Mining Compact High Utility Itemsets without Candidate Generation

Cheng-Wei Wu, Philippe Fournier-Viger, Jia-Yuan Gu, Vincent S. Tseng

Abstract Though the research topic of high utility itemset (HUI) mining has received extensive attention in recent years, current algorithms suffer from the crucial problem that too many HUIs tend to be produced. This seriously degrades the performance of HUI mining in terms of execution and memory efficiency. Moreover, it is very hard for users to discover meaningful information in a huge number of HUIs. In this paper, we address this issue by proposing a promising framework with a novel algorithm named *CHUI* (*Compact High Utility Itemset*)-*Mine* to discover closed$^+$ HUIs and maximal HUIs, which are compact representations of HUIs. The main merits of CHUI-Mine lie in two aspects: First, in terms of efficiency, unlike existing algorithms that tend to produce a large amount of candidates during the mining process, CHUI-Mine computes the utility of itemsets directly without generating candidates. Second, in terms of losslessness, unlike current algorithms that provide incomplete results, CHUI-Mine can discover the complete closed$^+$ or maximal HUIs with no miss. A comprehensive investigation is also presented to compare the relative advantages of different compact representations in terms of computational cost and compactness. To our best knowledge, this is the first work addressing the issue of mining compact high utility itemsets in terms of closed$^+$ and maximal HUIs without candidate generation. Experimental results show that CHUI-Mine achieves a massive reduction in the number of HUIs and is several orders of magnitude faster than benchmark algorithms.

Cheng-Wei Wu
National Ilan University, Ilan, Taiwan e-mail: wucw@niu.edu.tw

Philippe Fournier-Viger
Harbin Institute of Technology (Shenzhen), Shenzhen, China, e-mail: philfv8@yahoo.com

Jia-Yuan Gu
Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, e-mail: lester13.gu@gmail.com

Vincent S. Tseng
National Chiao Tung University, Hsinchu, Taiwan, e-mail: vtseng@cs.nctu.edu.tw

1 Introduction

Frequent Itemset Mining (*FIM*) [1, 10, 18] is a fundamental research topic with a wide range of applications. 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 real-life application, the traditional FIM may discover a large amount of frequent but low revenue itemsets. Consequently, the information on valuable itemsets with low selling frequencies will be lost. These problems are caused by the facts that 1) FIM treats all items as having the same importance (e.g., unit profit), and 2) the underlying assumption that each item in a transaction appears in a binary form (i.e., either present or absent in the transaction) ignores the quantity of an item in the transaction.

To address the above issues, *utility mining* [2, 5, 7, 11, 12, 20, 22, 24] has emerged as an important topic in the field of data mining. The main object of utility mining is to extract valuable information by considering profit, quantity, cost or other user preferences. *High utility itemset mining* is one of the most important tasks in utility mining, which aims at finding itemsets carrying with high utilities (e.g., high profits). An itemset is called *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. High utility itemset mining is an important task and has been applied to many applications [6, 7, 23] such as *cross marketing in chain stores* , *user behavior analysis* and *biomedical applications*.

Though high utility itemset mining is essential to many applications, it is not as easy as FIM due to the absence of *anti-monotone* property (also known as *downward closure property*). 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. To efficiently discover HUIs from databases, some algorithms were proposed, such as *Two-Phase* [12], *IHUP* [2], *IIDS* [14] *UP-Growth* [24] and *UP-Growth*$^+$ [20]. These algorithms use the concept of *transaction-weighted downward closure* property [12] to find HUIs and consist of two phases named *Phase I* and *Phase II*. In Phase I, they first generate candidate HUIs by overestimating the utility of itemsets. In Phase II, HUIs are identified from the set of candidate HUIs by scanning the original database once. Although these algorithms are pioneers for high utility itemset mining, they often generate a large amount of candidate HUIs during the mining process, which degrades their performance for low minimum utility thresholds. To address this issue, a novel algorithm named *HUI-Miner* [9] was recently proposed, which relies on a novel structure named *utility-list* to discover HUIs by using a single phase. The utility-list structure allows HUI-Miner to directly compute the utility of generated itemsets in main memory without re-scanning the original database. Besides, by using this structure, a tighter upper bound on the utility of supersets of an itemset can be obtained so that a large part of the search space can be effectively pruned.

Although existing algorithms perform well in some domains, they tend to produce a large number of HUIs for users. A large number of HUIs and candidates cause the mining tasks to suffer from long execution time and huge memory consumption.

With more itemsets and candidates being generated, the more resources are required by the algorithms. When the system resources are limited (memory, disk space or processing power), it is often impractical to generate the entire set of HUIs. In particular, the performance of these algorithms decreases sharply under low minimum utility thresholds. The situation becomes even worse when dealing with dense databases [19, 25], where items are highly correlated and often co-occur in many transactions. Hence, there are many long HUIs in dense databases. Moreover, a large amount of HUIs is also hard to be comprehended and analyzed by users.

In FIM, to reduce the computational cost of the mining task and present fewer but important patterns to users, many studies focused on developing concise representations of frequent itemsets, such as *free sets* [3], *non-derivable sets* [4], *maximal itemsets* [8] and *closed itemsets* [13, 15, 16, 17, 27, 28]. Although these representations successfully reduce the number of itemsets found, they were developed for FIM and are not applicable to high utility itemset mining.

Integrating these representations in HUI mining is not straightforward due to absence of the anti-monotonicity property. As indicated in [21, 26], integrating the concept of closed itemset mining into HUI mining results in a representation that is not lossless. To address this issue, [26] proposed a new compact and lossless representation of HUIs called *closed$^+$ high utility itemset* (*closed$^+$ HUI*). An itemset is called *closed high utility itemset* if its utility is no less than a user-specified *minimum utility threshold* and it has no proper supersets having the same frequency. Moreover, each closed HUI is annotated with a special structure called *utility unit array* such that the resulting itemset is called *closed$^+$ high utility itemset*. Utility unit array makes the set of closed$^+$ HUIs lossless because HUIs and their utilities can be derived from this set without re-scanning the original database. Besides, it was shown that the set of closed$^+$ HUIs can be several orders of magnitude smaller than the set of all HUIs, especially for dense databases and databases containing very long transactions. In [26], an efficient algorithm named *CHUD* is proposed to discover closed$^+$ HUIs in databases and an algorithm named *DAHU* is proposed to recover all HUIs from the set of closed$^+$ HUIs. Experimental studies have shown that CHUD outperforms UP-Growth by several orders of magnitude, especially for dense databases and databases containing very long transactions. The combination of CHUD with DAHU provides an alternative way to obtain all HUIs and it is much faster than UP-Growth for the task of mining all HUIs.

However, the CHUD algorithm has not been compared with the most recent high utility itemset mining algorithms. Thus, the question "*Is it still faster to discover closed$^+$ HUIs than HUIs?*" is unanswered. Besides, another critical problem with the CHUD algorithm is that it relies on the two-phase model and overestimates too many low utility itemsets as candidate HUIs, which degrades its overall performance in terms of execution time and memory usage.

Another critical problem is that the set of closed$^+$ HUIs may still be too large for some real-life datasets. In [19], an alternative compact representation of HUIs called *maximal high utility itemset* (*maximal HUI*) was proposed, which serves as a subset of closed$^+$ HUIs. Mining maximal HUIs is thus a promising approach to reduce the size of the result set when there exist too many closed$^+$ HUIs. An algorithm

has been proposed to discover maximal HUIs, but it is an algorithm for data stream rather than transactional databases. Besides, it is an approximate algorithm that does not provide users with the complete set of maximal HUIs and thus some valuable information may be omitted. It is therefore an open issue to develop algorithms for efficiently discovering maximal HUIs in transactional databases with no miss. Moreover, another issue is that no comparison as yet been done to evaluate these compact representations in terms of computational cost and reduction ratio.

Considering all issues as described above, an interesting question raised is: "*Would it be possible to define a unifying framework to discover both maximal and closed*$^+$*HUIs efficiently?*" Addressing this issue is a non-trivial task with several challenges. First, designing a common algorithm to mine both compact representations of HUIs requires delicate integration of techniques from closed itemset and maximal itemset mining [8, 13, 15, 16, 17] with those for HUI mining so that the complete set of closed$^+$ and maximal HUIs can be captured with no miss. Second, the resulting algorithm may be slower or less memory efficient than benchmark algorithms for mining HUIs or closed$^+$ HUIs. Therefore, the algorithm should have the ability to discover patterns in one phase to avoid the problem of producing too many candidates of two-phase algorithms. This would require designing a special structure that allows computing the utility of closed$^+$ and maximal HUIs in memory without producing candidates and without performing costly database scans to calculate the utilities or utility unit arrays.

