Direct Discovery of High Utility Itemsets without Candidate Generation

Junqiang Liu
Information & Electronic Engineering
Zhejiang Gongshang University
Hangzhou, China
Email: jjliu@alumni.sfu.ca

Ke Wang
Computing Science
Simon Fraser University
Burnaby, Canada
Email: wangk@cs.sfu.ca

Benjamin C. M. Fung
Information Systems Engineering
Concordia University
Montreal, Canada
Email: benjamin.fung@concordia.ca

***Abstract*—Utility mining emerged recently to address the limitation of frequent itemset mining by introducing interestingness measures that reflect both the statistical significance and the user's expectation. Among utility mining problems, utility mining with the itemset share framework is a hard one as no anti-monotone property holds with the interestingness measure. The state-of-the-art works on this problem all employ a two-phase, candidate generation approach, which suffers from the scalability issue due to the huge number of candidates.**

This paper proposes a high utility itemset growth approach that works in a single phase without generating candidates. Our basic approach is to enumerate itemsets by prefix extensions, to prune search space by utility upper bounding, and to maintain original utility information in the mining process by a novel data structure. Such a data structure enables us to compute a tight bound for powerful pruning and to directly identify high utility itemsets in an efficient and scalable way. We further enhance the efficiency significantly by introducing recursive irrelevant item filtering with sparse data, and a lookahead strategy with dense data. Extensive experiments on sparse and dense, synthetic and real data suggest that our algorithm outperforms the state-of-the-art algorithms over one order of magnitude.

***Keywords*-Utility mining; high utility itemsets; frequent itemsets; pattern mining**

I. Introduction

Data mining is a process of discovering interesting patterns, such as itemsets, subsequences, associations, or classifiers, where interestingness measures [8][16][19] play an important role. With frequent itemset mining [2], an itemset is regarded as interesting if its occurrence frequency exceeds a user-specified threshold. Frequent itemset mining has been a research area for decades with tremendous progress having been made, among which are Apriori [2] and FP-growth [9]. Both Apriori and FP-growth employ an anti-monotone property to prune search space: A superset of an infrequent itemset is also infrequent.

However, a user's interest may relate to many factors that can not be expressed in terms of the occurrence frequency. Utility mining [19] emerged recently to address the limitation of frequent itemset mining by considering the user's expectation or goal as well as the raw data, which can be further categorized into utility mining with the itemset share framework [10][18], weighted itemset mining [5][12][14], and objective-oriented utility-based association mining [15][6]. Utility mining with the itemset share framework can be explained by the following example.

Table I
DATABASE D AND EXTERNAL UTILITY TABLE XUT

(a) D: shopping transactions

iu \ *item* / *tid*	a	b	c	d	e	f	g
t_1	1		1		1		
t_2	6	2	2			5	
t_3	1	1	1	2	6		5
t_4	3	1		4	3		
t_5	2	1		2		2	

(b) XUT: prices

item	*eu*
a	1
b	3
c	5
d	2
e	2
f	1
g	1

Running example: Consider a supermarket manager who wants to identify every combination of products with high sales revenue, i.e., high utility itemset. An itemset has high utility if the revenue on the itemset in the transactions that contain the itemset is no less than an expected level. Given shopping transactions and price per product in Table I, the revenue of {a,b} is 27 as customers who buy {a,b} spend a total of 27 on {a,b}, the revenue of {a,b,c} is 31, the revenue of {a,b,c,d} is 13, and so on. Suppose the expected revenue is 30, {a,b,c} is a high utility itemset, but {a,b} and {a,b,c,d} are not. Clearly, the anti-monotone property does not hold with the utilities (revenues) of itemsets.

Utility mining with the itemset share framework is more challenging than frequent itemset mining, due to the lack of the anti-monotone property with the interestingness measure, while it is not the case with the other two categories of utility mining [19]. Subsequently, the existing algorithms with the itemset share framework [13][11][7][3][17] all employ an anti-monotone property with TWU, namely transaction weighted utilization. TWU of an itemset is the sum of the transaction utilities of all the transactions containing the itemset. For the running example, TWU of {a,b} is the sum of the transaction utilities of $t_2 \sim t_5$, i.e., 88, TWU of {a,b,c} is that of $t_2 \sim t_3$, i.e., 57, and TWU of {a,b,c,d} is that of t_3, i.e., 30. Obviously, an itemset with a high TWU may not be a high utility itemset.

