Efficient Mining of a Concise and Lossless Representation of High Utility Itemsets

Cheng Wei Wu[1], Philippe Fournier-Viger[1], Philip S. Yu[2], Vincent S. Tseng[1]
[1]*Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC*
[2]*Department of Computer Science, University of Illinois at Chicago, Chicago, Illinois, USA*
{*silvemoonfox, philippe.fv*}*@gmail.com, psyu@cs.uic.edu, tsengsm@mail.ncku.edu.tw*

***Abstract*—Mining *high utility itemsets* from transactional databases is an important data mining task, which refers to the discovery of itemsets with high utilities (e.g. high profits). Although several studies have been carried out, current methods may present too many high utility itemsets for users, which degrades the performance of the mining task in terms of execution time and memory requirement. 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*, which serves as a compact and lossless representation of high utility itemsets. We present an efficient algorithm called *CHUD* (**C**losed$^+$ *High utility* itemset *D*iscovery) for mining closed$^+$ high utility itemsets. Further, a method called *DAHU* (*D*erive *A*ll *High Utility* itemsets) is proposed to recover all high utility itemsets from the set of closed$^+$ high utility itemsets without accessing the original database. Results of experiments on real and synthetic datasets show that CHUD and DAHU are very efficient and that our approach achieves a massive reduction in the number of high utility itemsets (up to 800 times in our experiments). In addition, when all high utility itemsets can be recovered by DAHU, the combination of CHUD and DAHU outperforms the state-of-the-art algorithms for mining high utility itemsets.**

Keywords-utility mining; frequent itemset; closed*$^+$ *high utility itemset; lossless and concise representation

I. INTRODUCTION

Frequent itemset mining (abbreviated as FIM) [1, 10] 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 doesn't 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, 5, 6, 7, 8, 11, 13, 15, 19, 20, 24, 26] 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* (abbreviated as 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, 5, 19, 24], cross-marketing in retail stores [6, 20, 26] and biomedical applications [7].

However, HUIs mining is not an easy task since the *downward closure property* [1, 10] in FIM does not hold in utility mining. In other words, the search space cannot be directly pruned to find HUIs as it is done in FIM because a superset of a low utility itemset can be a high utility itemset. Many studies [2, 13, 15, 20, 21] 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.

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], *maximal itemsets* [9] and *closed itemsets* [14, 16-18 22, 27]. These representations successfully reduce the set of itemsets found, but they are developed for frequent itemset mining instead of high utility itemset 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* (Closed$^+$ HUIs), which integrates the concept of closed itemset into HUI 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 HUIs (up to 800 times).
3. We propose an efficient algorithm, named *CHUD* (*C*losed$^+$ *H*igh *U*tility itemset *D*iscovery), to find this representation. It includes three novel strategies named *REG*, *RML* and *DCM* that greatly enhance its performance. Results show that CHUD is much faster than the current best methods for mining all HUIs [20].
4. We propose a top-down method named *DAHU* (*D*erive *A*ll *H*igh *U*tility itemsets) for efficiently recovering all HUIs from the set of Closed$^+$ HUIs. The combination of CHUD and DAHU provides a new way to obtain all HUIs and outperforms *UPGrowth* [20], the state-of-the-art algorithm for mining HUIs.

The remainder of this paper is organized as follows. In Section II, we introduce the background for compact representations and utility mining. Section III defines the representation of closed$^+$ HUIs and presents our methods. Experiments are shown in Section IV and conclusions are given in Section V.

TABLE I. AN EXAMPLE TRANSACTIONAL DATABASE

TID	Transaction	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

TABLE II. UNIT PROFITS FOR EVERY ITEM

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

II. BACK GROUND

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

A. 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 $T_R \in D$ $(1 \le R \le N)$ 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 $a_i \in I$, $1 \le i \le K$, and K is 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. 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 denoted as L and $L = \{X \mid X \subseteq I, SC(X) > 0\}$.

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

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

Definition 4. The *utility of an itemset X in D* is denoted as $u(X)$ and defined as $u(X) = \sum_{X \subseteq T_R \wedge T_R \in D} u(X, T_R)$.

Definition 5. An itemset X is called *high utility itemset* if $u(X)$ is no less than a user-specified minimum utility threshold *min_utility*. Otherwise, X is a *low utility itemset*.

Definition 6. Let S be a set of itemsets and a function $f_H(S) = \{X \mid X \in S, u(X) \ge min\_utility\}$. 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. Let Table I be a database containing five transactions. Each row in Table I represents a transaction, in which each letter represents an item and has a purchase quantity (internal utility). The unit profit of each item is shown in Table II (external utility). In Table I, the utility of the item {F} in the transaction T_3 is $u(\{F\}, T_3) = p(\{F\}, D) \times q(\{F\}, T_3) = 3 \times 2 = 6$. The utility of {BF} in T_3 is $u(\{BF\}, T_3) = u(\{B\}, T_3) + u(\{F\}, T_3) = 1 + 6 = 7$. The utility of {BF} is $u(\{BF\}) = u(\{BF\}, T_3) + u(\{BF\}, T_5) = 17$, since {BF} is contained in T_3 and T_5. If the *min_utility* is set to 10, the set of HUIs in Table I is H = {{E}:12, {F}:15, {AE}:10, {AF}:17, {BE}:10, {BF}:17, {ABE}:12, {ABF}:19}, where the number beside each itemset is its utility.

