ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 4, Issue 2 (Apr. - Jun. 2016)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2016)

Mining the Laconic Representation of Itemsets

P. Vimal Kumar
PG Scholar, Dept. of Computer Engineering, P.S.R Engineering College, Sivakasi, Tamil Nadu, India

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

Keywords
Frequent itemset, closed + high utility itemset, lossless and concise representation, utility mining, and data mining.

I. Introduction

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

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

However, HUI mining is not an easy task since the downward closure property in FIM does not hold in utility mining. In other words, the search space for mining HUIs cannot be directly reduced as it is done in FIM because a superset of a low utility itemset can be a high utility itemset. Many studies 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, non-derivable sets, odds ratio patterns, disjunctive closed itemsets, maximal itemsets and closed itemsets. These representations successfully reduce the number of itemsets found, but they are developed for FIM instead of HUI mining.

II. Problem Definition

The problem of high-utility itemset mining is defined as follows. Let I be a finite set of items (symbols). An itemset X is a finite set of items such that $X \subseteq I$. A transaction database is a multiset of transactions $D = \{T1, T2,..., Tn\}$ such that for each transaction Tc, $Tc \subseteq I$ and Tc has a unique identifier c called its TID (Transaction ID). Each item $i \in I$ is associated with a positive number p (i), called its external utility (e.g. unit profit). Every item i appearing in a transaction Tc has a positive number q (i, Tc), called its internal utility (e.g. purchase quantity). For example, consider the database in Table 1, which will be used as the running example. It contains five transactions (T1, T2,…,T5). Transaction T2 indicates that items a, c, e and g appear in this transaction with an internal utility of respectively 2, 6, 2 and 5. Table 2 indicates that the external utility of these items are respectively 5, 1, 3 and 1.

Table1. Transaction database

TID	Transaction
T1	(a,1)(c,1)(d,1)
T2	(a,2)(c,6)(e,2)(g,5)
T3	(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)
T4	(b,4)(c,3)(d,3)(e,1)
T5	(b,2)(c,2)(e,1)(g,2)

Table2 : External utility values

Item	a b c d e f g
Profit	5 2 1 2 3 1 1

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

III. Related Work

Section 3.1 reviews the preliminaries associated with high utility itemset mining, Section 3.2 explains an closed itemset mining. Finally, Section 3.3 discusses other compact representations of high utility itemsets.

1. High Utility Itemset Mining

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

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

Definition 2 (Absolute utility of an item in a transaction): The absolute utility of an item ai in a transaction T_R is denoted as au (a_i, T_R) and defined as p (a_i, D) X q (a_i, T_R).

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

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

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

Definition 6 (High utility itemset): An itemset X is called high utility itemset iff u(X) is no less than a user-specified minimum utility threshold min_utility (0% < min_util ≤ 100%). Otherwise, X is a low utility itemset. An equivalent definition is that X is high utility iff au(X) ≥ abs_min_util, where abs_min_util is defined as min_util X TotalU.

Definition 7 (Complete set of HUIs in the database): Let S be a set of itemsets and a function f_H (S) = {X|X ∈ S, u(X) ≥ min_utility}. The complete set of HUIs in D is denoted as H (H ⊆ L) and defined as f_H (L). The problem of mining HUIs is to find the set H in D.

Example 1 (High Utility Itemsets): Let Table 3 be a database containing five transactions. Each row in Table 1 represents a transaction, in which each letter represents an item and has a purchase quantity (internal utility).

Table3 : An Example Database

TID	Transaction	Transaction Utility (TU)
T1	A(1), B(1), E(1), W(1)	5
T2	A(1), B(1), E(3)	8
T3	A(1), B(1), F(2)	8
T4	E(2), G(1)	5
T5	A(1), B(1), F(3)	11

Table 4 : Profit Table

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

The unit profit of each item is shown in Table 4 (external utility). In Table 3, the absolute utility of the item {F} in the transaction T_3 is au ({F}, T_3) = p ({F}, D) x q ({F}, T3) = 3 x 2 = 6. The absolute utility of {BF} in T_3 is au ({BF}, T3) = au ({B}, T3) + au ({F}, T3) = 1 + 6 = 7. The absolute utility of {BF} is au ({BF}) = u ({BF}, T3) + u ({BF}, T5) = 17. If abs_min_utility = 10, the set of HUIs in Table 3 is H = {{E} : 12, {F} : 15, {AE} : 10, {AF} : 7, {BE} : 10, {BF} : 17, {ABE} : 12, {ABF} : 19}, where the number beside each itemset is its absolute utility.

2. Closed Itemset Mining

In this section, introduce definitions and properties related to closed itemsets and mention relevant methods.

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

Property 2: For itemsets X, Y ∈ L, SC(X U Y) = | g(X) ∩ g(Y) |.**Definition 11 (Closure of an itemset):** The closure of an itemset X ∈ L, denoted as C(X), is the largest set Y ∈ L such that X ⊆ Y and SC(X) = SC(Y). Alternatively, it is defined as C(X) = ∩ R□g (X) T_R.