1550-4786/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDM.2012.20

Consequently, all the existing algorithms work in two phases: run an Apriori [2] based or FP-growth [9] based algorithm to find high TWU itemsets, i.e., candidates of high utility itemsets, in the first phase, and scans the raw data one more time to identify high utility itemsets from the candidates in the second phase.

The challenge is that the number of candidates can be huge, which is the scalability and efficiency bottleneck. Although FP-growth based utility mining algorithms [7][3][17] achieve a better performance than Apriori based utility mining algorithms [13][11], which is similar to the situation with frequent itemset mining, the existing FP-growth based algorithms [7][3][17] do not address this challenge when there are long transactions in raw data or the minimum utility threshold is small.

To address the challenge, this paper proposes a novel algorithm for utility mining with the itemset share framework, which directly discovers high utility itemsets in a single phase without generating candidates. The major contributions are as follows:

- A high utility itemset growth approach is proposed, which enumerates an itemset as a prefix extension of another itemset with powerful pruning. It efficiently computes the utility of each enumerated itemset and an upper bound on the utilities of prefix extensions of the itemset in order to directly identify high utility itemsets and to prune the search space. Our utility upper bound is much tighter than TWU [13], and is further tightened by recursively filtering out items irrelevant in growing high utility itemsets with sparse data.
- A lookahead strategy is incorporated with our approach, which tries to identify high utility itemsets earlier without enumeration. Such a strategy is based on a closure property and a singleton property, and enhances the efficiency in dealing with dense data.
- A novel data structure is proposed to represent original utility information in raw data, which enables efficient computation of utilities and upper bounds for enumerated itemsets. It targets the root cause of candidate generation with the existing FP-growth based algorithms [3][7][17]. Moreover, it uses less memory space than tree structures used by these algorithms.

The rest of the paper is organized as follows. Section II surveys the related works. Section III defines the problem and introduces our approach. Section IV proposes our pruning techniques. Section V discusses how to maintain utility information for efficient computation. Section VI presents our algorithm. Section VII evaluates our algorithm. Section VIII concludes the paper.

II. Related Works

Yao et al. [18] and Hilderman et al. [10] proposed the itemset share framework that takes into account weights on both attributes and attribute-value pairs. This paper falls into the same category where no anti-monotone property holds with the interestingness measure.

All existing high utility itemset algorithms with the itemset share framework employ the anti-monotone property with TWU, generate candidates, and work in two phases, except that Yao et al. [18] presented an upper bound property, i.e., the utility of a size-k itemset is no more than the average utility of its size $k-1$ subsets, which is however looser than the TWU property. They also proposed a prediction method that however may miss some high utility itemsets.

Liu et al. [13] proposed the anti-monotone property with TWU (transaction weighted utilization), and developed the TwoPhase algorithm by adapting Apriori [2] to generate a complete set of candidates with high TWUs in the first phase. Li et al. [11] improved the level-wise, multi-pass candidate generation process in the first phase by discarding isolated items to reduce the number of candidates and to shrink the database scanned in each pass.

Erwin et al. [7] proposed the CTU-PROL algorithm that uses the TWU property with FP-growth [9]. Ahmed et al. [3] proposed the IHUP$_{TWU}$ algorithm that also adapts FP-growth [9] by using a tree to maintain the TWU information of transactions. The latest, FP-growth based algorithm, UP$_{UPG}$, was proposed by Tseng et al. [17], which uses an UP-tree to maintain the revised TWU information, improves the TWU property based pruning, and thus generates less candidates in the first phase.

This paper proposes a new approach that mines high utility itemsets in a single phase without candidate generation, which fundamentally differs from and addresses the efficiency and scalability challenge with [18][13][11][7][3][17].

III. Utility Mining Problem and Our Approach

A. Utility Mining with Itemset Share Framework

Let I be the universe of items. Let D be a database of transactions $\{t_1, \cdots, t_n\}$, where each transaction $t_i \subseteq I$. Each item in a transaction is assigned a non-zero share. Each distinct item has a weight independent of any transaction, given by a table XUT. The research problem is to discover all high utility itemsets as formally defined as follows.

Definition *1:* The utility of an item i in a transaction t, denoted $u(i,t)$, is a function f of the share of i in t, $iu(i,t)$, and the weight of i independent of any transaction, $eu(i)$. That is, $u(i,t) = f(iu(i,t), eu(i))$. We also call $iu(i,t)$ the internal utility of i in t, and $eu(i)$ the external utility of i. We assume that the range of f is non-negative, i.e., $u(i,t) \geq 0$.

Although the utility function f may not be non-negative in an application, it is generally agreed that we can transform the utility function f into a non-negative function as discussed by Yao et al. [19].