Note that the utility constraint is neither *monotone* nor *anti-monotone*. In other words, a superset of a low utility itemset can be high utility and a subset of a high utility itemset 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* [13], which is based on the following definitions.

Definition 7. The *transaction utility* of a transaction T_R is denoted as $TU(T_R)$ and defined as $u(T_R, T_R)$.

Definition 8. The *transaction-weighted utilization* 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. An itemset X is a *high transaction-weighted utilization itemset* (*HTWUI*) if $TWU(X) \geq min\_utility$.

Property 1. 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, 13, 15, 20].

For example, the transaction utilities of T_1 and T_3 are $TU(T_1) = u(\{ABE\}, T_1) = 5$ and $TU(T_3) = 8$. When $min\_utility = 10$, {AB} is a HTWUI since $TWU(\{AB\}) = TU(T_1) + TU(T_3) = 13$ is no less than $min\_utility$. In contrast, the itemset {W} is not a HTWUI, and therefore all the supersets of {W} are low utility itemsets.

Many studies have been proposed for mining HUIs, including *Two-Phase* [13], *IHUP* [2], *TWU-Mining* [21], *IIDS* [15] and *UPGrowth* [20]. *Two-Phase*, *IHUP* and *TWU-Mining* utilize *transaction-weighted downward closure* property to find high utility itemsets. They consist of two phases. In phase I, they find all HTWUIs from the database. In phase II, high utility itemsets are identified from the set of HTWUIs by scanning the original database once. 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 (in terms of time and space). To reduce the number of candidates in phase I, various methods have been proposed (e.g. [15, 20]). Recently, Tseng et al. proposed *UPGrowth* with four strategies *DGU*, *DGN*, *DLU* and *DLN*, for mining HUIs. Experiments in [20] show that the number of candidates generated by UPGrowth in phase I can be order of magnitudes smaller than that of HTWUIs. To the best of our knowledge, UPGrowth is the state-of-the-art method for mining HUIs.

Although the above methods perform well in some case, 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 (the memory, disk space or 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 that eliminate redundancy, such as *free sets* [3], *non-derivable sets* [4], *maximal itemsets* [9] and *closed itemsets* [16-18]. Although these representations achieve a significant reduction in the number of extracted frequent itemsets, some of them lead to loss of information (e.g. [9]). To provide not only compact but also complete information about frequent itemsets to users, many studies were conducted on closed itemset mining.

B. Closed Itemset Mining

In this subsection, we introduce definitions and properties related to closed itemsets and mention relevant methods. For more details about closed itemsets, readers can refer to [14, 16-18, 22, 27].

Definition 10. 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 an itemset 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. 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 $C(X) = \cap_{R \in g(X)} T_R$.

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

Definition 12. An itemset $X \in L$ is a *closed itemset* if there exists no itemset $Y \in L$ such that (1) $X \subset Y$ and (2) $SC(X) = SC(Y)$. Otherwise X is a *non-closed itemset*. An equivalent definition is that X is closed if $C(X) = X$. For example, {B} is non-closed since $C(\{B\}) = T_1 \cap T_2 \cap T_3 \cap T_5 = \{AB\}$.

Definition 13. 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 $f_C(L)$. For example, the set of closed itemsets in Table I is C = {{E}:3, {EG}:1, {AB}:4, {ABE}:2, {ABF}:2, {ABEW}:1}, in which the number beside each itemset is its support count.

Property 4. $\forall X \in L$, $SC(X) = max\{SC(Y) \mid Y \in f_C(L) \wedge X \subseteq Y\}$. For example, the supersets of {B} in $f_C(L)$ are {AB}:4, {ABE}:2, {ABF}:2 and {ABEW}:1. Thus, $SC(\{B\})$ is the maximum of these support counts, i.e. 4.

Mining *frequent closed itemset* refers to the discovery of all the closed itemsets whose supports are no less than a user-specified threshold. It is widely recognized that the number of frequent closed itemsets can be much smaller than the set of frequent itemsets for real-life databases and that mining frequent closed itemsets can also be much faster and memory efficient than mining frequent itemsets [14, 22, 27]. The set of closed itemsets is *lossless* since all frequent itemsets and their supports can be easily derived from it by property 4 without scanning the original database [16-18]. Many efficient methods were proposed for mining frequent closed itemsets, such as *A-Close* [16-18], *CLOSET+* [22], *CHARM* [27] and *DCI-Closed* [14]. 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.

C. Compact Representations of High Utility Itemsets

To present representative HUIs to users, some concise representations of HUIs were proposed. Chan et al. introduced the concept of *utility frequent closed patterns* [7]. However, it is based on a definition of high utility itemset that is different from [2, 13, 15, 20] and our work. Shie et al. proposed a compact representation of high utility itemsets, called *maximal high utility itemset* and the *GUIDE* algorithm for mining it [19]. A HUI is said to be *maximal* if it is not a subset of any other HUI. For example, when $min\_utility = 10$, the set of maximal HUIs 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.

III. 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 not larger than the set of all HUIs.

A. Pushing Closed Property into HUI 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 {E}⊆{ABE} and u({E}) = u({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 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 [23], 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 4, $u(Y) > u(X) \geq min\_utility$, 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 Lemma 1 and Lemma 2.

Because the two join orders produce the same result, we remove the join order to obtain a general definition.