Property 3: YX ∈ L, SC(X) = SC(C(X)) ↔ g(X) = (C(X)).

Definition 12 (Closed itemset): An itemset X ∈ L is a closed itemset iff there exists no itemset Y ∈ L such that (1) X C Y and (2) SC(X) = SC(Y). Otherwise, X is non-closed itemset. An equivalent definition is that X is closed iff C(X) = X. For example, in the database of Table 1, {B} is non-closed because C ({B}) = $T_1 \cap T_2 \cap T_3 \cap T_5$ = {AB}.

Definition 13 (Complete set of closed itemset in the database): Let S be a set of itemsets and a function f_C(S) = {X|X ∈ S, ¬□Y ∈ S such that X C Y and SC(X) = SC(Y)}. The complete set of closed itemsets in D is denoted as C(C ⊆ L) and defined as f_C (L).

3. Compact Representations of High Utility Itemset Mining

To present representative HUIs to users, some concise representations of HUIs were proposed. Chan et al. introduced the concept of utility frequent closed patterns [6]. However, it is based on a definition of high utility itemset that is different from my work. Shie et al. proposed a compact representation of HUIs, called maximal high utility itemset and the GUIDE algorithm for mining it .A HUI is said to be maximal if it is not a subset of any other HUI. For example, if abs_min_utility = 10, the set of maximal HUIs in Table 3 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.

IV. Closed + High Utility Itemset Mining

In this section, incorporate the concept of closed itemset with high utility itemset mining to develop a representation named closed + high utility itemset. We theoretically prove that this new

representation is meaningful, lossless and concise (i.e., not larger than the set of all HUIs).

1. Push Closed Property into High Utility Itemset Mining

The first point describes how to incorporate the closed constraint into high utility itemset mining. There are several possibilities. First, define the closure on the utility of itemsets. However, this definition is unlikely to achieve a high reduction of the number of extracted itemsets. A second possibility is to define the closure on the supports of itemsets. In this case, there are two definitions.

— Mine all the high utility itemsets first and then apply the closed constraint.

— Mine all the closed itemsets first and then apply the utility constraint.

Definition 14 (Closed high utility itemset): Define the set of closed high utility itemsets as HC = {X | X ∈ L. X = C(X), u(X) ≥ min_utility}, HC = H' = C. An itemset X is called non-closed high utility itemset iff X ∈ H and X □/C. For example, if abs_min_utility = 10, the complete set of closed HU1s in Table 3 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.

2. Efficient Algorithms for Mining Closed + High Utility Itemsets Mining

Introduce three efficient algorithms AprioriHC (An Apriori-based algorithm for mining High utility Closed + itemsets), AprioriHC-D (AprioriHC algorithm with Discarding unpromising and isolated items) and CHUD (Closed + High Utility itemset Discovery) for mining CHUIs. They rely on the TWU-Model and include strategies to improve their performance. All algorithms consist of two phases named Phase I and Phase II. In Phase I, potential closed + high utility itemsets (PCHUIs) are found, which are defined as a set of itemsets having an estimated utility (e.g. TWU) no less than abs_min_utility. In Phase II, by scanning the database once, CHUIs are identified from the set of PCHUIs found in Phase I and their utility unit arrays are computed. The AprioriHC and AprioriHC-D are based on Apriori and the Two-Phase algorithms. They use a horizontal database and explore the search space of CHUIs in a breadth-first search. The algorithm AprioriHC is regarded as a baseline algorithm in this work and AprioriHC-D is an improved version of AprioriHC. On the other hand, the proposed algorithm CHUD is an extension of Eclat and DCI-Closed algorithms. The CHUD algorithm considers vertical database and mines CHUIs in a depth-first search. In the following, we present details of the three algorithms.

A. The AprioriHC Algorithm

Initially, a variable k is set to 1. The algorithm performs a database scan to compute the transaction utility of each transaction (Definition 4). At the same time, the TWU of each item is computed. Each item having a TWU no less than abs_min_utility is added to the set of 1-HTWUIs C_k. Then the algorithm proceeds recursively to generate itemsets having a length greater than k. During the k^{th} iteration, the set of k-HTWUIs L_k is used to generate (k + 1) candidates C_{k+1} by using the Apriori-gen function. Then the algorithm computes TWUs of itemsets in C_{k+1} by scanning the database Donce. Each itemset having a TWU no less than abs_min_utility is added to the set of (k + 1)-HTWUIs L_{k+1}. After that, the algorithm removes non-closed itemsets in L_{k+1} by the following process. For each candidate X in L_{k+1}, the algorithm checks if there exists a subset Y ⊆ X such that Y ∈ L_k and SC(X) = SC(Y). If true, Y is deleted from L_k because Y is not a closed + high utility itemset according to Definition 14. If false, Y is kept and marked as "closed" because it may be a closed + high utility itemset. The phase I of AprioriHC terminates when no candidate is generated. Then, the algorithm performs Phase II. In phase II, the algorithm scans the database once and calculates the utilities of HTWUIs that are marked as "closed" to identify the set of closed + high utility itemsets.