In this paper, we address all of the challenges mentioned above by proposing a new framework for *mining compact representations of HUIs without candidate generation*. To our best knowledge, this topic has not been explored so far. The contributions of this work are summarized as follows:

- Though some studies have compared the properties of frequent closed itemset and frequent maximal itemset, no comparison as yet been done to evaluate closed$^+$ HUI and maximal HUI in terms of computational cost and reduction ratio. This study is the first to investigate respective pros and cons of these representations and algorithms for mining these representations.
- A novel algorithm named *CHUI*(*Compact High Utility Itemset*)*-Mine* is proposed to mine the complete set of closed$^+$ HUIs (the variation is called *CHUI-Mine*(*Closed*)) or maximal HUIs (the variation is called *CHUI-Mine*(*Maximal*)). Both of them are one-phase algorithms and discover respective representations without producing candidates.
- CHUI-Mine(Closed) is the first algorithm that finds the complete set of closed$^+$ HUIs in the databases without producing candidates. Comparing CHUD (current best algorithm for mining closed$^+$ HUIs) with CHUI-Mine(Closed), CHUD performs two phases and uses *TWDC* property to prune the search space, whereas CHUI-Mine(Closed) performs just one phase and uses the proposed *PUDC* (*Pivot Utility Downward Closure*) property to prune the search space. This later property is more powerful in pruning.
- CHUI-Mine(Maximal) is the first algorithm that discovers the complete set of maximal HUIs in the databases with no miss. CHUI-Mine(Maximal) can provide a very compact summarization of HUIs to users and serve as a good alterna-

tive solution when other algorithms (e.g., CHUD and HUI-Miner) fail to mine (closed$^+$) HUIs.
- We further propose an efficient algorithm called *RHUI* (*Recover all High Utility Itemsets from maximal patterns*) to efficiently recover all HUIs and their exact utilities from the set of maximal HUIs. The combination of CHUI-Mine(Maximal) and RHUI constitutes an alternative solution to discover all HUIs and is even faster than the current best algorithms CHUD and HUI-Miner.
- We perform an extensive experimental study on several real datasets to evaluate the performance of the proposed algorithms. Results show that CHUI-Mine outperforms the current best algorithms for mining (closed$^+$) HUIs substantially, especially for dense datasets and low minimum utility thresholds. Besides, CHUI-Mine can be more than three orders of magnitude faster than the benchmark algorithms

The remaining of this paper is organized as follows. Section 2 introduces related definitions and formally defines the problem statement. Section 3 presents the related work. Section 4 describes the proposed structures and methods in detail. Performance evaluation is presented in Section 5. Section 6 draws the conclusion.

2 Background

Given a finite set of distinct *items* $I^* = \{I_1, I_2, \ldots, I_N\}$. Each item $I_i \in I^*$ is associated with a positive number $P(I_i, D)$ (e.g. unit profit), called its *external utility*. A *transactional database* $D = \{T_1, T_2, \ldots, T_M\}$ is a set of transactions, where each transaction $T_r \in D$ $(1 \leq r \leq M)$ is a subset of I^* and has an unique identifier r, called *Tid*. In a transaction T_r, each item $I_i \in I^*$ is associated with a positive number $q(I_i, T_r)$, called its *internal utility* in T_r (e.g. purchase quantity). An itemset $X = \{I_1, I_2, \ldots, I_k\}$ is a set of k distinct items, where $I_i \in I^*$ $(1 \leq i \leq k)$ and k is the length of X. A k-itemset is an itemset of length k. An itemset X is *contained* in a transaction T_r if $X \subseteq T_r$.

Let Table 2 be an example database containing five transactions. Each row in Table 2 represents a transaction, in which each letter represents an item and has a internal utility. The external utility of each item is shown in Table 1.

Table 1: Unit Profits of Items

Item	A	B	C	D	E	F
Unit Profit	4	3	1	2	1	4

Definition 1. Tidset and Support count. The *Tidset* of X is a set of Tids of all transactions containing X and denoted as $TidSet(X) = \{r \mid X \subseteq T_r\}$. The *support count* of an itemset X is defined as $SC(X) = |TidSet(X)|$.

Table 2: An Example Database

TID	Transaction	TU
T_1	A(3), B(3), C(2), E(3)	26
T_2	B(2), D(7)	20
T_3	A(2), C(5), D(5), E(2)	25
T_4	A(1), C(6), D(5), E(1), F(2)	29

Property 1. For a k-itemset $X = \{I_1, I_2, \ldots, I_k\}$, $SC(X) = |TidSet(I_1) \cap TidSet(I_2) \cap \ldots \cap TidSet(I_k)|$.

Property 2. Let an itemset Y be a proper superset of an itemset X. Then, $TidSet(Y) \subseteq TidSet(X)$.

Definition 2. Utility of an item in a transaction. The *utility of an item* $I_i \in I^*$ *in a transaction* T_r is denoted as $u(I_i, T_r)$ and defined as $p(I_i, D) \times q(I_i, T_r)$. For example, the $u(\{A\}, T_1) = p(\{A\}, D) \times q(\{A\}, T_1) = 4 \times 3 = 12$.

Definition 3. Transaction utility. The *transaction utility* (abbreviated as *TU*) of a transaction T_r is defined as $TU(T_r) = \sum_{I_i \in T_r} u(I_i, T_r)$. For example, $TU(T_1) = u(\{ABCE\}, T_1) = 26$.

Definition 4. Total utility. The *total utility of a database D* is denoted as $TotalU_D$ and defined as $\sum_{T_r \in D} TU(T_r)$.

Definition 5. Utility and relative utility of an itemset. The *utility of an itemset X* in a transaction T_r is defined as $u(X, T_r) = \sum_{I_i \in X} u(I_i, T_r)$. The utility of X in D is defined as $u(X) = \sum_{r \in TidSet(X)} u(X, T_r)$. The *relative utility of X* is defined as $ru(X) = u(X) / TotalU_D$.

Definition 6. High utility itemset. An itemset X is called *high utility itemset* (abbreviated as *HUI*) if $u(X)$ is no less than a user-specified *minimum utility threshold* min_util ($0 < min\_util \leq TotalU_D$). Otherwise, X is *low utility*. Let *relative minimum utility threshold* $rmin\_util = min\_util / TotalU_D$, an equivalent definition is that X is high utility if $ru(X) \geq rmin\_util$. The set of HUIs in D is denoted as H.

For example, if $min\_util = 30$, the set of HUIs in Table 2 is H = {{D}:34, {AC}:37, {AD}:32, {AE}:30, {CD}:31, {ACD}:43, {ACE}:43, {CDE}:34, {ADE}: 35, {ACDE}:46}, in which the number beside each itemset is its utility.

Definition 7. Closed itemset. An itemset X is called *closed itemset* if it has no proper superset Y in D such that $SC(X) = SC(Y)$. The complete set of closed itemsets is denoted as C.

For example, the complete set of closed itemsets in Table 2 is C = {{B}:2, {D}:3, {BD}:1, {ACE}:3, {ABCE}:1, {ACED}:2, {ACDEF}:1}, in which the number beside each itemset is its support count.

Definition 8. Closure of an itemset. Let Y be the superset of an itemset X. Y is called the *closure* of X iff Y is closed and $SC(Y) = SC(X)$. The closure of X is defined as $closure(X) = \cap_{r \in TidSet(X)} T_r$. For example, $closure(\{AC\}) = T_1 \cap T_3 \cap T_4 = \{ABCE\} \cap \{ACDE\} \cap \{ACDEF\} = \{ACE\}$.

Property 3. For any itemset X, $SC(X) = SC(closure(X))$. Equivalently, $SC(X) = max\{SC(Y) \mid Y \in C \text{ and } X \subseteq Y\}$. For example, $SC(\{AC\}) = max\{SC(\{ACE\}), SC(\{ABCE\}), SC(\{ACED\}), SC(\{ACDEF\})\} = 3$.

Definition 9. Closed high utility itemset. An itemset X is called *closed high utility itemset* (abbreviated as *closed HUI*) if X is closed and $u(X) \geq min\_util$. The complete set of closed HUIs is defined as $CH = H \cap C$.

Definition 10. Utility unit array. The *utility unit array* of a k-itemset $X = \{I_1, I_2, \ldots, I_k\}$ is denoted as $V(X) = [v_1, v_2, \ldots, v_k]$, where the i-th utility value v_i is defined as $V(X, I_i) = \sum_{r \in TidSet(X) \wedge I_i \in T_r} u(I_i, T_r)$.Besides, $u(X) = \sum_{i=1}^{k} V(X, I_i)$.

For example, the second utility value of the utility unit array of {ACE} is $V(\{ACE\},\{C\}) = u(\{C\}, T_1) + u(\{C\}, T_3) + u(\{C\}, T_4) = 2 + 5 + 6 = 13$. The utility unit array of {ACE} is $V(\{ACE\}) = [24, 13, 6]$. The utility of {ACE} is $V(\{ACE\}, \{A\}) + V(\{ACE\}, \{C\}) + V(\{ACE\}, \{E\}) = 43$.

