X) = [v_1, v_2, \ldots, v_K]$ *and contains K utility values. The ith utility value* v_i *in* $V(X)$ *is denoted as* $\mathrm{V}(X, a_i)$ *and defined as* $\sum_{R \in g(X) \wedge a_i \in T_R} au(a_i, T_R)$.

Example 3 (Utility unit array). Consider the database in Table 1 and the itemset {ABE} appearing in T_1 and T_2. The first utility value in *V*({ABE}) is *V*({ABE}, {A}) = $au(\{\mathrm{A}\}, T_1) + au(\{\mathrm{A}\}, T_2) = 2$. The utility unit array of {ABE} is $V(\{\mathrm{ABE}\}) = [2, 2, 8]$.

Property 6. $\forall X = \{a_1, a_2, \ldots, a_K\} \in L$, $au(X) = \sum_{i=1}^{K} V(X, a_i)$.

Proof. The utility of *X* is the sum of the utilities of items $a_1, a_2, \ldots, a_K$ in transactions containing *X*. For an item a_i, $V(X, a_i)$ represents the sum of the absolute utilities of a_i in transactions containing *X*. Therefore *au*(*X*) can be expressed as $V(X, a_1) + V(X, a_2) + \cdots + V(X, a_K)$. □

For example, $au(\{\mathrm{ABE}\}) = V(\{\mathrm{ABE}\}, \{\mathrm{A}\}) + V(\{\mathrm{ABE}\}, \{\mathrm{B}\}) + V(\{\mathrm{ABE}\}, \{\mathrm{E}\}) = 2 + 2 + 8 = 12$.

Property 7. *$\forall X \in L$, X is low utility if $C(X) \notin HC$.*

Proof. If $C(X) \notin HC$, $u(C(X)) < min\_utility$. Since $SC(X) = SC(C(X))$ and $X \subseteq C(X)$, by Definition 5 we have $u(X) \leq u(C(X)) < min\_utility$. □

Property 8 (Recovery of Utility). $\forall X = \{a_1, a_2, \ldots, a_K\} \in L$, *if* $C(X) \in HC$, the absolute utility of X can be calculated as $au(X) = \sum_{a_i \in X} V(C(X), a_i)$.

Proof. Because $X \subseteq C(X)$, there exists an entry $V(C(X), a_i)$ in $V(C(X))$ for each $a_i \in X$. Besides, $g(X) = g(C(X))$ since $SC(X) = SC(C(X))$ and $X \in C(X)$ (Property 3). Thus, $V(X, a_i) = V(C(X), a_i)$, by Definition 15. According to Property 6, $au(X) = \sum_{i=1}^{K} V(X, a_i)$. By replacing $V(X, a_i)$ with $V(C(X), a_i)$, we obtain Property 8. □

Definition 16 (Closed$^+$ high utility itemset). *An itemset X is called closed$^+$ high utility itemset (CHUI) iff* $X \in HC$ *and X is annotated with V(X). The set of CHUIs is a lossless representation of all HUIs. For any itemset* $\mathrm{Y} \in \mathrm{H}$, *its absolute utility can be inferred from the utility unit array of its closure by Property 8 without scanning the original database.*

3.2 Efficient Algorithms for Mining Closed$^+$ High Utility Itemsets

In this section, we introduce three efficient algorithms *AprioriHC (An Apriori-based algorithm for mining High utility Closed$^+$ itemsets)*, *AprioriHC-D (AprioriHC algorithm with Discarding unpromising and isolated items*) and *CHUD (Closed$^+$ High Utility itemset Discovery) for mining CHUIs*. They rely on the *TWU-Model* [2], [13], [14], [17], [19], [24] and include strategies to improve their performance. All algorithms consist of two phases named *Phase I* and *Phase II*. In *Phase I*, *potential closed$^+$ high utility itemsets* (PCHUIs) are found, which are defined as a set of itemsets having an estimated utility (e.g. TWU) no less than *abs_min_utility*. In Phase II, by scanning the database once, CHUIs are identified from the set of PCHUIs found in Phase I and their utility unit arrays are computed.

The AprioriHC and AprioriHC-D are based on Apriori [1] and the Two-Phase [17] algorithms. They use a horizontal database and explore the search space of CHUIs in a breadth-first search. The algorithm AprioriHC is regarded as a baseline algorithm in this work and AprioriHC-D is an improved version of AprioriHC. On the other hand, the proposed algorithm CHUD is an extension of Eclat [31] and DCI-Closed [18] algorithms. The CHUD algorithm considers vertical database and mines CHUIs in a depth-first search. In the following, we present details of the three algorithms.

3.2.1 The AprioriHC Algorithm

Initially, a variable *k* is set to 1. The algorithm performs a database scan to compute the transaction utility of each transaction (Definition 4). At the same time, the TWU of each item is computed. Each item having a TWU no less than *abs_min_utility* is added to the set of 1-HTWUIs C_k. Then the algorithm proceeds recursively to generate itemsets having a length greater than *k*. During the *k*th iteration, the set of *k*-HTWUIs L_k is used to generate $(k+1)$-candidates C_{k+1} by using the *Apriori-gen function* [1]. Then the algorithm computes TWUs of itemsets in C_{k+1} by scanning the database *D* once.

Each itemset having a TWU no less than *abs_min_utility* is added to the set of $(k+1)$-HTWUIs L_{k+1}. After that, the algorithm removes non-closed itemsets in L_{k+1} by the following process. For each candidate *X* in L_{k+1}, the algorithm checks if there exists a subset $Y \subseteq X$ such that $Y \in L_k$ and $SC(X) = SC(Y)$. If true, *Y* is deleted from L_k because *Y* is not a closed$^+$ high utility itemset according to Definition 14. If false, *Y* is kept and marked as "closed" because it may be a closed$^+$ high utility itemset. The phase I of AprioriHC terminates when no candidate is generated. Then, the algorithm performs Phase II. In phase II, the algorithm scans the database once and calculates the utilities of HTWUIs that are marked as "closed" to identify the set of closed$^+$ high utility itemsets.

3.2.2 The AprioriHC-D Algorithm

The AprioriHC-D algorithm is an improvement of AprioriHC. It includes two effective strategies to reduce the

ALGORITHM: AprioriHC-D

```
Input: D: the database; abs_min_utility;
       pCHUI: the set of PCHUIs pCHUI
Output: The complete set of CHUIs
01. pCHUI := ∅
02. L1 := 1-HTWUIs in D
03. D1:= DGU_Strategy(D, L1)
04. L1:= 1-HTWUIs in D1
05. AprioriHC-D_Phase-I (D1, pCHUI, abs_min_utility, L1)
06. AprioriHC-D_Phase-II(D1, pCHUI, abs_min_utility)
```

Fig. 1. AprioriHC-D algorithm.

PROCEDURE: AprioriHC-D_Phase-I

```
Input: Dk: the trimmed database; abs_min_utility;
       pCHUI: the set of PCHUIs
Output: the complete set of PCHUIs
Lk+1 := Lk
for (k = 1; Lk+1 ≠∅; k++)
{  Ck+1 := Apriori-gen(Lk)
   Lk+1 := EstimatedUtilityOf_Itemsets(Dk, Ck+1)
   Lk   := MarkClosed_Itemsets(Lk, Lk+1)
   pCHUI := pCHUI ∪ Lk-1
   Dk+1 := IIDS_Strategy(Dk, Lk+1)
}
```

Fig. 2. AprioriHC-D_Phase-I procedure.

number of PCHUIs generated in Phase I that are inspired by the UP-Growth [24] and IIDS algorithms [19]. The first strategy is based on the following definition and properties.