B. The AprioriHC-D Algorithm

The AprioriHC-D algorithm is an improvement of AprioriHC. It includes two effective strategies to reduce the number of candidates generated in Phase I.

Input: D: the database; abs_min_utility;
pCHUI: the set of PCHUIs pCHUI
Output: The complete set of CHUIs
pCHUI:=Ø
L1:=1-HTWUIs in D
D1:=DGU_Strategy(D,L1)
L1:=1-HTWUIs in D1
AprioriHC-D_Phase-I(D1, pCHUI, abs_min_utility,L1)
AprioriHC-D_Phase-II(D1, pCHUI, abs_min_utility)

Alg. 1: AprioriHC-D algorithm

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

Definition17 (Promising item): An item ip is a promising item iff TWU (i_p) ≥ abs_min_utility. Otherwise, it is an unpromising item.

Property9: Any unpromising item i_u and its supersets are not high utility itemsets.

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

Adapt Property 9 to the context of closed + high utility itemset mining as follows.

Property10: Any unpromising item i_u and its supersets are not closed + high utility itemsets.

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

Strategy1: DGU (Discarding Global Unpromising items). Discard global unpromising items and their exact utilities from transactions and transaction utilities of the database, respectively.

Rationale: By Property 10, unpromising items play no role in CHUIs. Therefore, unpromising items can be removed from each transaction T_R and their absolute utilities can be subtracted from TU (T_R). Thus, the utilities of unpromising items can be ignored in the calculation of the estimated utilities of itemsets (i.e., TWU). The second strategy is based on the following definition and properties.

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

Definition 18 (Isolated item): Let L_k be the set of HTWUIs of length k, an item i_o is called an isolated item of level k iff i_o is not contained in any itemset in L_k.
Property11: For any isolated item i_o of level k, its supersets of length l ($l \geq k$) are not high utility itemsets.
Property12: For any isolated item i_o of level k, its supersets of length l ($l \geq k$) are not CHUIs.
Proof: For any isolated item i_o of level k, its supersets of level l ($l \geq k$) are not HUIs (Property 11). Thus, the supersets of l ($l \geq k$) are not CHUIs.

C. The CHUD Algorithm

Present an efficient depth-first search algorithm named CHUD (Closed + High Utility itemset Discovery) to discover CHUIs. CHUD is an extension of DCIClosed, one of the currently best methods to mine closed itemsets.

Input: D: the database; abs_min_utility
Output: The complete set of CHUIs
InitialDatabaseScan(D)
RemoveUtilityUnpromisingItems(O,GTU)
For each item $a_k \in 0$ do
{ Create node N({a_k})
CHUD_Phase-I(N({a_k}),GTU,abs_min_utility)
REG_Strategy(g(a_k),GTU)}
CHUD_Phase-II(D,abs_min_utility)

Alg. 2: CHUD algorithm

CHUD is adapted for mining CHUIs and include several effective strategies for reducing the Similar to the DCI-Closed algorithm, CHUD adopts an Itemset-Tidset pair Tree (IT-Tree) to find CHUIs. In an IT-Tree, each node N(X) consists of an itemset X, its Tidset g(X), and two ordered sets of items named PREV-SET(X) and POST-SET(X). The IT-Tree is recursively explored by the CHUD algorithm until all closed itemsets that are HTWUIs are generated. Different from the DCI-Closed algorithm, each node N(X) of the IT-Tree is attached with an estimated utility value EstU(X).
A data structure called transaction utility table (TU-Table) 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.

Input: g(X): the Tidset of X; TU: a TU table
Output: EstU: the estimated utility of X
EstU :=0;
for each TID R ∈ g(X) do
{ EstU := EstU + TU.get(R)}
Return EstU

Alg. 3: CalculateEstUtility procedure

Given a TID R, the value TU (TR) 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 alg 3.