Definition 11. Closed$^+$ high utility itemset. A closed high utility itemset X is called *closed$^+$ high utility itemset* (abbreviated as *closed$^+$ HUI*) if X is attached with its utility unit array. The complete set of closed$^+$ HUI is denoted as CH^+.

For example, if $min\_util = 30$, the complete set of closed$^+$ HUIs in the database of Table 2 is $CH^+ = \{(\{D\}, [34]), (\{ACE\}, [24, 13, 6]), \{ACDE\}, [12, 11, 20, 3]\}$.

Property 4. $\forall X \in H, \exists Y \in CH^+$ such that $Y = closure(X)$ and $u(Y) \geq u(X)$.

Rationale. $\forall X$ in D, $\exists Y \in C$ such that $Y = closure(X)$ and $SC(X) = SC(Y)$. Since $X \in H$ and $u(X) \geq min\_util$, thus $SC(X) = SC(Y)$ and $X \subseteq Y$ will yield $u(Y) \geq u(X) \geq min\_util$ by Definition 6.

Property 5. For any itemset X, if $closure(X) \notin CH$, X is a low utility itemset.

Rationale. Let $Y = closure(X)$. If $Y \notin CH$, $u(Y) < min\_util$. Since $SC(X) = SC(Y)$ and $X \subseteq Y$, by Definition 6, we have $u(X) \leq u(Y) < min\_util$.

Property 6. The complete set of closed$^+$ HUIs is a *lossless representation* of all HUIs. For any k-itemset $X = \{I_1, I_2, \ldots, I_k\}$, if $closure(X) \in CH^+$, $u(X)$ can be calculated as $\sum_{i=1}^{k} V(closure(X), I_i)$ by using the utility unit array of its closure without accessing the original database.

Problem Statement for Closed$^+$ High Utility Itemset Mining. Given a database D with internal and external utility of items, and a user-specified minimum utility threshold *min_util*, the problem statement is to discover all closed itemsets having a utility no less than *min_util* and their utility unit arrays.

Definition 12. Maximal high utility itemse. An itemset X is call *maximal high utility itemset* (abbreviated as *maximal HUI*) iff $u(X) \geq min\_util$ and there is no HUI that is a proper superset of X. The complete set of maximal HUIs in the database is denoted as *MH.*

For example, if $min\_util = 30$, the complete set of maximal HUIs in the database of Table 2 is $MH = \{\{ACDE\}\}$.

Property 7. $MH \subseteq CH \subseteq H$.

Rationale. This property is directly obtained by the Definitions 6, 9 and 12.

Problem Statement for Maximal High Utility Itemset Mining. Given a database D with internal and external utility of items, and a user-specified minimum utility threshold *min_util*, the problem statement is to discover the complete set of maximal HUIs in D.

Then we compare closed$^+$ HUI with maximal HUI from different perspectives. In terms of *compactness*, the set *MH* is smaller than *CH* because *CH* is a subset of *MH* (Property 6). In terms of *interpretability*, the interpretation of maximal HUI is less obvious. A maximal itemset only tells the user that some of its subsets may be HUIs. But closed$^+$ HUIs are more meaningful as they are lossless and only remove redundancy [21, 26]. In terms of *recoverability*, both representations can be used to recover all HUIs, but it is more expensive with maximal HUIs. The reason is that maximal HUIs are not lossless. Therefore querying the utility of a HUI using maximal HUIs requires to perform an additional database scan, while it is not required for closed$^+$ HUIs. Both *MH* and *CH* are *representative* because both of them can be used to find all HUIs.

3 Related Work

High utility itemset mining is one of the most important topics in data mining. Many efficient algorithms, such as *Two-Phase* [12], *IHUP* [2], *IIDS* [14], *UP-Growth* [24] *and UP-Growth* [20], have been proposed for mining HUIs. However, these algorithms are two-phase methods and often produce too many candidates during the mining process. Recently, a one-phase algorithm named *HUI-Miner* [9] was proposed. It discovers HUIs without producing candidates. Though the above algorithms perform well in some cases, their performance quickly degrades when the database contains many long HUIs (e.g., dense databases). A large number of HUIs and candidates cause these algorithms to suffer from very long execution time and huge memory consumption. Besides, it is hard for users to discover meaningful information in a large number of HUIs.

To reduce the number of discovered patterns and present users with more representative patterns, a promising solution is to mine compact representations of HUIs. Though this topic is very important, it has not been deeply explored and only very

preliminary work have been done on this topic. Chan et al. introduced the concept of *utility frequent closed patterns* [5]. But, it is based on a definition of HUI that is different from [2, 9, 12, 14, 20, 24] and our work. Shie et al. define a compact representation of HUIs called *maximal high utility itemset* and proposed an approximate algorithm named *GUIDE* for mining this representation. Wu et al. proposed the concept of *closed*$^+$ *high utility itemsets* (*closed*$^+$ *HUI*) and an algorithm named *CHUD* for mining *closed*$^+$ *HUIs*.

Although the above studies are pioneers for compact representation of HUIs, they have one or more of the following deficiencies: (1) No systematic comparison as yet been done to compare these two compact representations. (2) The CHUD algorithm has not been compared with the current best method for mining HUIs. Hence, the answer to the question "*Is it still faster to discover compact HUIs than HUIs?*" is unknown. (3) GUIDE is an approximate algorithm and thus it cannot capture the complete set of maximal HUIs in the databases. (4) No algorithm has been proposed for efficiently deriving all HUIs from maximal HUIs. (5) GUIDE and CHUD are two-phase algorithms so that they may produce too many candidates during the mining process, which degrades their mining performance.

In the next section, we address the above issues by proposing a one-phase algorithm named *CHUI*(*Compact High Utility Itemset*)-*Mine*, which adopts a special structure named *EU-List* (*Extended Utility-List*) to mine the complete set of closed$^+$ HUIs (the variation is called *CHUI-Mine*(*Closed*)) or maximal HUIs (the variation is called *CHUI-Mine*(*Maximal*)) in the databases without generating candidates. An efficient algorithm named *RHUI* (*Recover all High Utility Itemsets from maximal patterns*) is also proposed for recovering all HUIs from maximal HUIs.

4 The Proposed Methods

In this section, we first introduce the *EU-List* (*Extended Utility-List*) structure and then describe the *CHUI-Mine*(*Closed*) and *CHUI-Mine*(*Maximal*) algorithms in details.

4.1 Construction of EU-List

In the proposed algorithms, the utility information of itemsets in transactions are maintained in a special structure named *EU-List*. For example, Fig. 1 shows *EU-Lists* of items. The construction of *EU-List* can be performed with two database scans. In the first database scan, the transaction utility of each transaction (Definition 3) and *TWU* of items (Definition 13) [12] are calculated.

Definition 13. TWU of an itemset. The *TWU* (*Transaction-Weighted Utilization*) of an itemset X is defined as $TWU(X) = \sum_{r \in TidSet(X)} TU(T_r)$. For example, $TWU(\{A\}) = TU(T_1) + TU(T_3) + TU(T_4) = 80$. Table 3 shows the *TWU* of items.

Table 3: TWU values of items

Item	A	B	C	D	E	F
TWU	80	80	80	74	46	29

Definition 14. (TWDC property. The *TWDC* (*Transaction-Weighted Utilization Downward Closure*) property [2, 12] states that for any itemset X if $TWU(X) < min\_util$, all supersets of X are low utility.

After scanning the database once, *TWU* of items are obtained. By the *TWDC* property, if *TWU* of an item is less than *min_util*, all its supersets are unpromising to be high uutility. These items are called *unpromising items*.

Definition 15. Promising and unpromising items. An item $I_i \in I^*$ $(1 \leq i \leq N)$ is called *promising item* iff $TWU(I_i) \geq min\_util$. Otherwise, the item is called *unpromising item*.

During the second database scan, when a transaction T_j $(1 \leq j \leq |D|)$ is retrieved, unpromising items are removed from T_j and their utilities are eliminated from the transaction utility of T_j since only supersets of promising items can be high utility. The promising items in T_j are sorted in TWU descending order. Note that other orders can be used.

A transaction resulting from the above process is called a *reorganized transaction.* For example, if *min_util* = 30, Table 4 shows reorganized transactions in the database of Table 2, where the number beside each item is the multiplication of its internal and external utilities. After scanning the database twice, the *EU-List* of each promising item is constructed.