Definition *2:* (a) A transaction t *contains* an itemset X if X is a subset of t, i.e., $X \subseteq t$, which means that every item i in X has a non-zero share in t, i.e., $iu(i,t) \neq 0$. (b) The *transaction set* of an itemset X, denoted $TS(X)$,

is the set of transactions that contain X. The number of transactions in $TS(X)$ is the *support* of X, denoted $s(X)$.

For the running example, the database D and the external utility table XUT are shown in Table I, and the utility function f is instantiated as the product of $iu(i,t)$ and $eu(i)$. For transaction t_1 = {a,c,e}, we have $iu(\text{a}, t_1) = 1, iu(\text{c}, t_1) = 1$, $iu(\text{e}, t_1) = 1$, $eu(\text{a}) = 1, eu(\text{c}) = 5$, and $eu(\text{e}) = 2$. Thus, $u(\text{a}, t_1) = 1$, $u(\text{c}, t_1) = 5$, and $u(\text{e}, t_1) = 2$.

DEFINITION *3:* (a) For an itemset X contained in a transaction t, the utility of X in t, denoted $u(X,t)$, is the sum of the utility of every constituent item of X in t, i.e.,

$$u(X,t) = \sum_{i \in X \subseteq t} u(i,t).$$

(b) The utility of X, denoted $u(X)$, is the sum of the utility of X in every transaction containing X, i.e.,

$$u(X) = \sum_{t \in TS(X)} u(X,t) = \sum_{t \in TS(X)} \sum_{i \in X} u(i,t).$$

DEFINITION *4:* An itemset X is a *high utility itemset*, abbreviated as HUP, if the utility of X is no less than a user-defined *minimum utility threshold*, denoted $minUtil$. High utility itemset mining is to discover the set, $HUPset$, of all high utility itemsets from a database D given an external utility table XUT and $minUtil$, i.e., $HUPset = \{X | X \subseteq I, u(X) \geq minUtil\}$

For the running example, $minUtil$ is set to 30, and we have $TS(\{\text{a,c}\}) = \{t_1, t_2, t_3\}$, $s(\{\text{a,c}\}) = 3$, $u(\{\text{a,c}\}) = u(\{\text{a,c}\}, t_1) + u(\{\text{a,c}\}, t_2) + u(\{\text{a,c}\}, t_3) = u(\text{a}, t_1) + u(\text{c}, t_1) + u(\text{a}, t_2) + u(\text{c}, t_2) + u(\text{a}, t_3) + u(\text{c}, t_3) = 28$. Thus, {a,c} is not a high utility itemset while {a,b,c} is as $u(\{\text{a,b,c}\}) = 31$. Consequently, $HUPset$ is { {a,b,c}, {a,b,d}, {a,d,e}, {a,b,d,e}, {b,d,e}, {d,e}, {a,b,c,d,e,g} }.

B. High Utility Itemset Growth Approach

We propose a high utility itemset growth approach. The basic idea is to enumerate an itemset as a prefix extension of another itemset, to determine if the enumerated itemset is a high utility itemset by efficiently computing its utility, and to prune the prefix extensions of the enumerated itemset if an upper bound on the utilities of the prefix extensions is less than $minUtil$.

In order to avoid repetitive enumeration of itemsets, an ordering of items is imposed, with which an itemset can also be represented as an ordered sequence. For brevity, we use the set notation, e.g., {a,b,c}, in place of the sequence notation, e.g., <a,b,c>.

DEFINITION *5:* The *imposed ordering of items*, denoted Ω, is a pre-determined, ordered sequence of all the items in I. Accordingly, for items i and j, $i \prec j$ denotes that i is listed before j; $i \prec X$ denotes that $i \prec j$ for every $j \in X$, and $W \prec X$ denotes that $i \prec X$ for every $i \in W$, in accordance with Ω.

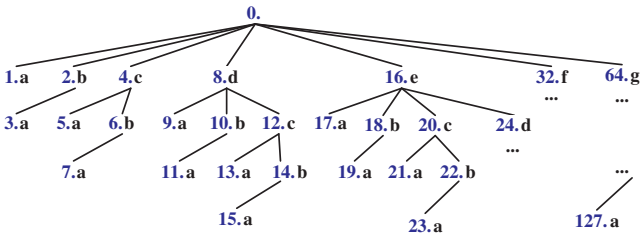

Figure 1. Prefix extension tree where each node is numbered in the order of depth-first search and is referred to by such a number.