3. Efficient Recovery of High Utility Itemsets

Present a top-down method named DAHU (Derive All High Utility itemsets) for efficiently recovering all the HUIs and their absolute utilities from the complete set of CHUIs. The pseudo code of DAHU is shown in alg. 4. It takes as input an absolute minimum utility threshold abs_min_utility, a set of CHUIs HC and ML the maximum length of itemsets in HC. DAHU outputs the complete set of high utility itemsets H□i=1 H_k respecting abs_min_utility, where H_k denotes the set of HUIs of length k. To derive all HUIs, DAHU proceeds as follows. First, the set HML is initialized to HC_{ML}, where the notation HC_k represents the set of k-itemsets in HC. During lines 2 to line 14 in alg. 4, each set H_k is constructed from k = (ML - 1) to k = 1. In each iteration,
H_{k-1} is recovered by using HC_k. For each itemset X = {$a_1, a_2, \ldots, a_k$} in HC_k, if the absolute utility of X is no less than abs_min_utility, the algorithm outputs the high utility itemset X with its absolute utility and then generates all (k -1)-subsets of X. The latter are obtained by removing each item $a_i \in X$ from X one at a time to obtain subsets of the form Y = X – {a_i}. If Y is not present in H_k or Y is present in H_k with SC(X) > SC(Y), Y is added to H_{k-1}, its support count is set to the support count of X (Property 4), i.e., SC(Y) = SC(X), and the absolute utility of Y is set to the absolute utility of X minus the i^{th} value in V(X), i.e., au(Y) = au(X) - V (X, a_i) (Property 8). This process is repeated until H has been completely recovered.

Input: ML: the maximum length of itemset in HC; abs_min_ utility;
HC= {$HC_1, HC_2, \ldots, HC_{ML}$}: the complete set of CHUIs;
Output: H: The complete set of HUIs
H_{ML} := HC_{ML}
For (k:=ML-1;k>0;k--) do
{ for each k-itemset X={ $a_1, a_2, \ldots, a_k$} ∈ HC_k do
{ if(au(X) < abs_min_utility) then delete X from HC_k
Else add X and its absolute utility au(X) to H.
{ for each item $a_i \in X$ do
{ Y:=X-{a_i}
au(Y):= au(X)-V(X , a_i)
if(au(Y) ≥ abs_min_utility) then
{ if Y ∈ HC_{k-1} and SC(X) > SC(Y) then
{ SC(Y):= SC(X) }
else if Y □ HC_{k-1} then
{ HC_{k-1}:= HC_{k-1} U Y
SC(Y) := SC(X) }}}}}}

Alg. 4: DAHU algorithm

V. Experimental Evaluation

In this section, evaluate the performance of the proposed algorithms and compare them with two state-of-the-art algorithms UP-Growth and Two-Phase. Although our methods produce different results from those algorithms, they also consist of two phases. In Phase I, the proposed algorithms generate candidates for CHUIs, whereas UP-Growth and Two-Phase generate candidate for HUIs. In Phase II, the proposed algorithms and UP-Growth/Two-Phase respectively identify CHUIs and HUIs from candidates produced in their Phase I. Furthermore, we have also considered the performance of CHUD with DAHU, denoted as CHUD+DAHU. CHUD+DAHU first applies CHUD to find all CHUIs and then uses DAHU to derive all HUIs from the set of CHUIs generated by CHUD. The process of CHUD+DAHU in Phase I is the same as that of CHUD. In Phase II, CHUD+DAHU first identifies CHUIs from candidates and then use CHUIs to derive all HUIs. In the experiments, we do not combine AprioriHC/ AprioriHC-D with

DAHU because CHUD outperforms these algorithms, as it will be shown, and they produce the same output. Experiments were performed on a desktop computer with an Intel Core 2 Quad Core Processor @ 2.66 GHz running Windows XP and 2 GB of RAM. All the algorithms were implemented in Java.

Both synthetic and real datasets were used to evaluate the performance of the algorithms. A synthetic dataset T12-I10-N1K-Q5-D200K was generated by the IBM data generator. The parameters of the data generator are described in Table 5. Mushroom and BMSWebView1 were obtained from FIMI Repository. Mushroom is a real-life dense dataset, each transaction containing 23 items. BMSWebView1 is a real-life dataset of click stream data with a mix of short and long transactions (up to 267 items).

Table 5 : Characteristics of Datasets

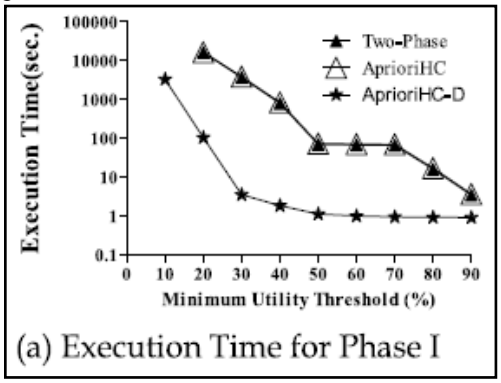
Dataset	N	T	D
Mushroom	119	23	8,124
Foodmart	1,559	4.4	4,141
BMSWebView1	497	2.51	59,601
T12-I10-N1K-Q5-D200K	1,000	12	200 K

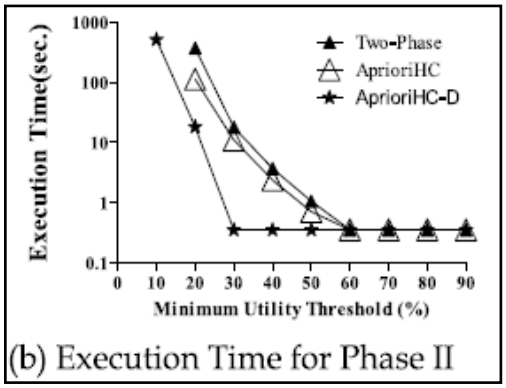
Foodmart is a real-life dataset obtained from the Microsoft foodmart 2,000 database, which contains real external and internal utilities. For remaining datasets, the quantity of each item is randomly generated from 1 to 5 and the external utility of each item is randomly generated from 0.01 to 10.00. The external utility follows a log normal distribution. Table 4 shows the characteristics of the above datasets. Depending on the applications, the characteristics and count distributions of the datasets can be very different. However, there are three kinds of datasets that are commonly encountered in real-life scenarios: (1) dense dataset, (2) sparse dataset, and (3) dataset containing long transactions.