Table 4: Reorganized Transactions

TID	ReorganizedTransaction
T_1	A[12], C[2], E[3], B[9]
T_2	D[14], B[6]
T_3	A[8], C[5], E[2], D[10]
T_4	A[4], C[6], E[1], D[10]

In the proposed methods, each item(set) is associated with an *EU-List*, support count of X, utility unit array of X, and two ordered sets named *PrevSet*(X) and *PostSet*(X). The *EU-List* of X consists of several *tuples*. Each tuple in the *EU-List* of X represents the utility information of X in the reorganized transaction T_r and has three fields: *Tid*, *EU* and *PU*. Fields *Tid* and *EU* respectively indicate the identifier of T_r and the exact utility of X in T_r. Field *PU* indicates the *pivot-based remaining utility of X in* T_r. The concept of *Pivot-based remaining Utility* (abbreivated as *PU*) is based on the following definitions.

Definition 16. Definition 16 (Precede and succeed). Given a set of items $\{I_1, I_2, \ldots, I_N\}$ and a total order R: $I_1 \prec I_2 \prec \ldots \prec I_N$ among items, the item I_i *precedes* the item I_j iff $I_i \prec I_j$ $(1 \leq i < j \leq N)$. Otherwise, I_i *succeeds* I_j. In our methods, items in itemsets are sorted according to R.

Definition 17. Definition 17 (Pivot). Given a total order R and a k-itemset $X = \{I_1, I_2, \ldots, I_k\}$, where $I_1 \prec I_2 \prec \ldots \prec I_k$. The item I_1 is called *pivot* of X and denoted as P_X.

Definition 18. Definition 18 (Succeeding superset). An itemset Y is called *succeeding superset* of the itemset X iff $X \subset Y$ and $P_X = P_Y$. For example, {CEDB} is a succeeding superset of {CDB}, but {ACDB} is not.

Definition 19. Definition 19 (PU of an item in a transaction). The *PU of an item* $I_i \in I^*$ *in a transaction* T_r is defined as $pu(I_i, T_r) = \sum_{I_j \in T_r \wedge I_j \succ I_j \wedge TWU(I_j \geq \min\_util} u(I_j, T_r)$. For example, $pu(\mathrm{C}, T_3) = u(\mathrm{E}, T_3) + u(\mathrm{D}, T_3) = 2 + 10 = 12$.

Definition 20. Definition 20 (PU of an itemset in a transaction). The *PU of a k-itemset* X $(k \geq 2)$ *in a transaction* T_r is defined as $pu(X, T_r) = pu(P_X, T_r) \sum_{I_j \in X \wedge I_j \succ P_x} u(I_j, T_r)$. For example, $pu(\{\mathrm{CD}\}, T_3) = pu(\{\mathrm{C}\}, T_3)) - u(\{\mathrm{D}\}, T_3) = 12 - 10 = 2$.

Definition 21. Definition 21 (PU of an itemset in a database). The *PU of a k-itemset* X $(k \geq 2)$ *in a database* D is denoted and defined as $pu(X) = \sum_{r \in TidSet(X)} pu(X, T_r)$. For example, $pu(\{\mathrm{CD}\}) = pu(\{\mathrm{CD}\}, T_3) + pu(\{\mathrm{CD}\}, T_4) = 2 + 1 = 3$.

Definition 22. Definition 22 (EU-List structure). The *EU-List of an itemset* X is denoted as $EL(X)$. An element *associated with* T_r in $EL(X)$ is denoted as $EL(X, T_r)$. The EU and PU of the element are denoted as $EL(X, T_r).EU$ and $EL(X, T_r).PU$, respectively.

For example, Fig. 1 shows the *EU-Lists* of promising items. The sums of EU and PU values in $EL(X)$ are denoted as $SumEU(X)$ and $SumPU(X)$, respectively.

{A}		
Tid	EU	PU
1	12	14
3	8	17
4	4	17

{C}		
Tid	EU	PU
1	2	12
3	5	12
4	6	11

{E}		
Tid	EU	PU
1	3	9
3	2	10
4	1	10

{D}		
Tid	EU	PU
2	14	6
3	10	0
4	10	0

{B}		
Tid	EU	PU
1	9	0
2	6	0

Fig. 1: EU-Lists of promising items

Theorem 1. ***PUDC Property.*** *The pivot utility downward closure (abbreviated as PUDC) property states that* $\forall X$ *if* $(u(X) + pu(X)) < min\_util$, *all the succeeding supersets of X are low utility.*

Proof. Let Y be a succeeding superset of X and $Z = Y - X$, $\forall I_j \in Z$, then we have $P_X \prec I_j$. Besides, $u(Z, T_r) \leq pu(X, T_r)$, $\forall r \in TidSet(Y)$. Because $TidSet(Y) \subseteq TidSet(X)$ and $u(Y, T_r) = u(X, T_r) + u(Z, T_r) \leq u(X, T_r) + pu(X, T_r)$, we have $u(Y) = \sum_{r \in TidSet(Y)} u(Y, T_r) \leq \sum_{r \in TidSet(Y)} [u(X, T_r) + pu(X, T_r)] \leq \sum_{s \in TidSet(X)} [u(X, T_s) + pu(X, T_s)] = u(X) + pu(X)$. Therefore, if $(u(X) + pu(X)) < min\_util$, Y is low utility.

For example, if $min\_util = 40$, the succeeding supersets of {CD} are low utility because $u(\{CD\}) + pu(\{CD\}) = 31 + 3 = 34$ is smaller than *min_util.*

Theorem 2. *For any itemset X, (u(X) + pu(X)) ≤ TWU(X).*

Proof. For any transaction T_r containing X, $(u(X, T_r) + pu(X, T_r)) \leq TU(T_r)$ can be directly obtained from Definition 3 and 20. Hence, $(u(X) + pu(X)) = \sum_{r \in TidSet(X)} u(X, T_r) + \sum_{r \in TidSet(X)} pu(X, T_r) \leq \sum_{r \in TidSet(X)} TU(T_r) = TWU(X)$.

4.2 The CHUI-Mine(Closed) Algorithm

The pseudo code of CHUI-Mine(Closed) is shown in Fig. 2. The algorithm has two input parameters: (1) a database D and (2) a user-specified minimum utility threshold *min_util.* It outputs the complete set of closed$^+$ HUIs in D.

The framework of the *CHUI-Mine* algorithm consists of two main parts: (1) construction of *EU-Lists* of promising items and (2) generation of closed$^+$ HUIs by using *EU-Lists*. In line 3 of Fig. 1, the procedure *GEN-CHUI*(Ø, P, Ø, *min_util*) is called to generate closed$^+$ HUIs by using *EU-Lists* of promising items, where P is the set of all promising items. The *GEN-CHUI* procedure is an extension of an efficient algorithm named *DCI-Closed* [13] for mining closed itemsets. For each closed itemset X discovered by the *DCI-Closed*, we construct its *EU-List* to calculate its utility to determine whether it is a closed$^+$ HUI and use the PUDC property to prune the search space (Theorem 1). The procedure has four input parameters: (1) an itemset X, (2) *PrevSet*(X), (3) *PostSet*(X) and (4) the minimum utility threshold *min_util*. It explores the search space of closed$^+$ HUIs that are succeeding supersets of X by appending items from *PostSet*(X) to X. The pseudo code of GEN-CHUI is shown in Fig. 3, which is performed as follows. While *PostSet*(X) is not empty, the procedure selects the smallest item I in *PostSet*(X) to create an itemset $Y = X$ (I and removes I from *PostSet*(X) (line 1 to 4 of Fig. 3). Then, *TidSet*(Y) is obtained by intersecting *TidSet*(X) and *TidSet*(I) and the support count of Y is set to $|TidSet(Y)|$. Next, the *EU-List* of Y is constructed by the following process (line 5 of Fig. 3). For each transaction $T_r \in TidSet(X) \cap TidSet(I)$, the element $EL(Y, T_r)$ is created in $EL(Y)$, where $EL(Y, T_r).EU$ is set to the sum of $EL(X, T_r).EU$ and $EL(I, T_r).EU$, and where $EL(Y, T_r).PU$ is set to $EL(X, T_r).PU - EL(I, T_r).EU$. Finally, *PrevSet*($Y$) is initialized to *PrevSet*(X). If the sum of *SumEU*(Y) and *SumPU*(Y) is less than *min_util*, all succeeding supersets of Y are low utility (Theorem 1). Otherwise, the algorithm calls the procedure *IsSubsumeCheck*(Y, *PreSet*(Y)) to check whether Y *is subsumed* by other itemsets.