DEFINITION *6:* Given an ordering Ω, an itemset Y is a *prefix extension* of an itemset X, if X is a suffix of Y, i.e., if $Y = W \cup X$ for some W with $W \prec X$ in Ω.

Our basic approach can be thought of as growing or searching a prefix extension tree in a depth-first manner [1][4]. On the prefix extension tree, the root is labelled by no item, each node N other than the root is labelled by an item, denoted $item(N)$, the path from N to the root represents an itemset, denoted $pat(N)$, and the child nodes of N are labelled by items listed *before* $item(N)$ in the imposed ordering Ω. Consequently, any prefix extension of $pat(N)$ is represented by a node in the subtree rooted at N. Note that only the branch currently searched is materialized in memory.

For the running example, Ω = {a,b,c,d,e,f,g}, and the prefix extension tree is shown in Figure 1 where {} is enumerated by the root, i.e., Node 0, {a} and {b} are enumerated as prefix extensions of {} by the children of the root, i.e., Node 1 and Node 2 respectively, {a,b} is enumerated as a prefix extension of {b} by Node 3, and so on. The whole tree is made of $2^7 = 128$ nodes.

The construction of our prefix extension tree differs from the regular set enumeration tree employed by [1][9][4] in that our prefix extension tree maintains a property: An itemset is always enumerated before its supersets in a depth-first search, e.g., {a} and {b} before {a,b}, and {a,b} and {c} before {a,b,c}, as shown in Figure 1. The regular set enumeration tree does not observe such a property, e.g., {a,b} before {b}, and {a,b,c} before{b,c} and {c}.

Most importantly, the construction of our prefix extension tree has two benefits. First, it enables the efficient and scalable computation of the transaction set supporting each enumerated itemset as in Section V. Second, it increases the likelihood for strong pruning proposed in the next section.

IV. PRUNING ITEMSETS IN ENUMERATION PROCESS

This section proposes strong pruning techniques, which lays the theoretical foundation for our approach and is critical to the efficiency.

A. Pruning by Utility Upper Bounding

We estimate an upper bound on utilities of prefix extensions of an itemset X based on original utility infor-

mation. Such a bound is tighter than bounds based on TWU [13][11][7][3][17]. If the upper bound is less than $minUtil$, all the prefix extensions of X are not high utility itemsets and can be pruned without enumeration.

DEFINITION *7:* Given an ordering Ω, an itemset Y is the *full prefix extension* of an itemset X w.r.t. a transaction t containing X, denoted $Y = fpe(X,t)$, if Y is a prefix extension of X derived by adding exactly all the items in t that are listed before X in Ω, i.e., if $Y = W \cup X$ with $W = \{i | i \in t \wedge i \prec X \wedge X \subseteq t\}$.

For the running example, the full prefix extensions of {c} w.r.t. t_1 and t_2 are $fpe(\{\text{c}\}, t_1) = \{\text{a,c}\}$ and $fpe(\{\text{c}\}, t_2) = \{\text{a,b,c}\}$ respectively.

THEOREM *1:* (Basic upper bounds) For an itemset X, the sum of the utility of the full prefix extension of X w.r.t. each transaction in $TS(X)$, denoted $uB_{fpe}(X)$, is no less than the utility of any prefix extension Y of X, that is,

$$uB_{fpe}(X) = \sum_{t \in TS(X)} u(fpe(X,t), t) \geq u(Y) \quad (1)$$

Proof: The premise, Y is a prefix extension of X, means $X \subseteq Y$, and thus has two implications. First, $TS(Y) \subseteq TS(X)$. Second, $\forall t \in TS(Y)$, $Y \subseteq fpe(X,t)$. As the utility function is non-negative, we have

$$uB_{fpe}(X) = \sum_{t \in TS(X)} u(fpe(X,t), t)$$

$$\geq \sum_{t \in TS(Y)} u(fpe(X,t), t) \geq \sum_{t \in TS(Y)} u(Y,t) = u(Y)$$

■

For example, when enumerating {} by Node 0 in Figure 1, we get $TS(\{\text{a}\}) = D$ and $uB_{fpe}(\{\text{a}\}) = u(\{\text{a}\}) = 15 < minUtil$, $TS(\{\text{b}\}) = \{t_2, t_3, t_4, t_5\}$ and $uB_{fpe}(\{\text{b}\}) = u(\{\text{a,b}\}) = 27 < minUtil$, $TS(\{\text{c}\}) = \{t_1, t_2, t_3\}$ and $uB_{fpe}(\{\text{c}\}) = u(\{\text{a,c}\}, t_1) + u(\{\text{a,b,c}\}, t_2) + u(\{\text{a,b,c}\}, t_3) = 37 > minUtil$, and so on. Thus, we prune Nodes 1 and 2 (with Node 3), and create Nodes 4, 8, 16, 32, and 64.