In the experiments, we use three real-life datasets Mushroom, Foodmart, BMSWebView1 to respectively represent the above three real cases. The experimental results on these datasets are separately shown and discussed in the Sessions 5.1, 5.2 and 5.3. The scalability and the memory consumption of the proposed algorithms are respectively shown in the Sessions 5.4 and 5.5.

1. Experiments on Mushroom Dataset

The performance of the algorithms on the Mushroom dataset is shown in Fig 1. In Fig 1a, the execution time of Two-Phase and AprioriHC is similar in Phase I. The reason is that Two-Phase and AprioriHC simply apply the TWU Model without using effective strategies to reduce the estimated utility of candidates in Phase I. Besides, AprioriHC-D runs faster than Two-Phase and AprioriHC. Table 6 shows the number of candidates generated by Two-Phase, AprioriHC and AprioriHC-D in Phase I. We did not show the number of HUIs and CHUIs in Table 6 because there are no HUIs and CHUIs when min_utility is higher than 10 percent. In Table 6, AprioriHC and AprioriHC-D generate fewer candidates than Two-Phase. This is because AprioriHC and AprioriHC-D produce candidates for CHUIs but Two-Phase needs to produce candidates for all HUIs. By applying strategies DGU and IIDS, AprioriHC-D produces fewer candidates than AprioriHC. In Fig. 1b, AprioriHC runs faster than Two-Phase because Two-Phase needs to verify the utility of more candidates in Phase II. Though AprioriHC needs to calculate the utility unit array of candidates and Two-Phase does not, this cost is not expensive. In Fig. 1c, we can see that CHUD outperforms all the other algorithms for both phases. For example, when min_utility = 1%, CHUD is 50 times faster than UP-Growth for Phase I and 63 times faster for Phase II. Moreover, when CHUD is combined with DAHU to discover all HUIs, the combination largely outperforms UP-Growth and was only slightly slower than CHUD. Table 7 shows the number of candidates and the number of results generated by UP-Growth, CHUD, and CHUD + DAHU. CHUD generates a much smaller number of candidates and results than UP-Growth. The smaller number of candidates generated by CHUD in Phase I is what makes CHUD perform better than UP-Growth for the total execution time. In Table 7, a huge reduction in the number of extracted patterns (up to 796 times) is achieved by the representation of closed + high utility itemsets. Moreover, by running DAHU, it is possible to recover all HUIs.





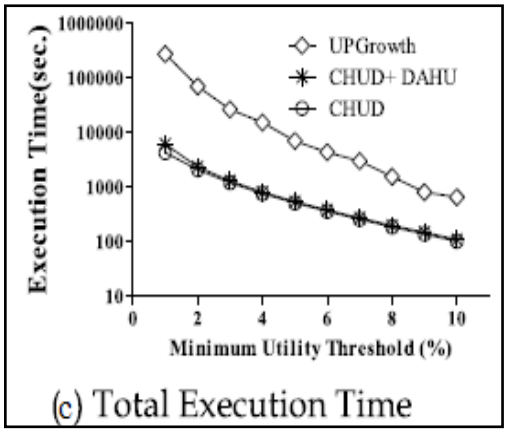


Fig. 1: Execution time on mushroom dataset

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

Table 6. Number of Candidates in Phase I on Mushroom

Mini-mum Utility	Cand for Two-Phase	Cand For AprioriHC	Cand for Apriori-HC-D
80%	25	18	5
60%	51	35	8
40%	533	341	29
20%	53,127	16,194	2,635

Moreover, when CHUD is combined with DAHU is to discover all HUIs, the combination.