Algorithm: CHUI-Mine(Closed)

Input: *D*, *min_util*;
Output: The complete set of closed$^+$ high utility itemsets;
01. ***Scan*** *D* twice to find promising items and construct *EU-Lists* of promising items;
02. ***GEN-CHUI***(Ø, *P*, Ø, *min_util*);

Fig. 2: The CHUI-Mine(Closed) algorithm

Procedure: GEN-CHUI

Input: An itemset *X*, *PostSet*(*X*), *PreSet*(*X*), *min_util*;
Output: The complete set of closed$^+$ high utility itemsets;

```
01.  While PostSet(X) ≠ Ø;
02.      I ← min≺(PostSet(X));
03.      Remove I from PostSet(X);
04.      Y ← X ∪ {I};
05.      Construct EU-List of Y;
06.      PrevSet(Y) ← PrevSet(X);
07.      If(SumEU(Y) + SumPU(Y)) ≥ min_util
08.          If(IsSubsumedCheck(Y, PrevSet(Y)) = false)
09.              EL(Y_C) ←EL(Y);
10.              PrevSet(Y_C) ← PrevSet(Y);
11.              PostSet(Y_C) ← Ø;
12.              Y_C ← ClosureComputation(Y, PostSet(Y_C), PostSet(X));
13.              V(Y_C) ← UpdateUtilityUnitArray(Y_C, V(Y_C));
14.              If(SumEU(Y_C)) ≥ min_util
15.                  Output Y_C, SC(Y_C) and V(Y_C);
16.              GEN-CHUI(Y_C, PostSet(Y_C), PrevSet(Y_C), min_util);
17.              PrevSet(X) ←PrevSet(X) ∪ I;
```

Fig. 3: The GEN-CHUI procedure

Definition 23. Subsume. An itemset *Y is subsumed* by an itemset *S* if $Y \subset S$ and $SC(Y) = SC(S)$.

Property 8. Given an itemset *Y* and an item $J \in I^*$ $(1 \leq i \leq N)$, $TidSet(X) \subseteq TidSet(J) \Leftrightarrow J \in closure(Y)$.

Property 9. Given two itemsets *Y* and *S*, if $Y \subset S$ and $SC(Y) = SC(S)$ then $closure(Y) = closure(S)$.

By Definition 23, if we can find an already mined closed$^+$ HUI *S* that subsumes *Y*, we can conclude that *Y* is not closed and $closure(S) = closure(Y)$. Hence, we can safely prune the itemset *Y* and stop exploring the search space of succeeding supersets of *Y*. Otherwise, the procedure returns true.

The pseudo code of the *IsSubsumedCheck* procedure is shown in Fig. 4. The procedure has two input parameters: (1) an itemset Y and (2) $PrevSet(Y)$ and it is performed as follows. For each item J in $PrevSet(Y)$, if $TidSet(Y)$ is contained in $TidSet(J)$, the procedure returns *true* to indicate that Y is subsumed by an already mined closed$^+$ HUI and thus that Y is not a closed$^+$ HUI (Property 7 and 8). If $TidSet(Y)$ is not contained in the *TidSet* of any item in $PrevSet(Y)$, the procedure returns *false* to indicate that the closure of Y is closed. If Y passes the "*IsSubsumed*" check, the procedure *ClosureComputation*(Y, $PostSet(Y)$, $PostSet(X)$) is called to compute the closure of Y and construct its *EU-List* by updating the *EU-List* of Y. The pseudo code of the *ClosureComputation* procedure is shown in Fig. 5, and it is performed as follows. Initially, a variable Y_C for storing the closure of Y is set to Ø and $EL(Y_C)$ is set to $EL(Y)$. Then, for each item Z in $PostSet(X)$, we check if $TidSet(Y)$ is contained in $Tidset(Z)$. If it is not contained in $Tidset(Z)$, Z is added to $PostSet(Y)$. Otherwise, Z is added to Y_C because Z is contained in the closure of Y by Property 7.

Procedure: IsSubsumedCheck

Input: An itemset *Y*, *PrevSet*(*Y*);
Result: Return *true* if *Y* is subsumed by already mined closed$^+$ high utility itemsets. Otherwise, return *false*.

```
01. For each item J∈PrevSet(Y) do
02.     If(TidSet(Y) ⊆ TidSet(J))
03.         Return true;
04.     Return false;
```

Fig. 4: The IsSubsumedCheck procedure

The *EU-List* of Y_C is updated by the following process. For each Tid $r \in TidSet(Y_C) \cap TidSet(Z)$, the $EL(Y_C, T_r)$ is updated to $<r, v_1, v_2>$, where v_1 is $EL(Y_C, T_r).EU + EL(Z, T_r).EU$ and v_2 is $EL(Y_C, T_r).PU - EL(Z, T_r).EU$. After processing all the items in $PostSet(X)$, the *EU-List* of Y_C is updated and Y_C captures the closure of *Y*. Then, the procedure returns *Y*'s closure Y_C.

After calling the *ClosureComputation* procedure, the *UpdateUtilityArray*(Y_C) procedure is then called to calculate the utility unit array of Y_C. If $Y_C = \{I_1, I_2, ..., I_k\}$, the utility unit array $V(Y_C)$ of Y_C is calculated as follows. For each item $I_i \in Y_C$, we calculate $V(Y_C, I_i)$ (Definition 10) by scanning $UL(I_i)$ once to sum up the utilities of I_i in all elements whose Tids are in $Tidset(Y_C)$. Note that in our implementation for any item $I \in Y_C$ such that $I \notin Y$, the utility value $V(I, Y_C)$ is calculated during the closure computation by the *ClosureComputation* procedure. For the sake of simplicity, this optimization is not shown in the pseudo code of the *ClosureComputation* procedure due to the page limitation.

Then the algorithm outputs Y_C if $SumEU(Y_C)$ is no less than *min_util* because Y_C meets the criteria of being a closed itemset, a high utility itemset, and is associated

Procedure: ClosureComputation

Input: An itemset Y, $PostSet(Y_C)$, $PostSet(X)$;
Result: Return Y's closure Y_C. Update $PostSet(Y_C)$ and *EU-List* of Y_C;

```
01. Y_C ← Y;
02. PostSet(Y_C) ← Ø;
03. For each item Z∈PostSet(X) do
04.     If(TidSet(Y_C) ⊆ TidSet(Z))
05.         Y_C ← Y_C ∪ Z;
06.         For each Tid r∈TidSet(Y_C) do
07.             EL(Y_C, T_r).EU ← EL(Y_C, T_r).EU + EL(Z, T_r).EU;
08.             EL(Y_C, T_r).PU ← EL(Y_C, T_r).PU − EL(Z, T_r).EU;
09.     else PostSet(Y_C) ← PostSet(Y_C) ∪ Z;
10. Return Y_C;
```

Fig. 5: The ClosureComputation procedure

with its utility unit array. In other word, Y_C is a closed$^+$ HUI. The algorithm then calls the *GEN-CHUI* procedure to further explore the search space for mining more closed$^+$ HUIs that are succeeding supersets of Y_C. Then, the item I is added into $PrevSet(X)$. When the *CHUI-Mine*(*Closed*) algorithm completes, all the closed$^+$ HUIs in the database are obtained.

Example. Consider the database in Table 2 and *min_util* = 30. The algorithm scans the database once to construct *EU-Lists* of promising items. The set of promising items is P = {A, C, E, D, B}. Suppose items in itemsets and transactions are sorted according to the alphabetical order as total order R. Then, the algorithm calls *GEN-CHUI*(Ø, P, Ø, *min_util*). The algorithm then selects the smallest item A from P and set Y = {A}. Because $SumEU(Y) + SumPU(Y) = (12 + 8 + 4) + (14 + 17 + 17) = 72 \geq$ *min_util* and $PrevSet(Y)$ = Ø, the algorithm calls *ClosureComputation*({A}, Ø, {C, E, D, B}) and finds that Y_C = {ACE} and $PostSet(Y_C)$ = {D, B}.

The EU-List and utility unit array of {ACE} are shown in Fig. 6. Because $SumEU(Y_C) = 17 + 15 + 11 = 43 \geq$ *min_util*, {ACE} is output as a closed$^+$ HUI. Then the algorithm calls *GEN-CHUI*({ACE}, {D, B}, Ø, *min_util*) recursively and finds another closed$^+$ HUI {ACED}. The algorithm then does not output {ACEDB} and {ACEB} because their (*SumEU* + *SumPU*) are lower than *min_util*. The item A is then appended to $PrevSet(\emptyset)$ because all the closed$^+$ HUIs with pivot item A have been found. The procedure then considers the next item in P, which is C. The item C passes the *SumEU* + *SumPU* check but fails at *IsSubsumedCheck*({C}, {A}) because *TidSet*({C}) ⊆ *TidSet*({A}) and this latter procedure thus returns *true*. Item E also fails at *IsSubsumedCheck*({E}, {A}). Then, item D is picked. Now, Y = {D}, $PostSet(\emptyset)$ = {B} and $PrevSet(\{D\})$ = {A}. $SumEU(\{D\}) + SumPU(\{D\}) = (14 + 10 + 10) + (6 + 0 + 0) = 40 \geq$ *min_util* and *IsSubsumedCheck*({D}, {A}) returns *false* due to TidSet({D}) $\nsubseteq$ TidSet({A}). *ClosureComputation*({D}, Ø, {B}) does

not add items to Y_C = {D} and *SumEU*({D}) = 14 + 10 + 10 = 34 ≥ *min_util*, thus {D} is output as a closed$^+$ HUI. Then CHUI-Mine(Closed) stops after the (*SumEU* + *SumPU*) check is failed by {DB} and {B}.