Definition 17 (Promising item). *An item i_p is a promising item iff $TWU(i_p) \geq abs\_min\_utility$. Otherwise, it is an unpromising item.*

Property 9. *Any unpromising item i_u and its supersets are not high utility itemsets.*

Proof. By the transaction-weighted downward closure property (Property 1), an item i_u and its supersets are not high utility itemsets iff $TWU(i_u) < abs\_min\_utility$. Because every CHUI has to be a HUI, the property holds. □

We adapt Property 9 to the context of closed$^+$ high utility itemset mining as follows.

Property 10. *Any unpromising item i_u and its supersets are not closed$^+$ high utility itemsets.*

Proof. For any itemset *X*, if *TWU*(*X*) is less than *abs_min_utility*, it is a low utility itemset. By Definition 14, a low utility itemset is not a *closed*$^+$ *high utility itemset*. □

Strategy 1. DGU (Discarding Global Unpromising items) [24]. *Discard global unpromising items and their exact utilities from transactions and transaction utilities of the database, respectively.*

Rationale. By Property 10, unpromising items play no role in CHUIs. Therefore, unpromising items can be removed from each transaction T_R and their absolute utilities can be subtracted from $TU(T_R)$. Thus, the utilities of unpromising items can be ignored in the calculation of the estimated utilities of itemsets (i.e., *TWU*). For more details about the strategy DGU, readers can refer to [24].

The second strategy is based on the following definition and properties.

Definition 18 (Isolated item). *Let L_k be the set of HTWUIs of length k, an item i_o is called an isolated item of level k iff i_o is not contained in any itemset in L_k.*

Property 11. *For any isolated item i_o of level k, its supersets of length l ($l \geq$ k) are not high utility itemsets.*

Proof. The reader is referred to [19] for the proof.

We adapt Property 11 to the context of closed$^+$ high utility itemset mining as follows.

Property 12. *For any isolated item i_o of level k, its supersets of length l ($l \geq$ k) are not CHUIs.*

Proof. For any isolated item i_o of level *k*, its supersets of level $l(l \geq k)$ are not HUIs (Property 11). Thus, the supersets of $l(l \geq k)$ are not CHUIs.

Strategy 2. IIDS (Isolated Items Discarding Strategy) [19] : *Discard isolated items and their actual utilities from transactions and transaction utilities of the database.*

Rationale. By Property 12, isolated items of level *k* play no role in CHUIs of length $l(l \geq k)$. Thus, when utilities of *k*-itemsets are being estimated, utilities of isolated items can be regarded as irrelevant, and therefore can be discarded. Therefore, isolated items of level *k* can be removed from each transaction T_R and their absolute utilities can be subtracted from the transaction utility $TU(T_R)$ during the *k*th database scan in the AprioriHC algorithm. Thus, the utilities of isolated items can be ignored in the calculation of the estimated utilities of itemsets (i.e., *TWU*). For more details about the IIDS strategy, readers are referred to [19].

Based on the DGU and IIDS strategies, we present the AprioriHC-D algorithm. The pseudo code is given in Fig. 1. It takes as parameters a horizontal database *D* and the *abs_min_utility* threshold. Because the algorithm is similar to AprioriHC, we here only describe the differences.

The first difference is the application of strategy DGU in the main procedure (line 3 of Fig. 1). This is done immediately after computing the TWUs of items to generate the set of 1-HTWUIs L_1. An additional database scan is here performed to remove unpromising items and discard their absolute utilities from the transaction utilities of the database *D*. The resulting database is named D_1. The algorithm then recalculates the set of 1-HTWUIs L_1 from D_1. The algorithm calls procedure *AprioriHC-D_Phase*-I to perform Phase I for finding candidates of CHUIs (line 5 of Fig. 1). The set of candidates found in Phase I are collected in the set *pCHUI*. Then, the algorithm calls procedure *AprioriHC-D_Phase*-II to perform Phase II. All the CHUIs are identified from the set *pCHUI* by scanning the database D_1 once (line 6 of Fig. 1).

The second difference is the integration of the IIDS strategy (Fig. 3) in the *AprioriHC-D_Phase*-I procedure (Fig. 2). As previously explained, this latter procedure recursively performs iterations to discover *k*-candidates starting from $k = 2$. Initially, for $k = 2$, the procedure uses the database *D* that it receives as parameter D_k to calculate the estimated utilities of candidates. The modification is that during the *k*th iteration, after removing non-closed HUIs in

PROCEDURE:IIDS_Strategy

Input: D_k: the trimmed database;
Output: the database D_{k+1} containing no isolated items of level k
for each transaction $T_R \in D_k$ ***do***
{ ***for each*** item $x \in T_R$ ***do***
  { ***If*** *x is not in* L_{k+1} ***then***
    { ***remove x from*** T_R and ***subtract*** $au(x, T_R)$ ***from*** $TU(T_R)$}}
}

Fig. 3. IIDS_strategy procedure.

L_{k-1}, isolated items of the k-candidates are identified from the set L_k. Then, the isolated items and their exact utilities are removed from D_k. The locally modified database D_k is then used for the next iteration.

The third difference is the method for calculating the absolute utilities of PCHUIs in Phase II. The procedure for Phase II is shown in Fig. 4. It takes as parameters, the database D_1, the set of PCHUIs *pCHUI* and *abs_min_utility*. The procedure performs k iterations starting from $k = 1$ to the maximum length of PCHUIs in *pCHUI*. During the kth iteration, the procedure considers the set of k-PCHUIs L_k in *pCHUI*. For each PCHUI X in L_k, the absolute utility and utility unit array is calculated by scanning D_k. If the absolute utility of X is no less than *abs_min_utility*, X is outputted as a CHUI. Then, the algorithm applies the IIDS strategy to remove isolated items of level k from D_k.

3.2.3 The CHUD Algorithm

In this section, we present an efficient depth-first search algorithm named *CHUD* (*Closed$^+$ High Utility itemset Discovery*) to discover CHUIs. CHUD is an extension of DCI-Closed [18], one of the currently best methods to mine closed itemsets. CHUD is adapted for mining CHUIs and include several effective strategies for reducing the number of candidates generated in Phase I. Similar to the DCI-Closed algorithm, CHUD adopts an *Itemset-Tidset pair Tree* (*IT-Tree)* [18], [31] to find CHUIs. In an IT-Tree, each node $N(X)$ consists of an itemset X, its *Tidset* $g(X)$, and two ordered sets of items named *PREV-SET*(X) and *POST-SET*(X). The IT-Tree is recursively explored by the CHUD algorithm until all closed itemsets that are HTWUIs are generated. Different from the DCI-Closed algorithm, each node $N(X)$ of the IT-Tree is attached with an estimated utility value *EstU*(X).

A data structure called *transaction utility table* (*TU-Table)* [28] is adopted for storing the transaction utilities of transactions. It is a list of pairs $<R, TU(T_R)>$ where the first value is a TID R and the second value is the transaction utility of T_R. Given a TID R, the value $TU(T_R)$ can be efficiently retrieved from the TU-Table. Given a node $N(X)$ with its Tidset $g(X)$ and a TU-Table *TU*, the estimated utility of the itemset X can be efficiently calculated by the procedure shown in Fig. 5.

PROCEDURE: *AprioriHC-D_Phase*-II

Input: D_1: the database containing no unpromising items;
  pCHUI: set of PCHUIs; *abs_min_utility*
Output: the complete set of CHUIs
for (k := 1; $L_k \neq \emptyset$; k++)
{ L_k := k-itemsets in *pCHUI*
  for all X in L_k ***do***
  { ***Calculate*** $au(X)$ and utility unit array of X ***from*** D_k
    If $au(X) \geq$ *abs_min_utility* ***then*** { ***output*** X }
  }
  D_{k+1} := ***IIDS_Strategy***(D_k, L_k)
}

Fig. 4. AprioriHC-D_Phase-II procedure.