2. Experiments on Foodmart Dataset

The performance of the algorithms on the Foodmart dataset is shown in Fig. 2. Results show that AprioriHC-D runs faster than both Two-Phase and AprioriHC. The execution time of AprioriHC and AprioriHC-D is similar in Phase II. When min utility = 0.05%, AprioriHC-D is three times faster than Two-Phase in total execution time. Table 7 shows the number of candidates generated by Two-Phase, AprioriHC and AprioriHC-D. AprioriHC and AprioriHC-D generate much less candidates than Two-Phase. For example, when min_utility = 0.05%, Two-Phase generates 233,185 candidates, and AprioriHC and AprioriHC-D generate both 6,657 candidates. The reason is that AprioriHC and AprioriHC-D produce candidates for CHUIs, but Two-Phase needs to produce candidates for all HUIs. In Fig. 2a, the total execution time of UP-Growth is less than CHUD, initially. But as the min_utility threshold became smaller, CHUD becomes faster (up to twice faster than UP-Growth). The reason why the performance gap between CHUD and UP-Growth is smaller for Foodmart than for Mushroom is due to the fact that Foodmart is a sparse dataset. As a consequence the reduction achieved by mining CHUIs is less (still up to 34.6 (230,617/6,656) times). Note that achieving a smaller reduction for sparse datasets is a well-known phenomenon in frequent closed itemset mining.

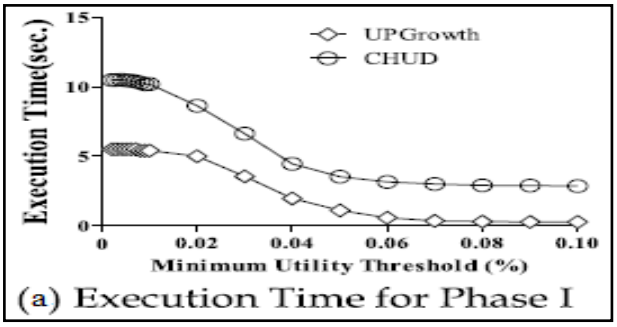


Fig. 2: Execution time on foodmart dataset

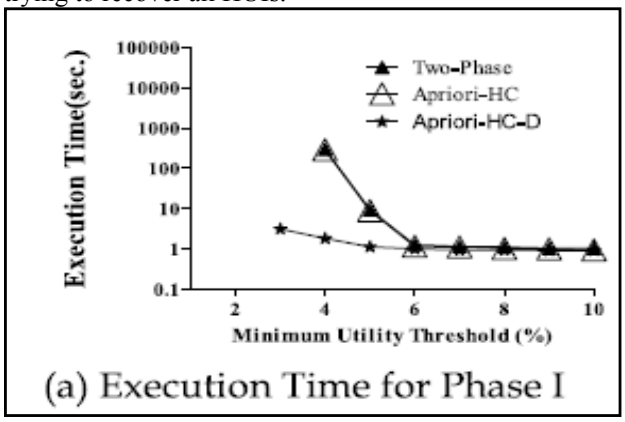
Table7. Number of Extracted Patterns on Foodmart

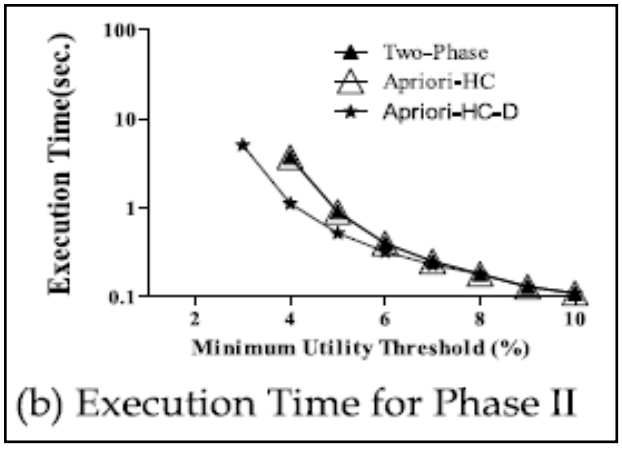
Minimum Utility (%)	UP-Growth		CHUD	
	Phase I #Cand. for HUIs	Phase II #HUIs	Phase I #Cand. for CHUIs	Phase II #CHUIs
0.1%	1,585	258	581	258
0.05%	37,158	6,266	1,444	1,076
0.01%	230,165	209,387	6,332	6,293
0.005%	233,032	230,617	6,657	6,656

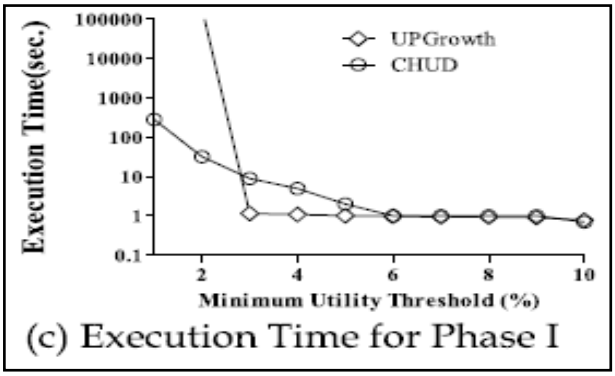
A similar phenomenon occurs in closed + high utility itemset mining. Besides, when DAHU was combined with CHUD, the execution time of CHUD+DAHU was up to twice faster than UP-Growth and slightly slower than CHUD.