{ACE}		
Tid	EU	PU
1	17	9
3	15	10
4	11	10
[24, 13, 6]		

{ACED}		
Tid	EU	PU
3	25	0
4	21	0
[12, 11, 3, 20]		

{ACEB}		
Tid	EU	PU
1	26	0
[12, 2, 3, 9]		

Fig. 6: EU-Lists and utility unit arrays of {ACE}, {ACED} and {ACEB}

4.3 The CHUI-Mine(Maximal) Algorithm

In this subsection, we present the CHUI-Mine(Maximal) algorithm. The algorithm has two input parameters: (1) a database D and (2) a minimum utility threshold *min_util*. It outputs the complete set of maximal HUIs in D. The framework of CHUI-Mine(Maximal) consists of two parts: (1) construction of *EU-Lists* of promising items and (2) generation of maximal HUIs by using *EU-Lists*. First, the algorithm scans D once to construct *EU-Lists* of promising items. Then, the algorithm calls the procedure *GEN-MHUI*(Ø, P, Ø, Ø, *min_util*) to generate maximal HUIs by using *EU-Lists*, where P is the set of all promising items.

In the GEN-MHUI procedure, each discovered maximal HUI is assigned with a unique identifier called *MID*. A special structure called *MIDList* is maintained for each item. The *MIDList* of an item I is denoted as *MIDList*(I) and consists of all the MIDs of already discovered maximal HUIs that are superset of I. Let $X = \{I_1, I_2, \ldots, I_k\}$ be a k-itemset, the *MIDList of X* is defined as *MIDList*(X) = *MIDList*(I_1) ∩ *MIDList*(I_2) ∩ . . . ∩ *MIDList*(I_k). If *MIDList*(X) is non-empty, X is not maximal because it is the subset of a previously discovered maximal HUI. The core idea of the GEN-MHUI procedure is that, it relies on the search procedure of CHUI-Mine(Closed) to generate maximal HUIs. For each closed HUI X generated by the search procedure of CHUI-Mine(Closed), the algorithm calculates *MIDList*(X) to check whether X is a subset of some already discovered maximal HUIs. If *MIDList*(X) is not empty, X is not a maximal HUI. Let $Z = X \cup PostSet(X)$, if *MIDList*(Z) is non-empty, the algorithm prunes the search space of maximal HUIs that are candidates of X. If *MIDList*(Z) is empty, the succeeding supersets of X could be maximal HUIs. The algorithm continues to explore the search space of maximal HUIs that are succeeding supersets of X. If there exists any succeeding supersets, X is not a

maximal HUI (Definition 12). Otherwise, X is a maximal HUI. Then the algorithm assigns a MID M_X to X and adds M_X to $MIDList(I_j)$ for every item $I_j \in X$.

4.4 Recovering High Utility Itemsets from Maximal High Utility Itemsets

In this subsection, we propose an algorithm named *RHUI* (*Recover all High Utility Itemsets from maximal patterns*) to efficiently recover all HUIs from the set of maximal HUIs. It takes as input the *min_utility* threshold and the set of maximal HUIs *MH*. RHUI outputs the complete set of HUIs respecting *min_utility*. RHUI proceeds as follows. First, the algorithm sorts items in each maximal HUI according to the total order R: $I_1 \prec I_2 \prec \ldots \prec I_N$ that was used to construct EU-Lists of promising items (Definition 16). Then, for each maximal HUI, the set of all its subsets is derived. For each subset, it is inserted into a lexicographic tree (i.e., a trie). The tree has a root representing the empty set and several branches. Each node in the branch represents the information of an itemset. A node representing the itemset X is denoted as $ND(X)$. Then, RHUI performs a depth-first traversal of the tree. For each traversed node $ND(X)$, RHUI calculates the utility of the itemset X by constructing the EU-Lists of X. If the utility of X is no less than *min_util*, the algorithm outputs X because it is a high utility itemset (Definition 6). If the sum of the remaining utility of X [9] and the utility of X is lower than *min_util*, nodes under the sub-tree of $ND(X)$ are pruned because itemsets represented by these nodes are low high utility. After traversing the tree once, all HUIs are obtained.

Example 2. Consider the database in Table 2. When *min_util* = 30, only one maximal high utility itemset {ACDE} is found by the CHUI-Mine(Maximal) algorithm. To recovery all the HUIs from the complete set of maximal HUIs, all the subsets of the maximal HUI {ACED} are derived. For each subset of {ACED}, we insert it into a lexicographic tree. For example, Fig. 7 shows a lexicographic tree with a total order R: A≺ C≺ E≺ D≺ B. Each node to the root represents a unique itemset. For example, the node C that is directly under the node A represents the itemset {AC}. For convenience, a node that represents an itemset X is denoted as $ND(X)$. For example, the node C that is directly under the node A is denoted as $ND(\{AC\})$. After constructing the tree, the RHUI algorithm starts a depth-first traversal of the tree. The first node it visits is $ND(\{A\})$ and its *EU-List* is already constructed by the previous process in HUI-Mine(Maximal). Then $SumEU(\{A\})$ and $SumPU(\{A\})$ are calculated. Because $SumEU(\{A\}) = 24 <$ *min_util*, {A} is not a HUI. Besides, because ($SumEU(\{A\}) + SumPU(\{A\})) = 78$ is higher than *min_util*, the algorithm visits the child nodes of $ND(\{A\})$ in lexicographic order. The next traversed node by the algorithm is $ND(\{AC\})$, the algorithm then constructs the *EU-List* of {AC} by $EL(\{A\})$ and $EL(\{C\})$ (Definition 22). Because $SumEU(\{AC\}) \geq$ *min_util*, it is outputted as a HUI.

Then the algorithm checks whether ($SumEU(\{AC\}) + SumPU(\{AC\})$) is higher than *min_util*. If it is not higher than *min_util*, the algorithm prune the nodes under

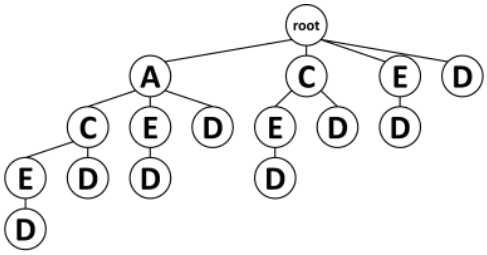


Fig. 7: A lexicographic tree for recovering high utility itemsets

the sub-tree of $ND(\{AC\})$. Otherwise, the algorithm continues to visits remaining nodes that have not been visited. After the traversal, all HUIs are obtained.

5 Experimental Evaluation

In this section, we evaluate the performance of CHUI-Mine(Closed) and CHUI-Mine(Maximal) with comparisons with CHUD [21] and HUI-Miner [9], which are benchmark algorithms for mining closed$^+$ HUIs and HUIs. Moreover, we evaluate the reduction obtained by mining closed$^+$ HUIs and maximal HUIs with respect to the set of all HUIs. Experiments were performed on a computer with a 3.40 GHz Intel Core Processor with 8 gigabytes of memory and Windows 7 SP1, 64 bit edition. All algorithms are written in Java, and run on JVM 1.7.0_45-b18. Both the initial and maximum heap memory sizes of the JVM are set to 2,048 MB, and memory measurement are done using the standard Java API. To analyze the performance of the algorithms in different situations, we test the algorithms with five different datasets, namely *Mushroom*, *Connect*, *Chess*, *Foodmart* and *Retail. Foodmart*. *Retail*, *Chess* and *Mushroom*, were obtained from the SPMF Repository [6]. The dataset *Foodmart* already contains unit profits and purchase quantities. For the other datasets, external utilities of items are generated between 1 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, as the settings of [20, 23]. Table 5 shows the characteristics of the datasets used in the experiments.