Definition 14. We define the set of *closed high utility itemsets* as $HC = \{X \mid X \in L, X = C(X), u(X) \geq min\_utility\}$, $HC = H' = C'$. An itemset X is called a *non-closed high utility itemset* if $X \in H$ and $X \notin C$. For example, the set of closed HUIs in Table I 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 4.

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 purchase X without its closure Y. Thus, when a customer purchase 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. $\forall X = \{a_1, a_2, \ldots, a_K\} \in L$, the *utility unit array* of X is denoted as $V(X) = [v_1, v_2, \ldots, v_K]$ and contains K utility values. The i-th utility value v_i in $V(X)$ is denoted as $V(X, a_i)$ and defined as $\Sigma_{R \in g(X) \wedge a_i \in T_R} u(a_i, T_R)$.

For example, consider the itemset {ABE} appearing in T_1 and T_2. The first utility value in V({ABE}) is V({ABE}, {A}) = u({A}, T_1) + u({A}, T_2) = 2. The utility unit array of {ABE} is V({ABE}) = [2, 2, 8].

Property 6. $\forall X = \{a_1, a_2, \ldots, a_K\} \in L$, $u(X) = \Sigma_{i=1}^{K} V(X, a_i)$.
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, the value $V(X, a_i)$ represents the sum of the utilities of a_i in transactions containing X. Therefore $u(X)$ can be expressed as $V(X, a_1) + V(X, a_2) + \ldots + V(X, a_K)$. For example, u({ABE}) = V({ABE}, {A}) + V({ABE}, {B}) + V({ABE}, {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 4 we have $u(X) \leq u(C(X)) < min\_utility$.

Property 8. $\forall X = \{a_1, a_2, \ldots, a_K\} \in L$, the utility of X can be calculated as $u(X) = \Sigma_{a_i \in X} V(C(X), a_i)$ by using the utility unit array of its closure if $C(X) \in HC$.
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 $X \subseteq C(X)$ (Property 3). Therefore, $V(X, a_i) = V(C(X), a_i)$, by Definition 15. According to Property 6, $u(X) = \Sigma_{i=1}^{K} V(X, a_i)$. By replacing $V(X, a_i)$ with $V(C(X), a_i)$, we obtain Property 8.

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

Although the set of closed$^+$ HUIs is *meaningful*, *concise* and *lossless*, mining closed$^+$ HUIs is not an easy task. There are two naive methods. The first one is to find all HUIs and then to remove non-closed itemsets. The main drawbacks of this method are that it cannot be more efficient than mining all HUIs and that in the worst case removing all non-closed itemset requires comparing all HUIs with each other. The second approach is to first mine all closed itemsets and then to remove those that are low utility itemsets. The drawback of this method is that it needs to generate all closed itemsets and this set can be very large since no threshold can be used.

B. Efficient Discovery of Closed$^+$ High Utility Itemsets

In this subsection, we present an efficient algorithm named *CHUD* (*C*losed$^+$ *H*igh *U*tility itemset *D*iscovery) for mining closed$^+$ HUIs. CHUD is an extension of DCI-Closed [14], one of the current best methods for mining closed itemsets, and it also integrates the TWU model and effective strategies to prune low utility itemsets. CHUD consists of two phases. In phase I, CHUD discovers candidates for closed$^+$ HUIs. In phase II, the closed$^+$ HUIs are identified from the set of candidates found in phase I and their utility unit arrays are computed by scanning the database once.

Similar to the DCI-Closed algorithm, CHUD adopts an *IT-Tree* (Itemset-Tidset pair Tree) [14, 27] to find closed$^+$ HUIs. 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 *TU-Table* (*T*ransaction *U*tility *Table*) [13] 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 Figure 1.

The main procedure of CHUD is named *Main* and is shown in Figure 2. It takes as parameter a database D and the *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*. An item is called a *promising item* if its estimated utility (e.g. its TWU) is no less than *min_utility*. After the first scan of database, promising items are collected into an ordered list $O = <a_1, a_2, \ldots, a_n>$, 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 low utility itemsets. According to [22], 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 k-th subspace is the set of itemsets containing the item a_k but no item $a_i \prec a_k$ [14]. 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 $a_i \prec a_k$. Finally, the *Main* procedure performs phase II on these candidates to obtain all closed$^+$ HUIs.

```
CalculateEstUtility(g(X), TU)
01.   EstU := 0;
02.   for each TID R ∈ g(X) do
03.   {     EstU := EstU + TU.get(R)     }
04.   return EstU
```

Figure 1. CalculateEstUtility

```
Main(D, min_utility)
01.   InitialDatabaseScan(D)
02.   RemoveUtilityUnpromisingItems(O, GTU).
03.   for each item a_k ∈ O do
04.         Create node N({a_k})
05.         CHUDPhase-I(N({a_k}), GTU, min_utility)
06.         //Apply Strategy 3(REG)
07.   PerformPhase-II(D)
```

Figure 2. Main

```
CHUDPhase-I (N_X, TU, min_utility)
01.   if (SubsumeCheck(N(X), PREV-SET(X)) == false) then
02.   {   X_C := ComputeClosure(N(X), POST-SET(X))
03.       if (EstU(X_C) ≥ min_utility) then  //Apply Strategy 5(DCM)
04.       { Output X_C with EstU(X_C)
05.         Explore(N(X_C), TU, min_utility)  } }
```