3. Experiments on BMSWebView1 Dataset

The performance of the algorithms on the BMSWebView1 dataset is shown in Fig. 3. In Fig. 3a, the execution time of Two-Phase and AprioriHC is similar in Phase I. In Fig. 3b, AprioriHC-D runs faster than Two-Phase and AprioriHC in Phase II. When min_utility = 4%, AprioriHC and Two-Phase cannot terminate within the time limit of 100,000 seconds and they generate more than 1,000,000 candidates in Phase I. When min_utility = 3%, AprioriHC-D performance starts degrading since too many candidates are produced. In Figs. 17c and 17d, UP-Growth runs faster than CHUD and CHUD+DAHU for min_utility ≥ 3%. However, for min_utility < 3%, the performance of UP-Growth decreases sharply. For min_utility = 2%, UP-Growth cannot terminate within the time limit of 100,000 seconds and it generates more than 1,000,000 candidates in Phase I, Whereas CHUD terminates in 80 seconds and produces only 7 CHUIs from 32 candidates. The reason why CHUD performs so well is that it achieves a massive reduction in the number of candidates by only generating a few long itemsets containing up to 149 items, while UP-Growth has to consider a huge amount of redundant subsets (for a closed itemset of 149 items, there can be up to 2149 - 2 non-empty subsets that are redundant). DAHU also suffers from the fact that there are too many HUIs. It runs out of memory for min_utility < 2% when trying to recover all HUIs.







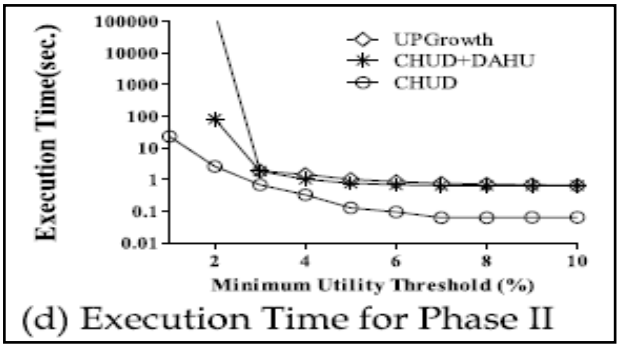
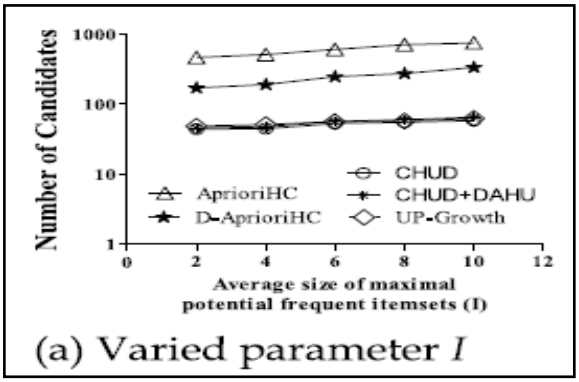


Fig. 3: Execution Time on BMSWebView1 Dataset

4. Scalability of the Proposed Methods

In this section, we evaluate the scalability of the algorithms. The scalability of the proposed algorithms under varied size of potential maximal frequent patterns is shown in Fig. 4a. The experiments are performed on
synthetic datasets T12-Ix-N1K-Q5-D100K, where x is varied from 2 to 10 (e.g. I2, I4, I6, I8 and I10).The scalability of the proposed algorithms under varied number of distinct items is shown in Fig. 4b. The experiments are performed on synthetic datasets T12-I8-NxK-Q5-D100K, where x is varied from 2 to 10 (e.g. N2K, N4K, N6K, N8K and N10K). In these two experiments, the min_utility threshold is fixed to 0.5 percent. As shown in Fig. 4, all the proposed algorithms have good scalability when I and N are varied. Besides, CHUD and CHUD+DAHU perform better than the other algorithms.