Table 5: Dataset Characteristics

Dataset	#Items	#Transactions	Avg. Transaction Length	Type
Mushroom	119	8,124	23	Dense
Connect	129	67,557	43	Dense
Chess	75	3,196	37	Dense
Foodmart	1,559	4,141	4.4	Sparse

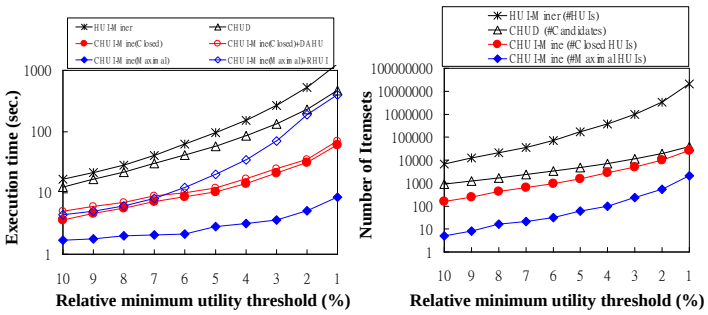
We also evaluate the performance for CHUI-Mine(Closed) with DAHU (denoted as CHUI-Mine(Closed)+DAHU) and CHUI-Mine(Maximal) with RHUI (denoted as CHUI-Mine(Maximal)+DAHU). CHUI-Mine(Closed)+DAHU first applies CHUI-Mine(Closed) to find all closed$^+$ HUIs and then uses DAHU to derive all HUIs from the set of generated closed$^+$ HUIs. CHUI-Mine(Maximal)+RHUI first applies CHUI-Mine(Maximal) to find all maximal HUIs and then uses RHUI to derive all HUIs from the set of generated maximal HUIs.

Table 6: Number of HUIs, Candidates Produces by CHUD, Closed$^+$ HUIs and Maximal HUIs on Different Datasets

Dataset	Relative min_util	HUI-Miner # HUIs	CHUD #Candidates	CHUI-Mine #Closed$^+$ HUIs	CHUI-Mine #Maximal HUIs
Mushroom	1%	20,392,064	39,522	25,611	2,081
Connect	30%	91,232	106,146	3,332	51
Chess	10%	58,482,852	99,635,8726	5,204,820	5,497
Foodmart	0.005%	230,617	6,637	6,635	3,676
Retail	0.005%	184,856	512,348	154,856	27,500

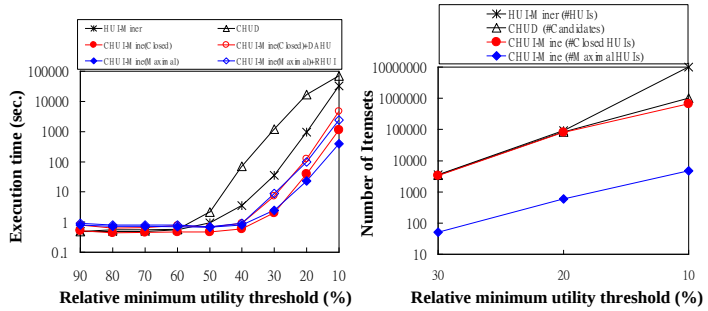
5.1 Experiments on Dense Datasets

We first evaluate the performance of the algorithms on dense datasets *Mushroom*, *Connect* and *Chess* under different relative minimum utility thresholds. Execution time of the algorithms and the number of HUIs, closed$^+$ HUIs and maximal HUIs are given in Fig. 8(a), Fig. 9(a), Fig. 10(a). In Fig. 9(b) and Fig. 10(b), we do not show the results when the relative *min_util* is higher than 30% because there is no HUIs in *Chess* and *Connect* for these threshold values. Results show that CHUI-Mine(Closed) outperforms CHUD in mining closed$^+$ HUIs. For example, on the *Chess* dataset, CHUI-Mine(Closed) completes the mining process in just 1,136 seconds, while CHUD takes over 70,000 seconds when the relative *min_util* =10%. A second observation is that mining maximal HUIs by CHUI-Mine(Maximal) is more efficient than discovering closed$^+$ HUIs or all HUIs. For example, when the relative *min_util* is set to 10 % for the *Connect* dataset, CHUI-Mine(Maximal) completes the mining task in just 296 seconds, while CHUI-Mine(Closed), *CHUD* and *HUI-Miner* take respectively 3,450, 120,000 and 130,000 seconds. Furthermore, when the relative *min_util* is less than 10%, the CHUD and HUI-Miner algorithms simply fail to finish the mining task within 200,000 seconds. The reason why CHUI-Mine performs so well is that CHUD and HUI-Miner respectively produce a large number of HUIs and candidates but CHUI-Mine discovers patterns without producing candidates, as shown in Table 4.



(a) Execution time (b) Number of itemsets

Fig. 8: The performance of the algorithms on Mushroom

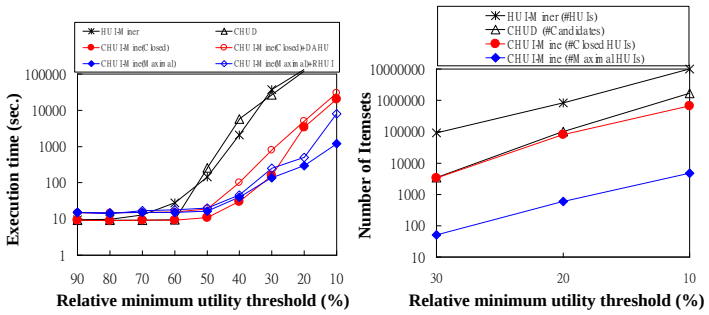


(a) Execution time (b) Number of itemsets

Fig. 9: The performance of the algorithms on Chess

5.2 Experiments on Sparse Datasets

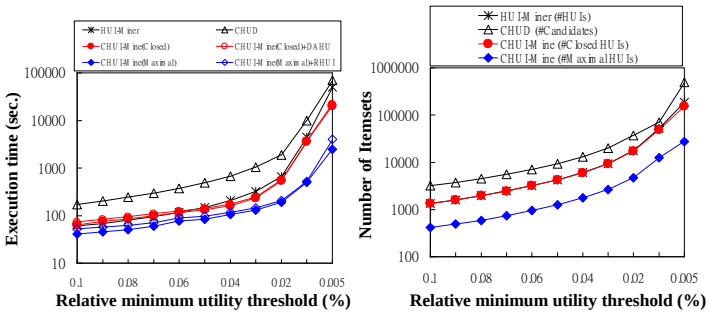
We then evaluate the performance of the algorithms on sparse datasets *Foodmart* and *Retail* under different relative minimum utility thresholds. Execution times of the algorithms and the number of HUIs, closed$^+$ HUIs and maximal HUIs are given in Fig. 11 and Fig. 12 Results show that CHUI-Mine(Closed) again outperforms both CHUD and HUI-Miner. But the performance gap is smaller than on dense datasets. When the relative minimum utility threshold is low, CHUI-Mine(Closed) is about twice faster than CHUD and from 3 to 9 times faster than HUI-Miner. At lower thresholds, CHUI-Mine(Maximal) is about 2 to 28 times faster than CHUD and from about 10 to 20 times faster than HUI-Miner. CHUI-Mine(Maximal) is also up to 8 times faster than CHUI-Mine(Closed) on the *Retail* dataset. But for the *Foodmart* dataset, the performance of CHUI-Mine(Maximal) is 5 % slower than CHUI-Mine(Closed). The reason why CHUI-Mine performs very well, similarly to results for dense datasets, is that (1) *CHUI-Mine* performs a *two way search* to build EU-Lists unlike HUI-Miner which performs a *three-way search* and (2) the closure



(a) Execution time (b) Number of itemsets

Fig. 10: The performance of the algorithms on Connect

of itemsets are constructed directly by *CHUI-Mine*, thus avoiding calculating utility-lists of several low utility or non-closed itemsets. Besides, we observe that mining closed$^+$ and maximal HUIs can greatly reduce the number of HUIs presented to the users. Fig. 11(b), Fig. 12(b) and Table 4 shows that when the relative *min_util* is set to 0.005%, the number of closed$^+$ HUIs is up to 35 times smaller than that of HUIs. Moreover, the number of maximal HUIs is up to about 2 times less than the number of closed$^+$ HUIs and 63 times less than that of HUIs.