PROCEDURE: CalculateEstUtility

Input: $g(X)$: the Tidset of X; TU: a TU table
Output: *Est*U: the estimated utility of X
EstU := 0;
for each TID $R \in g(X)$ ***do***
{ *EstU* := *EstU* + *TU.get*(R) }
return *EstU*

Fig. 5. CalculateEstUtility procedure.

The main procedure of CHUD is named *Main* and is shown in Fig. 6. It takes as parameter a database D and the *abs_min_utility* threshold. CHUD first scans D once to convert D into a vertical database. At the same time, CHUD computes the transaction utility for each transaction T_R and calculates TWU of items. When a transaction is retrieved, its Tid and transaction utility are loaded into a global TU-Table named *GTU*. As previously defined, an item is called a *promising item* if its estimated utility (e.g. its TWU) is no less than *abs_min_utility*. After the database scan, *promising items* (cf. Definition 17) are collected into an ordered list $O = <a_1, a_2, \dots, a_n>$, sorted according to a fixed order $\prec$ such as increasing order of support. Only promising items are kept in O since supersets of unpromising items are not CHUIs (by Property 10). According to [24], the utilities of unpromising items can be removed from the GTU table. This step is performed at line 2 of the *Main* procedure. Then, CHUD generates candidates in a recursive manner, starting from candidates containing a single promising item and recursively joining items to them to form larger candidates. To do so, CHUD takes advantage of the fact that by using the total order $\prec$, the complete set of itemsets can be divided into n non-overlapping subspaces, where the kth subspace is the set of itemsets containing the item a_k but no item $a_i \prec a_k$ [18]. For each item $a_k \in O$, CHUD creates a node $N(\{a_k\})$ and puts items a_1 to a_{k-1} into *PREV-SET*($\{a_k\}$) and items a_{k+1} to a_n into *POST-SET*($\{a_k\}$). Then CHUD calls the *CHUDPhase*-I procedure for each node $N(\{a_k\})$ to produce all the candidates containing the item a_k but no item $a_i \prec a_k$. After that, the *REG* strategy is applied by calling the *REG_Strategy* sub-function, which will be described later. Finally, the *Main* procedure performs Phase II on these candidates to obtain all CHUIs.

The *CHUDPhase*-I procedure shown in Fig. 7 takes as parameter a node $N(X)$, a TU-Table *TU* and *abs_min_utility*.

ALGORITHM: CHUD

Input: D: the database; *abs_min_utility*
Output: complete set of CHUIs
01. ***InitialDatabaseScan***(D)
02. ***RemoveUtilityUnpromisingItems***(O, *GTU*)
03. ***for each*** item $a_k \in O$ ***do***
04. { Create node $N(\{a_k\})$
05.   ***CHUD_Phase*-I**($N(\{a_k\})$, *GTU*, *abs_min_utility*)
06.   ***REG_Strategy***($g(a_k)$, *GTU*)}
07.   ***CHUD_Phase*-II**(D, *abs_min_utility*)

Fig. 6. CHUD algorithm.

```
PROCEDURE: CHUDPhase-I
Input: N(X): the node of X N(X);
       GTU: the global TU table; abs_min_utility
Output: The complete set of PCHUIs
01. if (SubsumeCheck(N(X), PREV-SET(X)) == false) then
02. {  XC := ComputeClosure(N(X), POST-SET(X))
03.    DCM_Strategy(XC)
04.    Explore(N(XC), TU, abs_min_utility)  }  }
```

Fig. 7. CHUDPhase-I procedure.

```
PROCEDURE: SubsumeCheck
Input: N(X): the node ode of X; PREV-SET(X): the pre-set of X
Output: True: if X is non-closed and subsumed by other itemsets
        False: if X is not subsumed by other itemsets
for each item a∈PREV-SET(X) do
{  if (g(X) ⊆ g(a)) then return True  }
return False
```

Fig. 8. SubsumeCheck procedure.

```
PROCEDURE: ComputeClosureOf_Itemsets
Input: N(X): the node ode of X; POST-SET(X): the post-set of X
Output: XC: The closure of X
XC:= X
for each item a∈POST-SET(X) do
{  if (g(X) ⊆ g(a)) then
   {  POST-SET(X) := POST_SETX / {a}
      XC := XC ∪ {a}  }  }
return XC
```

Fig. 9. ComputeClosureOf_Itemsets procedure.

The procedure first performs *SubsumeCheck* on X as presented in Fig. 8. This check verifies if there exists an item a from *PREV-SET*(X) such that $g(X) \subseteq g(a)$. If there exists such an item, it means that X is included in a closed itemset that has already been found and supersets of X do not need to be explored (see [18] for a complete justification). Otherwise, the next step is to compute the closure $X_C = C(X)$ of X, which is performed by the procedure *ComputeClosureOf_Itemsets*(*N*(*X*), *POST-SET*(*X*)) shown in Fig. 9 [18]. Then the estimated utility of X_C is calculated. If it is no less than *abs_min_utility*, X_C is considered as a candidate for Phase II and it is outputted with its estimated utility value $EstU(X_C)$. Note that CHUD does not maintain any discovered candidate in memory. Instead, when a candidate itemset is found, it is outputted. After this, the DCM strategy is applied by calling the *DM_Strategy* subfunction, which will be described later. Then, a node $N(X_C)$ is created and the procedure *Explore* is called for finding candidates that are supersets of X_C.

The *Explore* procedure is shown in Fig. 10. It takes as parameter a node *N*(*X*), a TU-Table TU_X and *abs_min_ utility*. The *Explore* procedure explores the search space of closed candidates that are superset of X by appending items from *POST-SET*(*X*) to X. We here briefly explain this process. For a proof that this method is a correct way of exploring closed candidates, the reader can consult the paper describing DCI-Closed [18]. For each item a_k of *POST-SET*(*X*), the procedure first removes a_k from *POST-SET*(*X*) to create a node *N*(*Y*) with $Y = X \cup \{a_k\}$. The Tidset of Y is then calculated as $g(Y) = g(X) \cap g(a_k)$ by Property 2. The set *POST-SET*(*Y*) and *PREV-SET*(*Y*) are respectively set to *POST-SET*(*X*) and *PREV-SET*(*X*). Then, the estimated utility of Y is calculated by calling the *CalculateEstUtility* procedure with $g(Y)$ and TU_x. If *EstU*(*Y*) and *EstU*(*X*) are no less than *abs_min_utility*, the procedure *CHUDPhase*-I is recursively called with *N*(*Y*) (to consider the search space of *Y*), TU_X and *abs_min_utility*. Then, a_k is added to *PREV-SET*(*X*). If *EstU*(*Y*) is lower than *abs_min_utility*, the search space of Y is pruned since Y and its supersets are low utility (Property 1). Finally, the *RML* strategy is applied by calling the *RML_Strategy* sub-function.

```
PROCEDURE: Explore
Input: N(X): the node of X; TUX: the TU-Table of X; abs_min_utility;
Output: The set of PCHUIs containing X
for each item ak∈POST-SET(X) do
{  POST-SET(X) := POST-SET(X) /{ak}
   Create a node N(Y), where Y := X∪∪{ak}
   g(Y) := g(X)∩g(ak)
   POST-SET(Y) := POST-SET(X)
   PREV-SET(Y) := PREV-SET(X)
   EstU(Y) := CalculateEstUtility(g(Y), TUX)
   if(EstU(Y), EstU(X) ≥ abs_min_utility) then
   {  CHUDPhase-I (N(Y), TUX, abs_min_utility)
      PREV-SET(X) := PREV-SET(X)∪{ak}
   }
   RML_Strategy(g(Y), TUX)
}
```

Fig. 10. Explore procedure.

```
PROCEDURE: REG_Strategy
Input: g(ak): the Tidset of item ak; GTU: the global TU-Table
Output: GTU: the global TU-Table
for each Tid R∈g(ak) do
{  remove au(ak, TR) from <R, GTU(TR)>  }
```

Fig. 11. REG_strategy procedure.