Figure 3. CHUDPhase-I

```
SubsumeCheck (N(X), PREV-SET(X))
01.   for each item a ∈ PREV-SET(X) do
02.   {     if (g(X) ⊆ g(a)) then return true  }
03.   return false
```

Figure 4. SubsumeCheck

ComputeClosure (*N*(*X*), POST-SET(*X*))
01. $X_C := X$
02. ***for each*** item $a \in$ POST-SET(*X*) ***do***
03. { ***if*** ($g(X) \subseteq g(a)$) ***then***
04. { POST-SET(*X*) := POST_SET$_X$/{*a*}
05. $X_C := X_C \cup \{a\}$ } }
06. ***return*** X_C

Figure 5. ComputeClosure

Explore (*N*(*X*), TU_X, *min_utility*)
01. ***for each*** item $a_k \in$ POST-SET(*X*) ***do***
02. { POST-SET(*X*) := POST-SET(*X*)/{a_k}
03. Create a node *N*(*Y*), where $Y := X \cup \{a_k\}$
04. $g(Y) := g(X) \cap g(a_k)$
05. POST-SET(*Y*) := POST-SET(*X*)
06. PREV-SET(*Y*) := PREV-SET(*X*)
07. *EstU*(*Y*) := *CalculateEstUtility*(*g*(*Y*), TU_X)
08. ***if*** (*EstU*(*Y*), *EstU*(*X*) ≥ *min_utility*) ***then***
09. { ***CHUDPhase-I*** (*N*(*Y*), TU_X, *min_utility*)
10. PREV-SET(*X*) := PREV-SET(*X*) ∪ {a_k} }
11. // Apply Strategy 4(RML) }

Figure 6. Explore

The *CHUDPhase*-I procedure shown in Figure 3 takes as parameter a node *N*(X), a TU-Table *TU* and the *min_utility* threshold. The procedure first performs *SubsumeCheck* on *X* as presented in Figure 4. 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 [14] for a complete justification). Otherwise, the next step is to compute the closure $X_C = C(X)$ of *X*. This is performed by the procedure *ComputeClosure*(*N*(*X*), POST-SET(*X*)) shown in Figure 5 [14]. Then the estimated utility of X_C is calculated. If it is no less than *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 to disk. After this, 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 Figure 6. It takes as parameter a node *N*(*X*), a TU-Table and the *min_utility* threshold. 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 [14]. 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*. If *EstU*(*Y*) and *EstU*(*X*) are no less than *min_utility*, the procedure *CHUDPhase*-I is recursively called with *N*(*Y*) to consider the search space of *Y* and a_k is added to PREV-SET(*X*). If *EstU*(*Y*) is lower than *min_utility*, the search space of *Y* is pruned since *Y* and its supersets are low utility itemsets (Property 1).

After recursions of the *Explore* and *CHUDPhase*-I procedures are completed, closed candidates that have been outputted are processed by phase II. Phase II consists of taking each candidate *X* and to calculate its exact utility and utility unit array. Each candidate that is a low utility itemset is discarded. Calculating the exact utility of a candidate *X* is performed by doing the summation of $u(X, T_R)$ 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).

We now prove that this basic version of the CHUD algorithm generates the complete set of closed$^+$ HUIs. We consider the two phases of CHUD to prove the correctness. The first phase produces a set of candidates $P \subseteq C$, since it is based on the DCI-Closed algorithm that generates all closed itemsets *C* (see [14] for the proof that DCI-Closed generates *C*). The second phase consists of discarding candidates that are low utilities from *P* to obtain *C*'. The algorithm is therefore correct if and only if $C' \subseteq P$ (the set of candidates *P* produced in Phase I contains all closed$^+$ HUIs *C*'). To prove this, we need to show that the modifications that have been made to DCI-Closed will not discard any closed$^+$ high utility itemset $X \subseteq C'$. We discuss the correctness of these modifications thereafter.

Strategy 1. Considering only promising items. 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 and 3 of the *Main* procedure. **Rationale.** It was shown in [14] that unpromising items cannot be part of a HUI and that the utility of unpromising items can be ignored in the calculation of the estimated utility of itemsets when searching for high utility itemsets.

Strategy 2. Discarding itemsets having an estimated utility lower than min_utility. The second strategy in CHUD is to discard the itemset X_C such that $EstU(X_C) \geq$ *min_utility*. This strategy is integrated in line 3 of the *CHUDPhase*-I procedure.
Rationale. It was demonstrated in Section 2 that an itemset that is not a HWTUI is not a high utility itemset as well as all of its supersets (see Property 1 and Definition 4, 8 and 9). Because DCI-Closed discovers candidates recursively by considering supersets of candidates, discarding an itemset such that $EstU(X_C) <$ *min_utility* will not discard any itemset from *P* that is in *C*'.

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. Removing the Exact utilities of items from the Global TU-Table (REG). Strategy 3 is called *REG*, which is applied after line 5 of the procedure *Main*. Each time that an item $a_k \in O$ has been processed, $u(a_k)$ is removed 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 $a_j \succ a_i$. Therefore, $u(a_i)$ can be removed from the transaction utility of each transaction containing a_j in the global TU-Table. The pseudo code for this strategy is shown as follows.