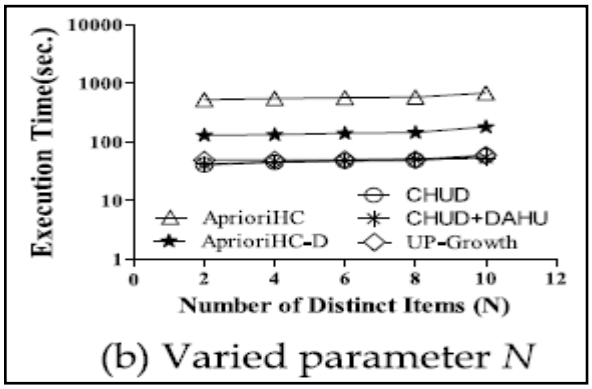
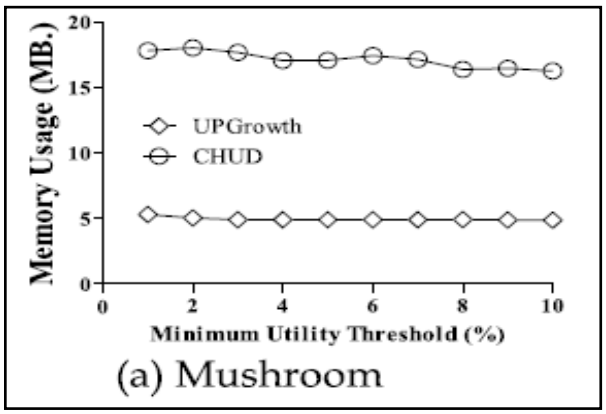
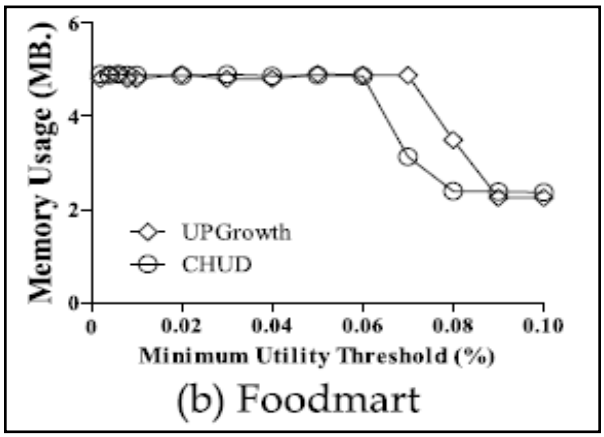


Fig. 4: Scalability Test

5. Memory Usage

Measure the maximum memory usage of UP-Growth and CHUD in phase I by using the Java API. In general, CHUD uses as much or more memory than UP-Growth because the latter uses a compact trie-based data structure for representing the database that is more memory efficient than a vertical database. For example, detailed results for Mushroom and Foodmart are presented in Figs. 19a and 19b. However, when the databases contain very long HUIs such as BMSWebView1, the number of candidates can be very large. As shown in Fig. 5c, when the minimum utility threshold is set to 2 percent, the memory consumption of UP-Growth rises dramatically because it needs to create a number of conditional UP Trees that is proportional to the number of candidates.





ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

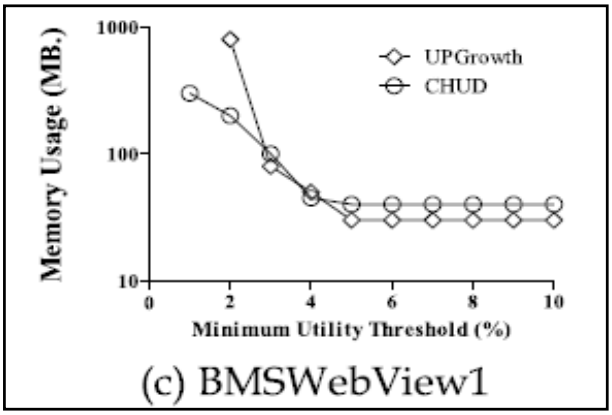


Fig. 5: Memory Consumption

VI. Conclusions

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

References

[1] *VikramGoyal, Siddharth Dawar," UP-Hist Tree: Efficient Data Structure for High Utility Pattern Mining from Transaction Databases" VOL. 20, NO. 4, pp.726-739, MARCH 2015*

[2] *G.C.Lan, T.P. Hong, and V. S. Tseng, "An efficient projection based indexing approach for mining high utility itemsets" Knowl. Inf. Syst, vol. 38, no. 1, pp. 85–107, 2014.*

[3] *T. Hamrouni, "Key roles of closed sets and minimal generators in concise representations of frequent patterns" Intell. Data Anal, vol. 16, no. 4, pp. 581–631, 2012.*

[4] *C. W. Wu, B.E. Shie, V. S. Tseng, and P. S. Yu, "Mining top-k high utility itemsets" in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2012, pp. 78–86.*

[5] *C.W. Lin, T.P. Hong, and W.H. Lu, "An effective tree structure for mining high utility itemsets," Expert Syst. Appl., vol. 38, no. 6, pp. 7419–7424, 2011.*

[6] *B.E. Shie, H.F. Hsiao, V. S. Tseng, and P. S. Yu, "Mining high utility mobile sequential patterns in mobile commerce environments" in Proc. Int. Conf. Database Syst. Adv. Appl., 2011, vol. 6587, pp. 224–238.*

[7] *B.E. Shie, V. S. Tseng, and P. S. Yu, "Online mining of temporal maximal utility itemsets from data streams" in Proc. Annu. ACM Symp. Appl. Comput., 2010, pp. 1622–1626.*

[8] *V. S. Tseng, C.W. Wu, B.E. Shie, and P. S. Yu, "UP-Growth: An efficient algorithm for high utility itemset mining" in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2010, pp. 253–262.*

Author Profile

P.Vimalkumar is a M.E candidate in the Department of Computer Science and Engineering, P.S.R Engineering College, Sivakasi, in 2014 to 2016. His research interests include datamining.