After recursions of the *Explore* and *CHUDPhase-I* procedures are completed, candidates that have been outputted are processed by Phase II. Phase II consists of taking each candidate X and to calculate its utility and utility unit array. Each candidate that is low utility is discarded. Calculating the absolute utility of a candidate X is performed by doing the summation of $au(X, T_R)$ for each $R \in g(X)$. This is done very efficiently thanks to the vertical representation of the database (only transactions containing X are considered to calculate its utility).

The first strategy that we have incorporated in CHUD is to only consider promising items for generating candidates and to remove the utilities of unpromising items from the GTU table. It is applied in line 2 of the *Main* procedure. The second strategy that we have incorporated in CHUD is to discard each itemset X_C such that $EstU(X_C) \geq abs\_min\_utility$. This strategy is integrated in line 3 of the *CHUDPhase*-I procedure. To enhance the performance of CHUD, we integrate three additional strategies, which have never been used in vertical mining of HUIs. They are described as follows.

Strategy 3. REG (Removing the Exact utilities of items from the Global TU-Table). *Each time that an item* $\mathrm{a_k} \in \mathrm{O}$ *has been processed in the main procedure (Fig. 6), this strategy is applied by calling the REG_Strategy procedure (line 6). The pseudo code of the procedure is given in Fig. 11. The procedure is called with* $g(a_k)$ *and the global utility table GTU. It*

PROCEDURE: RML_Strategy

Input: $g(Y)$: Tidset of itemset Y; TU_X: TU-Table of X
Output: $EstU(X)$: the estimated utility of X

```
for each Tid R∈g(Y) do
{    remove miu(a_k) from <R,TU_X(T_R)>   }
remove miu(a_k) × SC(Y) from EstU(X)
```

Fig. 12. RML_strategy procedure.

removes the utility of a_k from the transaction utility of each transaction containing a_k in the global TU-Table.

Rationale. CHUD explores the search space of patterns by dividing it into non-overlapping subspaces such that each item a_i that has been processed is excluded from the subspace of item $a_j \succ a_i$. Thus, absolute utility of a_i can be removed from the transaction utility of each transaction containing a_j in the global TU-Table.

Definition 17 (The minimum item utility of an item). *The minimum item utility of an item a is denoted as miu(a) and defined as the value $au(a, T_r)$ for which $\exists T_s \in D$ such that $au(a, T_s) < au(a, T_r)$.*

Definition 18 (Local TU Table). *Let N(X) be a node for the itemset X and a be an item in POST-SET(X). The local TU-Table for the node $Y = X \cup \{a\}$ is denoted as TU_Y and is initialized with the entries from TU_X corresponding to transactions from g(Y). The local TU-Table for the root node of the IT-Tree is the same as GTU.*

Strategy 4. RML (Removing the Mius of items from Local TU-Tables). *The strategy consists of using a local TU-Table TU_X for each node N(X) in the IT-Tree. Let N(X) be the current node being processed by Explore and N(Y) be a child node of N(X) that has been created by appending an item a_k from POST-SET(X) to X such that $Y = X \cup \{a_k\}$. The strategy is applied after line 11 of the Explore procedure (Fig. 10) by calling the RML_Strategy sub-function (Fig. 12). The RML_Strategy procedure takes as parameters (1) the transaction utility of N(X) and (2) the set of transactions that contains Y. The procedure first removes $miu(a_k)$ from the transaction utility of each transaction containing a_k in TU_X. The updated local TU-Table TU_X is used for all child nodes of N(X). This process reduces the estimated utility of N(X) and that of its child nodes. Besides, $miu(a_k) \times SC(Y)$ is removed from EstU(X). If the updated EstU(X) is less than abs_min_utility, the algorithm will not process $X \cup \{a_k\}$ for each following item $a_k \in POST\text{-}SET(X)$.*

Rationale. Each item a_i that is processed for a node $N(X)$ will not be considered for any child node $N(Y)$, where $Y = X \cup \{a_j\}$ and $a_j \succ a_i$. Thus, $miu(a_i)(Y)$ and $miu(a_i)$ can be removed from $EstU(X)$ and the transaction utility of each transaction containing a_j from TU_X.

Definition 19 (The maximum item utility of an item). *The maximum item utility of an item a is denoted as mau(a) and defined as the value $au(a, T_r)$ for which $\exists T_s \in D$ such that $au(a, T_s) > au(a, T_r)$.*

Definition 20 (The maximum item utility of an itemset). *The maximum item utility of an itemset $X = \{a_1, a_2, \ldots, a_K\}$ is defined as $MAU(X) = \sum_{i=1}^{K} mau(a_i) \times SC(X)$.*

PROCEDURE: DCM_Strategy

Input: X_C: the closure of $X = \{a_1, a_2, \ldots, a_K\}$;
Output: the PCHUI X_C if X_C is not low utility

```
MAU(X) = ∑_{i=1}^{K} mau(a_i) × SC(X)
if (min{EstU(X_C), MAU(X_C)} ≥ abs_min_utility) output X_C
```

Fig. 13. DCM_strategy procedure.

Lemma 4. *$\forall X$, X is low utility if MAU(X) < abs_min_utility.*

Proof. The absolute utility of an itemset X (i.e., $au(X)$) is the sum of the absolute utility of its items in transactions containing X. $MAU(X)$ is the sum of the maximum item utility of each item multiplied by the number of transactions containing X. Since the *maximum item utility of each item* represents the highest utility that an item can have, $MAU(X)$ is higher or equal to $au(X)$. □

Strategy 5. DCM (Discarding Candidates with a MAU that is less than the minimum utility threshold). *The last strategy is called DCM and is applied in line 3 of the CHUD-Phase-I procedure. A candidate X_C can be discarded from Phase II if its estimated utility $EstU(X_C)$ or $MAU(X_C)$ is less than abs_min_utility. The pseudo code of the strategy is given in Fig. 13. The procedure takes as parameter an itemset X_C. It first computes the maximum utility $MAU(X_C)$ of X_C. Then if $EstU(X_C)$ or $MAU(X_C)$ is no less than abs_min_utility, X_C is output with its estimated utility.*

Rationale. Lemma 4 guarantees that an itemset X is not a CHUI iff $MAU(X) <$ *abs_min_utility*.

3.3 Efficient Recovery of High Utility Itemsets

In this section, we present a top-down method named *DAHU* (*Derive All High Utility itemsets*) for efficiently recovering all the HUIs and their absolute utilities from the complete set of CHUIs.