06. ***for each*** Tid $R \in g(a_k)$ ***do***
07. { ***remove*** $u(a_k)$ ***from*** <R, GTU(T_R)> }

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

Definition 18. 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 GTU.

Strategy 4. Removing the Mius of items from Local TU-Tables (RML). Strategy 4 is called *RML*, which is applied after line 11 of the procedure *Explore*. This strategy consists of using a local TU-Table TU_X for each node N(X) in the IT-Tree. Let $Y = X \cup \{a_k\}$ and N_Y be the child node of N_X. Each time that an item a_k from POST-SET(X) is processed, $miu(a_k)$ is removed 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 children nodes. Besides, $miu(a_k) \times SC(Y)$ is removed from $EstU(X)$. If the updated $EstU(X)$ is less than *min_utility*, the algorithm will not process $X \cup \{a_k\}$ for each item $a_k \in$ POST-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 $a_j \succ a_i$. Therefore, $miu(a_i) \times SC(Y)$ and $miu(a_i)$ can be removed from $EstU(X)$ and the transaction utility of each transaction containing a_j from TU_X. The pseudo code for this strategy is shown as follows.

11. ***for each*** Tid $R \in g(Y)$ ***do***
12. { ***remove*** $miu(a_k)$ ***from*** <R,$TU_X(T_c)$> }
13. ***remove*** $miu(a_k) \times SC(Y)$ ***from*** $EstU(X)$

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

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

Lemma 4. $\forall X$, X is low utility if $MAU(X) < min\_utility$.
Proof. The utility of an itemset X is the sum of the 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 equals to the utility of X.

Strategy 5. Discarding Candidates with a MAU that is less than the minimum utility threshold (DCM). The last strategy is called DCM and is applied to line 3 of the *CHUDPhase*-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 *min_utility*.
Rationale. Lemma 4 guarantees that an itemset X is not a closed$^+$ HUI if $MAU(X) < min\_utility$. The pseudo code for the strategy 5 is shown below.

03. ***if*** ($min\{EstU(X_C), MAU(X_C)\} \geq min\_utility$) ***then***

C. Efficient Recovery of High Utility Itemsets

In this subsection, we present a top-down method named *DAHU* (*D*erive *A*ll *High Utility* itemsets) for efficiently recovering all the HUIs. The pseudo code of DAHU is shown in Figure 7. It takes as input a *min_utility* threshold, a set of closed$^+$ HUIs *HC* and *Kmax* the maximum length of itemsets in *HC*. DAHU outputs the complete set of high utility itemsets $H = \cup_{i=1}^{K} H_K$ respecting *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_{Kmax} is initialized to HC_{Kmax}, where the notation HC_K represents the set of K-itemsets in *HC*. During step 2 to step 14 in Figure 7, each set H_K is constructed from $K = (Kmax - 1)$ 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 utility of X is no less than *min_utility*, the algorithm outputs the high utility itemset X with its exact 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 utility of Y is set to the utility of X minus the i-th value in $V(X)$, i.e., $u(Y) = u(X) - V(X, a_i)$ (Property 6-8). In addition, 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.

DAHU(HC, *min_utility*, *Kmax*)
01. $H_{Kmax} := HC_{Kmax}$
02. ***for*** ($K := Kmax - 1; K > 0 ; K--$) ***do***
03. { ***for each*** K-itemset $X = \{a_1, a_2, \ldots, a_K\}$ in CH_K ***do***
04. { ***if*** ($u(X) <$ *min_utility*) ***then*** delete X from CH_K
05. ***else*** add X and its exact utility $u(X)$ to H.
06. { ***for each*** item $a_i \in X$ ***do***
07. { $Y := X - \{a_i\}$
08. $u(Y) := u(X) - V(X, a_i)$
09. ***if***($u(Y) \geq$ *min_utility*) ***then***
10. { ***if*** $Y \in CH_{(K-1)}$ ***and*** $SC(X) > SC(Y)$ ***then***
11. { $SC(Y) := SC(X)$ }
12. ***else if*** ($Y \notin HC_{(K-1)}$) ***then***
13. { ***put*** Y ***into*** $HC_{(K-1)}$
14. $SC(Y) := SC(X)$ }}}}}}

Figure 7. DAHU

TABLE III. PARAMETER FOR SYNTHETIC DATASETS

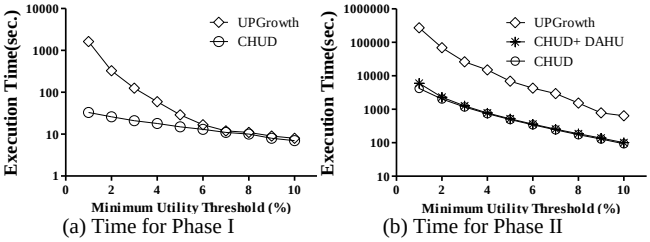
Parameter Descriptions		Default
D:	Total number of transactions	200K
T:	Average transaction length	12
N:	Number of distinct items	1,000
I:	Average size of maximal potential frequent itemsets	8

TABLE IV. 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
T10I8D200K	1,000	10	200K

TABLE V. NUMBER OF EXTRACTED PATTERNS FOR MUSHROOM

Minimum Utility Threshold (%)	CHUD		UPGrowth		Reduction Ratio
	Phase I	Phase II	Phase I	Phase II	
	# Candidates For CHUIs	# CHUIs	# Candidates For HUIs	#HUIs	#HUI / #CHUI
10	889	157	84,392	6,655	42.39
6	3,311	634	692,470	72,810	56.04
2	19,362	4,911	15,687,252	3,381,719	197.65
1	39,522	25,611	68,634,458	20,392,064	796.22