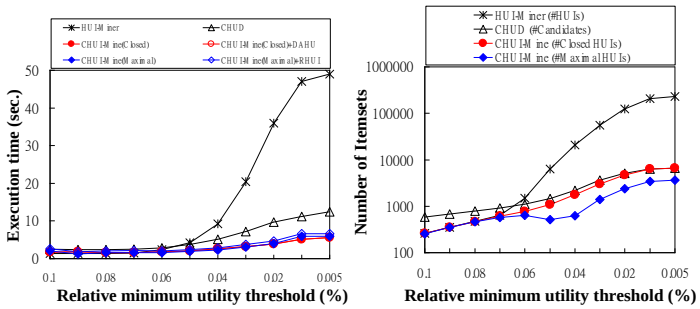


(a) Execution time (b) Number of itemsets

Fig. 11: The performance of the algorithms on Retail

5.3 Memory Usage Evaluation

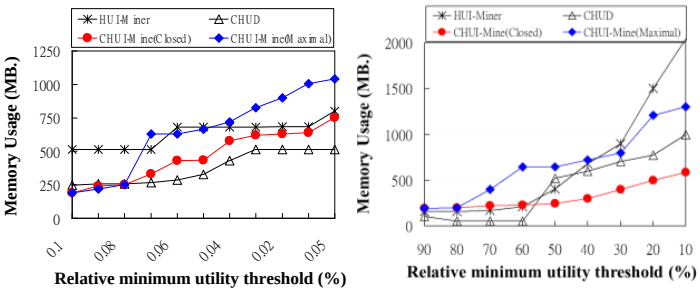
Then, we evaluate the memory consumption of the algorithms. Fig. 13 shows the memory consumption on a sparse dataset *Foodmart* and a dense dataset *Chess*. In



(a) Execution time (b) Number of itemsets

Fig. 12: The performance of the algorithms on Foodmart

Fig. 13(a), CHUD uses less memory than CHUI-Mine(Closed) because the main data structure of CHUD is local-TU Table, which stores less information that the structure of HUI-Miner (e.g., utility-list) and CHUI-Mine (e.g., EU-List). In Fig. 13(b), CHUI-Mine(Closed) performs better than CHUD because CHUD is a two-phase algorithm and it may produce too many candidates during the mining process, which may occupy lots of memory consumption, while CHUI-Mine(Closed) discovers patterns without producing candidates. In Fig. 13, CHUI-Mine(Closed) uses less memory than CHUI-Mine(Maximal) because CHUI-Mine(Maximal) needs to maintain *MIDLists* to check whether a closed HUI is maximal or not.



(a) Foodmart (b) Chess

Fig. 13: Memory consumption on different datasets

5.4 Recovery of All High Utility Itemsets

Fig. 8, 9 and 10 show the performance of the algorithms on dense databases *Mushroom*, *Chess* and *Connect*, respectively. These results show that both CHUI-Mine(Closed)+DAHU and CHUI-Mine(Maximal)+RHUI outperform HUI-Miner. An insightful observation is that when most subsets of a maximal HUI are not HUIs, HUI-Miner loses its advantages and CHUI-Mine(Maximal)+RHUI performs better than other algorithms. Fig. 11 and Fig. 12 show the performance of the algorithms on sparse databases *Retail* and *Foodmart*, respectively. In Fig. 11 and Fig. 12, our approaches still perform better than HUI-Miner. CHUI-Mine(Maximal)+RHUI runs slightly slower than CHUI-Mine(Closed)+DAHU on Foodmart because HUIs in Foodmart are quite short and most of closed$^+$ HUIs are maximal. In this case, CHUI-Mine(Maximal)+RHUI takes more processing time.

5.5 Summary

In the experiments, the most efficient algorithm is CHUI-Mine. First, we compare CHUI-Mine with CHUD. In general, CHUI-Mine always runs faster than CHUD. This is because CHUD performs two phases and may produce too many candidates in phase I, whereas CHUI-Mine performs just one phase and does not produce candidates. Then, we compare CHUI-Mine with HUI-Miner. CHUI-Mine performs better than HUI-Miner because the latter may produce a large number of HUIs during the mining process but CHUI-Mine discovers only a small summarization of HUIs. Besides, the cost for recovering HUIs is not expensive. Hence, the overall performance of CHUI-Mine is better than CHUD and HUI-Miner.

6 Conclusion

This paper is the first work that proposes a unifying framework for mining compact high utility itemsets in forms of closed$^+$ and maximal sets without candidate generation. A novel algorithm named *CHUI*(*Compact High Utility Itemset*)*-Mine* is proposed to efficiently discover the complete set of closed$^+$ and maximal high utility itemsets in the databases. Experimental results on several real datasets show that *CHUI-Mine* is more than two orders of magnitude faster than the benchmark algorithms for (closed$^+$) high utility itemset mining. The unifying framework with novel algorithms proposed in this work is a significant advancement in the field of utility mining with many promising and interesting topics that could be explored even further.

References

1. Agrawal, R. and Srikant, R.: Fast algorithms for mining association rules. *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, pp. 487-499 (1994)
2. Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., and Lee, Y.-K.: Efficient Tree Structures for High utility Pattern Mining in Incremental Databases. *IEEE Transactions on Knowledge and Data Engineering*, 21(12): 1708-1721 (2009)
3. Boulicaut, J. -F., Bykowski, A., and Rigotti, C.: Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1), pp. 5–22 (2003)
4. Calders, T. and Goethals, B.: Mining all non-derivable frequent itemsets. *Proc. of European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 74-85 (2002)
5. Chan, R., Yang, Q. and Shen, Y.: Mining high utility itemsets. *Proc. of IEEE Int'l Conf. on Data Mining*, pp. 19-26 (2003)
6. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C. W., Tseng, V. S.: SPMF: a Java Open-Source Pattern Mining Library, *Journal of Machine Learning Research*, 15, pp. 3389–3393 (2014)
7. Gan, W., Lin, J. C. W., Fournier-Viger, P., Chao, H. C., Tseng, V. S., Yu, P.: A Survey of Utility-Oriented Pattern Mining. arxiv:1805.10511 (2018)
8. Gouda, K. and Zaki, M. J.: GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets. *Data Min. Knowl. Discov.* 11(3): 223–242 (2005)
9. Liu, M, Qu,. J.: Mining high utility itemsets without candidate generation. *Proc. of ACM Int'l Conf. on Information and knowledge Management*. pp. 55–64 (2012)
10. Han, J., Pei, J., and Yin, Y.: Mining frequent patterns without candidate generation. *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, pp. 1-12 (2000)
11. Li, H. F., Huang, H. Y., Chen, Y. C., Liu, Y. J. and Lee S. Y.: Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams. *Proc. of IEEE Int'l Conf. on Data Mining (ICDM)*, pp. 881-886 (2008)
12. Liu, Y., Liao, W. and Choudhary, A.:A fast high utility itemsets mining algorithm. *Proc. of the Utility-Based Data Mining Workshop*, pp. 90-99 (2005)
13. Lucchese, C., Orlando, S. and Perego, R.:Fast and Memory Efficient Mining of Frequent Closed Itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 18(1): 21-36 (2006)
14. Li, Y. C., Yeh, J. S. and Chang, C. C.:Isolated Items Discarding Strategy for Discovering High utility Itemsets. *Data & Knowledge Engineering*, 64(1), pp. 198-217 (2008)
15. Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L.:Efficient mining of association rules using closed itemset lattice. *Journal of Information Systems*, Vol 24, Issue 1, pp. 25–46 (1999)
16. Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L.:Discovering Frequent Closed Itemsets for Association Rules. *Proc. of Int'l Conf. on Database Theory*, pp. 398–416, (1999)
17. Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L.:Generating a Condensed representation for Association Rules, *Journal of Intelligent Information Systems*, Vol 24, Issue 1, pp. 29–60 (2005)
18. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S. and Yang, D.:H-mine: fast and space-preserving frequent pattern mining in large databases. *IIE Transactions*, 39(6), pp. 593-605, (2007)
19. Shie, B. E., Tseng, V. S. and Yu, P. S.:Online Mining of Temporal Maximal Utility Itemsets from Data Streams. *Proc. of Annual ACM Symposium on Applied Computing*, pp. 1622-1626 (2010)
20. Tseng, V. S., Shie, B. E., Wu, C. W., Yu., P. S.:Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(8): 1772–1786 (2013)
21. Tseng, V. S., Wu, C. W., Fournier-Viger, P., Yu, P. S.:Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 27(3): 726-739 (2015)
22. Tseng, V. S., Wu, C. W., Fournier-Viger, P., Yu, P. S.:Efficient Algorithms for Mining Top-K High Utility Itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 28(1): 54–67 (2016)

23. Tseng, V. S., Wu, C. W., Lin, J. H., Fournier-Viger, P.: UP-Miner: A Utility Pattern Mining Toolbox. *Proc. of IEEE Int'l Conf. on Data Mining*, pp. 1656–1659 (2015)
24. Tseng, V. S., Wu, C. W., Shie, B. E. and Yu, P. S.: UP-Growth: an efficient algorithm for high utility itemset mining. *Proc. of Int'l Conf. on ACM SIGKDD*, pp. 253–262, 2010.
25. Wu, C. W., Fournier-Viger, P., Gu, J. Y., Tseng, V. S.: Mining Closed+ High Utility Itemsets without Candidate Generation. *Proc. of Conf. on Technologies and Applications of Artificial Intelligence*, pp. 187–194 (2015)
26. Wu, C. W., Fournier-Viger, P., Yu, P. S. and Tseng, V. S.: Efficient Mining of a Concise and Lossless Representation of High Utility Itemsets. *Proc. of IEEE Int'l Conf. on Data Mining*, pp. 824-833 (2011)
27. Wang, J., Han, J. and Pei, J.: Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. *Proc. of Int'l Conf. on ACM SIGKDD*, pp. 236–245, (2003).
28. Zaki, M. J. and Hsiao, C. J.: Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4): 462–478, (2005)