The pseudo code of DAHU is shown in Fig. 14. It takes as input an absolute minimum utility threshold *abs_min_utility*, a set of CHUIs *HC* and *ML* the maximum length of itemsets in *HC*. DAHU outputs the complete set of high utility itemsets $H = \cup_{i=1}^{k} H_k$ respecting *abs_min_utility*, where H_k denotes the set of HUIs of length *k*. To derive all HUIs, DAHU proceeds as follows. First, the set H_{ML} is initialized to

ALGORITHM: DAHU

Input: ML: the maximum length of itemset in HC; *abs_min_utility*;
$HC = \{HC_1, HC_2, \ldots, HC_{ML}\}$: the complete set of CHUIs;
Output: H: the complete set of HUIs

```
H_ML := HC_ML
for (k := ML – 1; k > 0 ; k – –) do
{  for each k-itemset X = {a_1, a_2,..., a_k}∈HC_k do
   {  if (au(X) < abs_min_utility) then delete X from HC_k
      else add X and its absolute utility au(X) to H.
      {  for each item a_i∈X do
         {  Y := X – {a_i}
            au(Y) := au(X) – V(X, a_i)
            if(au(Y) ≥ abs_min_utility) then
            {  if Y∈HC_{k-1} and SC(X) > SC(Y) then
               {  SC(Y) := SC(X)  }
            else if Y∉HC_{k-1} then
            {  HC_{k-1} := HC_{k-1} ∪Y
               SC(Y) := SC(X)      }}}}}
```

Fig. 14. DAHU algorithm.

TABLE 3
Parameter for Synthetic Datasets

Parameter Descriptions	Default
D: Total number of transactions	200 K
T: Average transaction length	12
N: Number of distinct items	1,000
I: Average size of maximal potentialFIs	10
Q: Maximum number of purchased items in transactions	5

TABLE 4
Characteristics of Datasets

Dataset	N	T	D
Mushroom	119	23	8,124
Foodmart	1,559	4.4	4,141
BMSWebView1	497	2.51	59,601
T12-I10-N1K-Q5-D200K	1,000	12	200 K

HC_{ML}, where the notation HC_k represents the set of k-itemsets in HC. During lines 2 to line 14 in Fig. 14, each set H_k is constructed from $k = (ML - 1)$ to $k = 1$. In each iteration, H_{k-1} is recovered by using HC_k. For each itemset $X = \{a_1, a_2, \ldots, a_k\}$ in HC_k, if the absolute utility of X is no less than *abs_min_utility*, the algorithm outputs the high utility itemset X with its absolute utility and then generates all $(k-1)$-subsets of X. The latter are obtained by removing each item $a_i \in X$ from X one at a time to obtain subsets of the form $Y = X - \{a_i\}$. If Y is not present in H_k or Y is present in H_k with $SC(X) > SC(Y)$, Y is added to H_{k-1}, its support count is set to the support count of X (Property 4), i.e., $SC(Y) = SC(X)$, and the absolute utility of Y is set to the absolute utility of X minus the ith value in $V(X)$, i.e., $au(Y) = au(X) - V(X, a_i)$ (Properties 6-8). Besides, the utility unit array of $V(Y)$ is set to $V(X)$ with the value $V(X, a_i)$ removed (Property 8). This process is repeated until H has been completely recovered.

4 Experimental Evaluation

In this section, we evaluate the performance of the proposed algorithms and compare them with two state-of-the-art algorithms UP-Growth [24] and Two-Phase [17]. Although our methods produce different results from those algorithms, they also consist of two phases. In Phase I, the proposed algorithms generate candidates for CHUIs, whereas UP-Growth and Two-Phase generate candidate for HUIs. In Phase II, the proposed algorithms and UP-Growth/Two-Phase respectively identify CHUIs and HUIs from candidates produced in their Phase I. Furthermore, we have also considered the performance of CHUD with DAHU, denoted as CHUD+DAHU. CHUD+DAHU first applies CHUD to find all CHUIs and then uses DAHU to derive all HUIs from the set of CHUIs generated by CHUD.

The process of CHUD+DAHU in Phase I is the same as that of CHUD. In Phase II, CHUD+DAHU first identifies CHUIs from candidates and then uses CHUIs to derive all HUIs. In the experiments, we do not combine AprioriHC/AprioriHC-D with DAHU because CHUD outperforms these algorithms, as it will be shown, and they produce the same output. Experiments were performed on a desktop computer with an Intel Core 2 Quad Core Processor @ 2.66 GHz running Windows XP and 2 GB of RAM. All the algorithms were implemented in Java.

Both synthetic and real datasets were used to evaluate the performance of the algorithms. A synthetic dataset T12-I10-N1K-Q5-D200K was generated by the IBM data generator [1]. The parameters of the data generator are described in Table 3. Mushroom and BMSWebView1 were obtained from FIMI Repository [32]. Mushroom is a real-life dense dataset, each transaction containing 23 items. BMSWebView1 is a real-life dataset of click-stream data with a mix of short and long transactions (up to 267 items). Foodmart is a real-life dataset obtained from the Microsoft foodmart 2,000 database, which contains real external and internal utilities. For remaining datasets, the quantity of each item is randomly generated from 1 to 5 and the external utility of each item is randomly generated from 0.01 to 10.00. The external utility follows a log normal distribution [13], [17], [24]. Table 4 shows the characteristics of the above datasets.

Depending on the applications, the characteristics and count distributions of the datasets can be very different. However, there are three kinds of datasets that are commonly encountered in real-life scenarios: (1) dense dataset, (2) sparse dataset, and (3) dataset containing long transactions. In the experiments, we use three real-life datasets Mushroom, Foodmart, BMSWebView1 to respectively represent the above three real cases. The experimental results on these datasets are separately shown and discussed in the Sessions 4.1, 4.2 and 4.3. The scalability and the memory consumption of the proposed algorithms are respectively shown in the Sessions 4.4 and 4.5.

4.1 Experiments on Mushroom Dataset

The performance of the algorithms on the Mushroom dataset is shown in Fig. 15. In Fig. 15a, the execution time of Two-Phase and AprioriHC is similar in Phase I. The reason is that Two-Phase and AprioriHC simply apply the TWU-Model without using effective strategies to reduce the estimated utility of candidates in Phase I. Besides, AprioriHC-D runs faster than Two-Phase and AprioriHC. Table 5 shows the number of candidates generated by Two-Phase, AprioriHC and AprioriHC-D in Phase I. We did not show the number of HUIs and CHUIs in Table 5 because there are no HUIs and CHUIs when *min_utility* is higher than 10 percent. In Table 5, AprioriHC and AprioriHC-D generate fewer candidates than Two-Phase. This is because AprioriHC and AprioriHC-D produce candidates for CHUIs but Two-Phase needs to produce candidates for all HUIs. By applying strategies DGU and IIDS, AprioriHC-D produces fewer candidates than AprioriHC. In Fig. 15b, AprioriHC runs faster than Two-Phase because Two-Phase needs to verify the utility of more candidates in Phase II. Though AprioriHC needs to calculate the utility unit array of candidates and Two-Phase does not, this cost is not expensive. In Fig. 15f, we can see that CHUD outperforms all the other algorithms for both phases. For example, when *min_utility* = 1%, CHUD is 50 times faster than UP-Growth for Phase I and 63 times faster for Phase II. Moreover, when CHUD is combined with DAHU to discover all HUIs, the combination

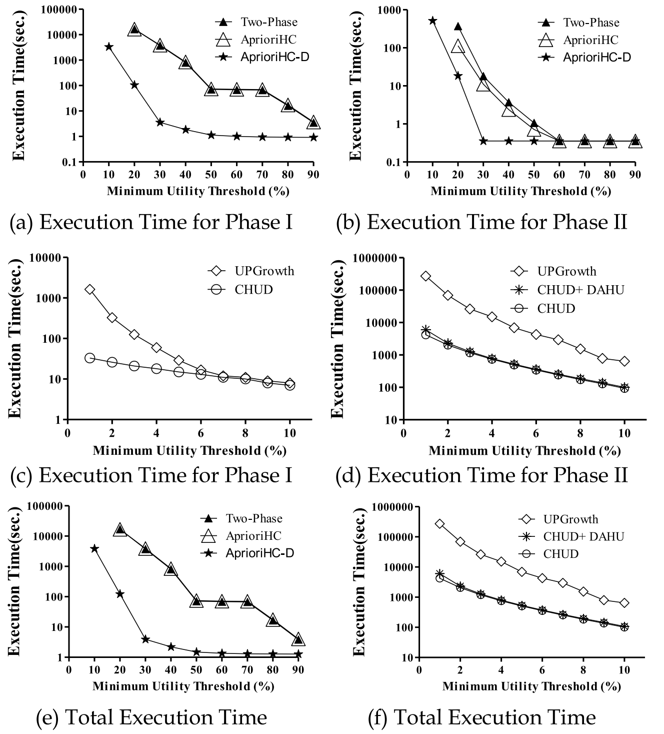


Fig. 15. Execution time on mushroom dataset.