(a) Time for Phase I (b) Time for Phase II

Figure 8. Execution time on Mushroom

IV. EXPERIMENTS

In this section, we compare the performance of CHUD and DAHU with UPGrowth [20], which is to our best knowledge, the state-of-the-art method for high utility itemset mining. Although CHUD and UPGrowth produce different results, both of them consist of two phases. In Phase I, CHUD and UPGrowth respectively generate candidates for CHUIs and HUIs. In Phase II, CHUD and UPGrowth respectively identify CHUIs and HUIs from candidates produced in their Phase I. The combination of CHUD and DAHU is denoted as CHUD+DAHU, which first applies CHUD to find all closed$^+$ high utility itemsets and then uses DAHU to derive all high utility itemsets from the set of closed$^+$ high utility itemsets 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 the set of candidates and then uses CHUIs to derive all HUIs. Experiments were performed on a desktop computer with an Intel® Core 2 Quad Processor @ 2.66 GHz running Windows XP and 2 GB of RAM. CHUD and DAHU were implemented in Java. The implementation of UPGrowth was obtained from Tseng et al. [20], which is also implemented in Java. All memory measurements were done by using the Java API. Both synthetic and real datasets were used to evaluate the performance of the algorithms. A synthetic dataset T12I8D200K was generated by the IBM data generator [1]. The parameters of the data generator are described in Table III. Real datasets Mushroom and BMSWebView1 were obtained from FIMI Repository [32]. Foodmart is a real dataset obtained from the Microsoft foodmart 2000 database. Except the Foodmart dataset, the external and internal utility of each item are generated with the settings used in [20]. Foodmart already contains unit profits and purchase quantities of items. The total utility of Foodmart is 120,160.84. Table IV shows the characteristics of the above datasets. Mushroom is a real-life dense dataset, each transaction containing 23 items. Foodmart is a real-life sparse dataset from a retail store, with real utility values. BMSWebView1 is a real-life sparse dataset of click-stream data with a mix of short and long transactions (up to 267 items). T10I8D200K is a large sparse dataset with an average transaction length of 10.

A. Experiments on Mushroom Dataset

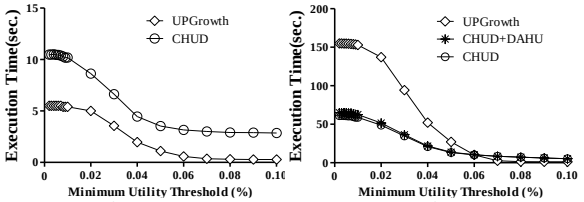
The first experiment consisted of running UPGrowth, CHUD, and DAHU on the Mushroom dataset, while varying *min_utility* from 10% to 1 %. The execution time of UPGrowth, CHUD, and CHUD+DAHU is shown in Figure 8 for Phase I and Phase II. Results show that CHUD outperforms UPGrowth for both phases, and the performance gap increased as *min_utility* was set lower. For example, when *min_utility* = 1%, CHUD is 50 times faster than UPGrowth for Phase 1 and 63 times faster for Phase II. Moreover, when CHUD is combined with DAHU to discover all high utility itemsets, the combination largely outperforms UPGrowth and was only slightly slower than CHUD. Table V shows the number of candidates and the number of results generated by UPGrowth, CHUD, and CHUD+DAHU. CHUD generates a much smaller number of candidates and results than UPGrowth. The smaller number of candidates generated by CHUD in Phase I is what makes CHUD perform better than UPGrowth in Phase II and for the total execution time (because Phase II is more costly than Phase I [20]). Lastly, we measured the reduction achieved by the representation of closed$^+$ high utility itemsets generated by CHUD compared to the set of all high utility itemsets generated by UPGrowth. As shown in Table V, a huge reduction is obtained (up to 796 times). Moreover, by running DAHU, it is possible to recover all high utility itemsets.

B. Experiments on Foodmart Dataset

The second experiment consists of running UPGrowth, CHUD and DAHU on the Foodmart dataset, while varying *min_utility* from 0.10% to 0.005 % of the total utility in the database. Execution times for Phase I and Phase II are shown in Figure 9. The total execution time of UPGrowth is less than CHUD, initially. But as the *min_utility* threshold became smaller, CHUD becomes faster (up to two times faster than UPGrowth). The reason why the performance gap between CHUD and UPGrowth 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 closed$^+$ high utility itemsets is less (still up to 34.6 times, as shown in Table VI).