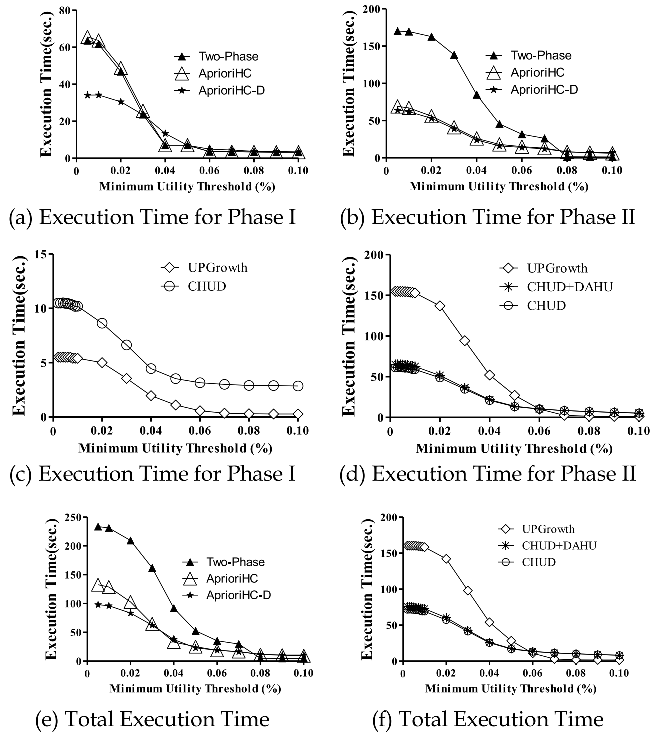


Fig. 16. Execution time on foodmart dataset.

TABLE 5
Number of Candidates in Phase I on Mushroom

Minimum Utility	#Cand. for Two-Phase	#Cand. for AprioriHC	#Cand. for AprioriHC-D
80%	25	18	5
60%	51	35	8
40%	533	341	29
20%	53,127	16,194	2,635

TABLE 6
Number of Extracted Patterns on Mushroom

Minimum Utility	UP-Growth		CHUD	
	Phase I	Phase II	Phase I	Phase II
	#Cand. for HUIs	#HUIs	#Cand. for CHUIs	#CHUIs
10%	84,392	6,655	889	157
6%	692,470	72,810	3,311	634
2%	15,687,252	3,381,719	19,362	4,911
1%	68,634,458	20,392,064	39,522	25,611

largely outperforms UP-Growth and was only slightly slower than CHUD.

Table 6 shows the number of candidates and the number of results generated by UP-Growth, CHUD, and CHUD+DAHU. CHUD generates a much smaller number of candidates and results than UP-Growth. The smaller number of candidates generated by CHUD in Phase I is what makes CHUD perform better than UP-Growth for the total execution time. In Table 6, a huge reduction in the number of extracted patterns (up to 796 times) is achieved by the representation of closed$^+$ high utility itemsets. Moreover, by running DAHU, it is possible to recover all HUIs.

TABLE 7
Number of Candidates in Phase I on Foodmart

Minimum Utility	#Cand. for Two-Phase	#Cand. for AprioriHC	#Cand. for AprioriHC-D
0.1%	1,728	1,641	1,627
0.05%	62,514	2,981	2,936
0.01%	232,505	6,345	6,345
0.005%	233,185	6,657	6,657

4.2 Experiments on Foodmart Dataset

The performance of the algorithms on the Foodmart dataset is shown in Fig. 16. Results show that AprioriHC-D runs faster than both Two-Phase and AprioriHC. The execution time of AprioriHC and AprioriHC-D is similar in Phase II. When $min\_utility = 0.05\%$, AprioriHC-D is three times faster than Two-Phase in total execution time. Table 7 shows the number of candidates generated by Two-Phase, AprioriHC and AprioriHC-D. AprioriHC and AprioriHC-D generate much less candidates than Two-Phase. For example, when $min\_utility = 0.05\%$, Two-Phase generates 233,185 candidates, and AprioriHC and AprioriHC-D generate both 6,657 candidates. The reason is that AprioriHC and AprioriHC-D produce candidates for CHUIs, but Two-Phase needs to produce candidates for all HUIs.

In Fig. 16f, the total execution time of UP-Growth is less than CHUD, initially. But as the *min_utility* threshold became smaller, CHUD becomes faster (up to twice faster than UP-Growth). The reason why the performance gap between CHUD and UP-Growth is smaller for Foodmart than for Mushroom is due to the fact that Foodmart is a sparse dataset. As a consequence the reduction achieved by mining CHUIs is less (still up to 34.6 (230,617/6,656) times, as shown in Table 8). Note that achieving a smaller

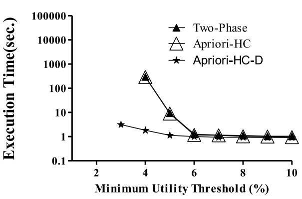
TABLE 8
Number of Extracted Patterns on Foodmart

Minimum Utility (%)	UP-Growth		CHUD	
	Phase I #Cand. for HUIs	Phase II #HUIs	Phase I #Cand. for CHUIs	Phase II #CHUIs
0.1%	1,585	258	581	258
0.05%	37,158	6,266	1,444	1,076
0.01%	230,165	209,387	6,332	6,293
0.005%	233,032	230,617	6,657	6,656

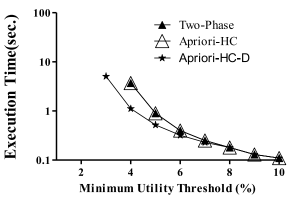
reduction for sparse datasets is a well-known phenomenon in frequent closed itemset mining. A similar phenomenon occurs in closed$^+$ high utility itemset mining. Besides, when DAHU was combined with CHUD, the execution time of CHUD+DAHU was up to twice faster than UP-Growth and slightly slower than CHUD.

4.3 Experiments on BMSWebView1 Dataset

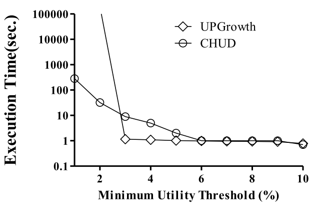
The performance of the algorithms on the BMSWebView1 dataset is shown in Fig. 17. In Fig. 17a, the execution time of Two-Phase and AprioriHC is similar in Phase I. In Fig. 17b, AprioriHC-D runs faster than Two-Phase and AprioriHC in Phase II. When $min\_utility = 4\%$, AprioriHC and Two-Phase cannot terminate within the time limit of 100,000 seconds and they generate more than 1,000,000 candidates in Phase I. When $min\_utility = 3\%$, AprioriHC-D performance starts degrading since too many candidates are produced. In Figs. 17c and 17d, UP-Growth runs faster than CHUD and CHUD+DAHU for $min\_utility \geq 3\%$. However, for $min\_utility < 3\%$, the performance of UP-Growth decreases sharply. For $min\_utility = 2\%$, UP-Growth cannot terminate within the time limit of 100,000 seconds and it generates more than 1,000,000 candidates in Phase I, whereas CHUD terminates in 80 seconds and produces only 7 CHUIs from 32 candidates. The reason why CHUD performs so well is that it achieves a massive reduction in the number of candidates by only generating a few long itemsets containing up to 149 items, while UP-Growth has to consider a huge amount of redundant subsets (for a closed itemset of 149 items, there can be up to $2^{149} - 2$ non-empty subsets that are redundant). DAHU also suffers from the fact that there are too many HUIs. It runs out of memory for $min\_utility < 2\%$ when trying to recover all HUIs.



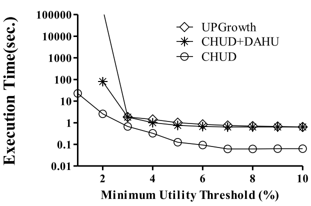
(a) Execution Time for Phase I



(b) Execution Time for Phase II

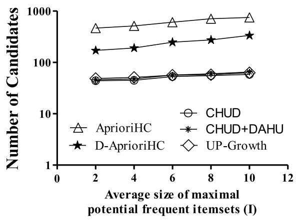


(c) Execution Time for Phase I

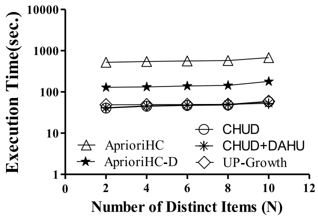


(d) Execution Time for Phase II

Fig. 17. Execution time on BMSWebView1 dataset.



(a) Varied parameter I



(b) Varied parameter N

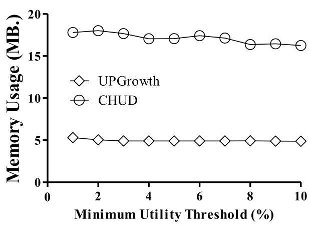
Fig. 18. Scalability test.

4.4 Scalability of the Proposed Methods

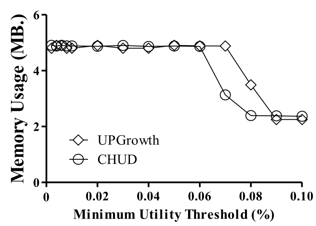
In this section, we evaluate the scalability of the algorithms. The scalability of the proposed algorithms under varied size of potential maximal frequent patterns is shown in Fig. 18a. The experiments are performed on synthetic datasets T12-Ix-N1K-Q5-D100K, where x is varied from 2 to 10 (e.g. I2, I4, I6, I8 and I10).The scalability of the proposed algorithms under varied number of distinct items is shown in Fig. 18b. The experiments are performed on synthetic datasets T12-I8-NxK-Q5-D100K, where x is varied from 2 to 10 (e.g. N2K, N4K, N6K, N8K and N10K). In these two experiments, the $min\_utility$ threshold is fixed to 0.5 percent. As shown in Fig. 18, all the proposed algorithms have good scalability when I and N are varied. Besides, CHUD and CHUD+DAHU perform better than the other algorithms.

4.5 Memory Usage

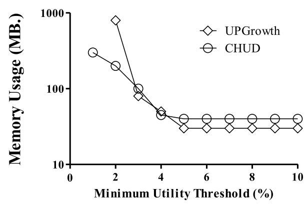
We measure the maximum memory usage of UP-Growth and CHUD in phase I by using the Java API. In general, CHUD uses as much or more memory than UP-Growth because the latter uses a compact trie-based data structure for representing the database that is more memory efficient than a vertical database. For example, detailed results for Mushroom and Foodmart are presented in Figs. 19a and 19b. However, when the databases contain very long HUIs such as BMSWebView1, the number of candidates can be very large. As shown in Fig. 19c, when the minimum utility



(a) Mushroom



(b) Foodmart



(c) BMSWebView1

Fig. 19. Memory consumption.

threshold is set to 2 percent , the memory consumption of UP-Growth rises dramatically because it needs to create a number of conditional UPTrees that is proportional to the number of candidates.

5 Discussions

In this section, we summarize the experimental results and compare characteristics of the different algorithms. First, we compare AprioriHC and AprioriHC-D. In general, AprioriHC-D runs faster and produces fewer candidates than AprioriHC. This is because AprioriHC-D applies DGU and IIDS to prune candidates and it calculates the exact utilities of candidates by scanning the trimmed database instead of the original database. For dense datasets such as Mushroom, AprioriHC-D performs better than AprioriHC when *min_utility* is high because there are many unpromising items and isolated items when *min_utility* is high. In the experiments, both algorithms can't perform well on dense databases when *min_utility* is low since they suffer from the problem of a large amount of candidates.

The most efficient algorithm is CHUD. The overall execution time of CHUD is always faster than UP-Growth, especially when there is much less CHUIs than HUIs. Moreover, the combination of CHUD and DAHU is also faster than UP-Growth for total execution time. In the experiments, deriving all HUIs is inexpensive. Though CHUD + DAHU performs better than UP-Growth, it may fail to recover all high utility itemsets due to the memory limitation when there are too many HUIs in the database.

Although CHUD performs better than the proposed two Apriori-based approaches, the Apriori-based approaches are easier to be comprehended and implemented by the readers. Besides, for some applications such as discovering patterns in the presence of the memory constraint [5], Apriori-based approach is preferred and plays an essential role. Although AprioriHC and AprioriHC-D do not perform better than CHUD in some cases, they are quite general and easy to be extended for more applications.

Depending on the characteristics of datasets, the reduction ratios achieved by the proposed representation can be very different. For the dense dataset such as Mushroom, the proposed representation can achieve a massive reduction in the number of extracted patterns. For the sparse dataset such as Foodmart, the proposed representation achieves less reduction. For the dataset containing very long transactions such as BMSWebView1, a massive reduction can be achieved by the proposed representation. Although the proposed representation may not achieve a massive reduction on very sparse datasets, it still has good performance in several real cases.

6 Conclusions

In this paper, we addressed the problem of redundancy in high utility itemset mining by proposing a lossless and compact representation named *closed*$^+$ *high utility itemsets*, which has not been explored so far. To mine this representation, we proposed three efficient algorithms named *AprioriHC* (*Apriori-based approach for mining High utility Closed itemset*), *AprioriHC-D* (*AprioriHC algorithm with Discarding unpromising and isolated items*) and *CHUID* (*Closed*$^+$ *High Utility itemset Discovery*). AprioriHC-D is an improved version of AprioriHC, which incorporates strategies DGU [24] and IIDS [19] for pruning candidates. *AprioriHC* and *AprioriHC-D* perform a breadth-first search for mining closed$^+$ high utility itemsets from horizontal database, while CHUD performs a depth-first search for mining closed$^+$ high utility itemsets from vertical database. The strategies incorporated in CHUD are efficient and novel. They have never been used for vertical mining of high utility itemsets and closed$^+$ high utility itemsets. To efficiently recover all high utility itemsets from closed$^+$ high utility itemsets, we proposed an efficient method named *DAHU* (*Derive All High Utility itemsets*). Results on both real and synthetic datasets show that the proposed representation achieves a massive reduction in the number of high utility itemsets on all real datasets (e.g. a reduction of up to 800 times for Mushroom and 32 times for Foodmart). Besides, CHUD outperforms UP-Growth, one of the currently best algorithms by several orders of magnitude (e.g. CHUD terminates in 80 seconds on BMSWebView1 for $min\_utility = 2\%$, while UP-Growth cannot terminate within 24 hours). The combination of CHUD and DAHU is also faster than UP-Growth when DAHU could be applied.