TABLE VI. NUMBER OF EXTRACTED PATTERNS FOR FOODMART

Minimum Utility Threshold (%)	CHUD		UPGrowth		Reduction Ratio
	Phase I	Phase II	Phase I	Phase II	
	# Candidates For CHUIs	# CHUIs	# Candidates For HUIs	#HUIs	#HUI / #CHUI
0.1	581	258	1,585	258	1
0.05	1,444	1,076	37,158	6,266	5.8
0.01	6,332	6,293	230,165	209,387	33.273
0.005	6,657	6,656	233,032	230,617	34.647

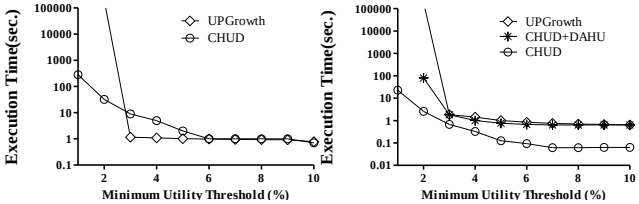


(a) Time for Phase I (b) Time for Phase II

Figure 9. Execution time on Foodmart

TABLE VII. NUMBER OF EXTRACTED PATTERNS FOR BMSWEBVIEW1

Minimum Utility Threshold (%)	CHUD		UPGrowth		Reduction Ratio
	Phase I	Phase II	Phase I	Phase II	
	# Candidates For CHUIs	# CHUIs	# Candidates For HUIs	#HUIs	#HUI / #CHUI
3	6	2	165	2	1
2.6	13	4	198	4	1
2.2	23	5	536	5	1
2	32	7	*	*	1

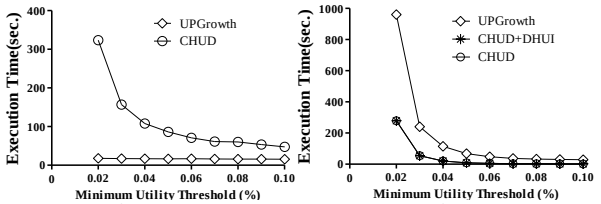


(a) Time for Phase I (b) Time for Phase II

Figure 10. Execution time on BMSWebView1

TABLE VIII. NUMBER OF EXTRACTED PATTERNS FOR T12I8D200K

Minimum Utility Threshold (%)	CHUD		UPGrowth		Reduction Ratio
	Phase I	Phase II	Phase I	Phase II	
	# Candidates For CHUIs	# CHUIs	# Candidates For HUIs	#HUIs	#HUI / #CHUI
0.1	28	8	488	8	1
0.05	165	47	1,297	47	1
0.03	1,093	169	4,811	169	1
0.02	4,776	782	16,444	782	1



(a) Time for Phase I (b) Time for Phase II

Figure 11. Execution time on T12I8D200K

Note that achieving a smaller reduction for sparse datasets is a well-known phenomenon in frequent closed itemset mining. A similar phenomenon occurs in closed$^+$ HUI mining. Besides, when DAHU was combined with CHUD, the execution time of CHUD+DAHU was up to two times faster than UPGrowth for low minimum utility thresholds and slightly slower than CHUD.

C. Experiments on BMSWebView1 Dataset

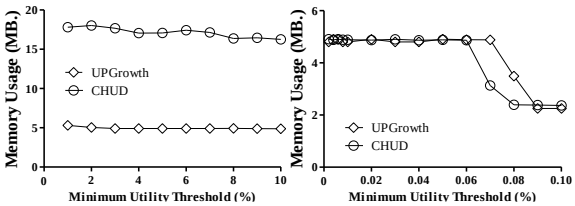
The third experiment consists of running UPGrowth, CHUD and CHUD+DAHU on BMSWebView1 while varying *min_utility* from 10% to 1 % of the total utility of the database. Results are presented in Figure 10 and Table VII. UPGrowth runs faster than CHUD and CHUD+DAHU for *min_utility* ≥ 3%. However, for *min_utility* < 3%, the performance of UPGrowth decreases sharply. For *min_utility* = 2%, UPGrowth 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 seven closed$^+$ HUIs 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 UPGrowth 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 proper 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 because it has to generate too many subsets.

D. Experiments on Sythetic Dataset

The fourth experiment is to run the algorithms on T12I8D200K with *min_utility* varying from 0.1% to 0.02% of the database total utility. Results are presented in Figure 11 and Table VIII. For this dataset, CHUD is faster than UPGrowth for the total execution time. Although the reduction on this synthetic dataset is not as good (since it produced the same result as UPGrowth), CHUD is faster because it generates about three times less candidates in Phase I. CHUD takes more times to generate candidates in Phase I. But the total execution time of CHUD is less than UPGrowth because Phase II is more costly than Phase I. CHUD+DAHU also outperforms UPGrowth, since DAHU only spend one second to derive all HUIs.

E. Memory Usage

During the previous experiments, we also measure the maximum memory usage of UPGrowth and CHUD. Results for Mushroom and Foodmart are presented in Figure 12 and are similar for the other datasets. In general, CHUD uses as much or slightly more memory than UPGrowth because the latter uses a compact tries-based data structure for representing the database that is more memory efficient than a vertical database. However, when the databases contain very long HUIs such as BMSWebView1, the number of candidates can be very large. In this case, the memory consumption of UPGrowth rises dramatically because it needs to create a number of conditional UPTrees that is proportional to the number of candidates.



(a) Mushroom (b) Foodmart

Figure 12. Memory usages for the algorithms in phase I

V. CONCLUSION

In this paper, we addressed the problem of redundancy in high utility itemset mining by proposing a compact representation of all high utility itemsets named closed+ high utility itemsets. To our knowledge, this is the first study on compact and lossless representation of high utility itemsets. To mine this representation, we proposed an efficient algorithm named CHUD. We propose three effective strategies named *REG*, *RML* and *DCM* to enhance the performance of CHUD. These strategies are novel since they have never been used for vertical mining of high utility itemsets. To efficiently recover all high utility itemsets from this representation, we proposed a top-down method named DAHU. Real and synthetic datasets having varied characteristics were used to perform a thorough performance evaluation. Results 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 UPGrowth, the current best algorithm by several orders of magnitude under low minimum utility thresholds (e.g. CHUD terminates in 80 seconds on BMSWebView1 for *min_utility* = 2%, while UPGrowth cannot terminate within 24 hours). The combination of CHUD and DAHU is also faster than UPGrowth when DAHU could be applied.

REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proc. of the 20th Int'l Conf. on Very Large Data Bases, pp. 487-499, 1994.

[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. In IEEE Transactions on Knowledge and Data Engineering, Vol. 21, Issue 12, pp. 1708-1721, 2009.

[3] J. -F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. In Data Mining and Knowledge Discovery, Vol. 7, Issue 1, pp. 5–22.

[4] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In Proc. of the Int'l Conf. on European Conference on Principles of Data Mining and Knowledge Discovery, pp. 74-85, 2002.

[5] C.-J. Chu, V. S. Tseng, T and Liang. An efficient algorithm for mining temporal high utility itemsets from data streams. In Journal of Systems and Software Vol. 81, Issure 7, pp. 1105-1117, 2008.

[6] C.-J. Chu, V. S. Tseng, and T. Liang. An Efficient Algorithm for Mining High utility Itemsets with Negative Values in Large Databases. In Applied Mathematics and Computation, Vol. 215, Issue. 2, pp. 767-778, 2009.

[7] R. Chan, Q. Yang, and Y. Shen. Mining high utility itemsets. In Proc. of IEEE Int'l Conf. on Data Mining, pp. 19-26, 2003.

[8] A. Erwin, R. P. Gopalan, and N. R. Achuthan. Efficient Mining of High utility Itemsets from Large Datasets. In Int'l Conf. on PAKDD, pp. 554-561, 2008.

[9] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In Proc. of IEEE Int'l Conf. on Data Mining, pp. 163-170, 2001.

[10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, pp. 1-12, 2000.

[11] 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. of IEEE Int'l Conf. on Data Mining, pp. 881-886, 2008.

[12] B. Le, H. Nguyen, T. A. Cao, and B. Vo. A Novel Algorithm for Mining High utility Itemsets. In Proc. of First Asian Conference on Intelligent Information and Database Systems, pp.13-17, 2009.

[13] Y. Liu, W. Liao, and A. Choudhary. A fast high utility itemsets mining algorithm. In Proc. of the Utility-Based Data Mining Workshop, pp. 90-99, 2005.

[14] C. Lucchese, S. Orlando and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets," In IEEE Transactions on Knowledge and Data Engineering, Vol. 18, Issue 1, pp. 21-36, 2006.

[15] Y.-C. Li, J.-S. Yeh, and C.-C. Chang. Isolated Items Discarding Strategy for Discovering High utility Itemsets. In Data & Knowledge Engineering, Vol. 64, Issue 1, pp. 198-217, 2008.

[16] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattice," In Journal of Information Systems, Vol 24, Issue 1, pp. 25–46, 1999.

[17] N. Pasquier, T. Bastide, R. Taouil, and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. In Proc. of Int'l Conf. on Database Theory, pp. 398–416, Israel, 1999.

[18] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Generating a Condensed representation for Association Rules, In Journal of Intelligent Information Systems, Vol 24, Issue 1, pp. 29–60, 2005.

[19] B.-E. Shie, V. S. Tseng, and P. S. Yu. Online Mining of Temporal Maximal Utility Itemsets from Data Streams. In Proc. of Annual ACM Symposium on Applied Computing, pp. 1622-1626, 2010.

[20] 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. of ACM SIGKDD Int'l Conf. on KDD, pp. 253–262, 2010.

[21] B. Vo, H. Nguyen, T. B. Ho, and B. Le. Parallel Method for Mining High utility Itemsets from Vertically Partitioned Distributed Databases. In Proc. of Int'l Conf. on Knowledge-based and Intelligent Information and Enginnering Systems, pp. 251-260, 2009.

[22] J. Wang, J. Han, and J. Pei. Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In Proc. of Int'l Conf. on ACM SIGKDD, pp. 236–245, 2003.

[23] U. Yun. Mining lossless closed frequent patterns with weight constraints. In Knowledge-Based Systems, Vol. 20, pp. 86–97, 2007.

[24] J.-S. Yeh, C.-Y. Chang, and Y.-T. Wang. Efficient Algorithms for Incremental Utility Mining. In Proc. of the 2nd Int'l Conf. on Ubiquitous information management and communication, pp. 212-217, 2008.

[25] H. Yao, H. J. Hamilton, L. Geng, A unified framework for utility-based measures for mining itemsets. In Proc. of ACM SIGKDD 2nd Workshop on Utility-Based Data Mining, pp. 28-37, 2006.

[26] S.-J. Yen and Y.-S. Lee. Mining High utility Quantitative Association Rules. In Proc. of Int'l Conf. on Data Warehousing and Knowledge Discovery, pp. 283-292, 2007.

[27] M. J. Zaki and C. J. Hsiao. Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. In IEEE Transactions on Knowledge and Data Engineering, Vol. 17, Issue 4, pp. 462–478, 2005.

[28] Frequent itemset mining implementations repository, http://fimi.cs.helsinki.fi/