7 Future Works

In this work, we only study the integration of closed itemset mining and high utility itemset mining. However, there are many other compact representations [3], [4], [11], [15] that have not yet been combined with high utility itemset mining. Is it possible to develop other compact representations of high utility itemsets inspired by our work to reduce the number of redundant high utility patterns? This is an interesting research question. Although closed$^+$ high utility itemset mining is essential to many research topics and industrial applications, it is still a novel and challenging problem. Other related research issues are worthwhile to be explored in the future.

Acknowledgments

This research was partially supported by Ministry of Science and Technology, Taiwan, R.O.C. under Grant no. 101-2221-E-006-255-MY3 and US National Science Foundation (NSF) Grant OISE-1129076.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.

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

[3] J.-F. Boulicaut, A. Bykowski, and C. Rigotti, "Free-sets: A condensed representation of Boolean data for the approximation of frequency queries," *Data Mining Knowl. Discovery*, vol. 7, no. 1, pp. 5–22, 2003.

[4] T. Calders and B. Goethals, "Mining all non-derivable frequent itemsets," in *Proc. Int. Conf. Eur. Conf. Principles Data Mining Knowl. Discovery*, 2002, pp. 74–85.

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

[6] R. Chan, Q. Yang, and Y. Shen, "Mining high utility itemsets," in *Proc. IEEE Int. Conf. Data Min.*, 2003, pp. 19–26.

[7] A. Erwin, R. P. Gopalan, and N. R. Achuthan, "Efficient mining of high utility itemsets from large datasets," in *Proc. Int. Conf. Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2008, pp. 554–561.

[8] K. Gouda and M. J. Zaki, "Efficiently mining maximal frequent itemsets," in *Proc. IEEE Int. Conf. Data Mining*, 2001, pp. 163–170.
[9] T. Hamrouni, "Key roles of closed sets and minimal generators in concise representations of frequent patterns," *Intell. Data Anal.*, vol. 16, no. 4, pp. 581–631, 2012.
[10] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 1–12.
[11] T. Hamrouni, S. Yahia, and E. M. Nguifo, "Sweeping the disjunctive search space towards mining new exact concise representations of frequent itemsets," *Data Knowl. Eng.*, vol. 68, no. 10, pp. 1091–1111, 2009.
[12] H.-F. Li, H.-Y. Huang, Y.-C. Chen, Y.-J. Liu, and S.-Y. Lee, "Fast and memory efficient mining of high utility itemsets in data streams," in *Proc. IEEE Int. Conf. Data Mining*, 2008, pp. 881–886.
[13] C.-W. Lin, T.-P. Hong, and W.-H. Lu, "An effective tree structure for mining high utility itemsets," *Expert Syst. Appl.*, vol. 38, no. 6, pp. 7419–7424, 2011.
[14] G.-C. Lan, T.-P. Hong, and V. S. Tseng, "An efficient projection-based indexing approach for mining high utility itemsets," *Knowl. Inf. Syst*, vol. 38, no. 1, pp. 85–107, 2014.
[15] H. Li, J. Li, L. Wong, M. Feng, and Y. Tan, "Relative risk and odds ratio: A data mining perspective," in *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles Database Syst.*, 2005, pp. 368–377.
[16] B. Le, H. Nguyen, T. A. Cao, and B. Vo, "A novel algorithm for mining high utility itemsets," in *Proc. 1st Asian Conf. Intell. Inf. Database Syst.*, 2009, pp. 13–17.
[17] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in *Proc. Utility-Based Data Mining Workshop*, 2005, pp. 90–99.
[18] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 21–36, Jan. 2006.
[19] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 198–217, 2008.
[20] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient mining of association rules using closed itemset lattice," *J. Inf. Syst.*, vol 24, no. 1, pp. 25–46, 1999.
[21] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-mine: Fast and space-preserving frequent pattern mining in large databases," *IIE Trans.*, vol. 39, no. 6, pp. 593–605, Jun. 2007.
[22] B.-E. Shie, H.-F. Hsiao, V. S. Tseng, and P. S. Yu, "Mining high utility mobile sequential patterns in mobile commerce environments," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2011, vol. 6587, pp. 224–238.
[23] B.-E. Shie, V. S. Tseng, and P. S. Yu, "Online mining of temporal maximal utility itemsets from data streams," in *Proc. Annu. ACM Symp. Appl. Comput.*, 2010, pp. 1622–1626.
[24] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "UP-Growth: An efficient algorithm for high utility itemset mining," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2010, pp. 253–262.
[25] B. Vo, H. Nguyen, T. B. Ho, and B. Le, "Parallel method for mining high utility itemsets from vertically partitioned distributed databases," in *Proc. Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst.*, 2009, pp. 251–260.
[26] C.-W. Wu, B.-E. Shie, V. S. Tseng, and P. S. Yu, "Mining top-k high utility itemsets," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 78–86.
[27] J. Wang, J. Han, and J. Pei, "Closet+: Searching for the best strategies for mining frequent closed itemsets," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 236–245.
[28] C.-W Wu, P. Fournier-Viger, P. S. Yu, and V. S. Tseng, "Efficient mining of a concise and lossless representation of high utility itemsets," in *Proc. IEEE Int. Conf. Data Mining*, 2011, pp. 824–833.
[29] U. Yun, "Mining lossless closed frequent patterns with weight constraints," *Knowl.-Based Syst.*, vol. 20, pp. 86–97, 2007.
[30] H. Yao, H. J. Hamilton, and L. Geng, "A unified framework for utility-based measures for mining itemsets," in *Proc. ACM SIGKDD 2nd Workshop Utility-Based Data Mining*, 2006, pp. 28–37.
[31] M. J. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," in *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 462–478, Apr. 2005.
[32] Frequent itemset mining implementations repository http://fimi.ua.ac.be/data/

Vincent S. Tseng received the PhD degree with major in computer science from National Chiao Tung University, Taiwan, in 1997. He is currently a distinguished professor in the Department of Computer Science and Information Engineering, National Cheng Kung University (NCKU), Taiwan. He is the chair for IEEE CIS Tainan Chapter. Before joining NCKU in 1999, he was a postdoctoral research fellow in Computer Science Division, University of California at Berkeley. He was the president of Taiwanese Association for Artificial Intelligence during 2011-2012 and the director for Institute of Medical Informatics of NCKU during August 2008 and July 2011. During February 2004 and July 2007, he was the director for Informatics Center in National Cheng Kung University Hospital. He has a wide variety of research interests covering data mining, big data, biomedical informatics, multimedia databases, mobile and web technologies. He has published more than 280 research papers in referred journals and international conferences a has 15 patents. He has been on the editorial board of a number of journals including the *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Journal on Biomedical and Health Informatics*, *ACM Transactions on Knowledge Discovery from Data*, etc. He was the program chair of PAKDD'14 and was a chair/program committee member for a number of premier international conferences related to data mining and biomedical informatics.

Cheng-Wei Wu received the MSs degree in computer science and information engineering from Ming Chuan University, Taiwan, ROC. in 2009. He is currently working toward the PhD degree at the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC. His research interests include data mining, utility mining, pattern discovery, machine learning and big data analytics.

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

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