Clearly, the tighter the upper bound, the stronger the pruning. An observation is that many items never occur in high utility itemsets when raw data are sparse. It is possible to exclude them to tighten the upper bound.

COROLLARY *2:* (Relevance of an item) For an itemset X and an item $i \prec X$, the sum of the utility of the full prefix extension of X w.r.t. every transaction in $TS(\{i\} \cup X)$, denoted $uB_{item}(i, X)$, is no less than the utility of a prefix extension Y of X that contains i, that is,

$$uB_{item}(i, X) = \sum_{t \in TS(\{i\} \cup X)} u(fpe(X,t), t) \geq u(Y) \quad (2)$$

Proof: The extra premise in addition to Theorem 1 is that $i \subseteq Y$, which results in that $\{i\} \cup X \subseteq Y$. In the light of Theorem 1, we get this corollary. ■

Corollary 2 states that an item $i \prec X$ is *irrelevant* to any high utility itemset that is a prefix extension of X if $uB_{item}(i, X) < minUtil$, and can be ignored in enumerating prefix extensions of X.

For example, when enumerating {d,e} by Node 24 in Figure 1, we have $uB_{item}(\text{a}, \{\text{d,e}\}) = uB_{item}(\text{b}, \{\text{d,e}\}) = u(\{\text{a,b,c,d,e}\}, t_3) + u(\{\text{a,b,d,e}\}, t_4) = 45 > minUtil$, and $uB_{item}(\text{c}, \{\text{d,e}\}) = u(\{\text{a,b,c,d,e}\}, t_3) = 25 < minUtil$. Thus, items a and b are relevant, and item c is irrelevant in growing prefix extensions of {d,e}.

Furthermore, we can apply Corollary 2 recursively as excluding an irrelevant item may decrease uB_{item} and uB_{fpe} of other items.

COROLLARY *3:* (Tighter upper bounds) For an itemset X and its prefix extension Y that is relevant in growing high utility itemsets, a tighter upper bound on the utility of Y is

$$uB'_{fpe}(X) = \sum_{t \in TS(X)} u(fpe'(X,t), t) \geq u(Y) \quad (3)$$

where $fpe'(X,t)$ is derived from $fpe(X,t)$ by excluding all irrelevant items $i \prec X$ by Corollary 2.

For example, as item c is irrelevant in growing prefix extensions of {d,e} enumerated by Node 24 in Figure 1, we compute uB_{item} the second time by excluding item c, which yields $uB_{item}(\text{a}, \{\text{d,e}\}) = uB_{item}(\text{b}, \{\text{d,e}\}) = u(\{\text{a,b,d,e}\}, t_3) + u(\{\text{a,b,d,e}\}, t_4) = 40$. The bounds get tighter though the set of relevant items does not shrink.

B. Lookahead with Closure and Singleton Properties

In addition to pruning by upper bounding, we look ahead to the supersets of the itemset currently enumerated in order to identify high utility itemsets earlier without enumeration, which is made possible with the construction of the prefix extension tree. Concretely, our lookahead strategy covers the following two cases.

Case 1: Similar to closed frequent itemsets [20], it is possible that every prefix extension (superset) of an itemset has a utility over $minUtil$. Identifying such closed high utility itemsets without enumeration improves the efficiency.

THEOREM *4:* (Closure property) For an itemset X and a set W of items with $X \cap W = \emptyset$, if $s(\{i\} \cup X) = s(X)$ and $u(\{i\} \cup X) \geq minUtil$ for all $i \in W$, then

$$u(S \cup X) \geq minUtil, \forall S \subseteq W \wedge S \neq \emptyset. \quad (4)$$

For example, when enumerating {d,e} by Node 24 in Figure 1, we get $s(\{\text{a}\} \cup \{\text{d,e}\}) = s(\{\text{b}\} \cup \{\text{d,e}\}) = s(\{\text{d,e}\}) = 2$, and $u(\{\text{a}\} \cup \{\text{d,e}\}) = 34 > minUtil$ and $u(\{\text{b}\} \cup \{\text{d,e}\}) = 36 > minUtil$ while items a and b are relevant items and item c is not. Therefore, we know that all the prefix extensions of {d,e} with relevant items a and b, enumerated by Nodes 24-27, are high utility itemsets without creating the rest of the subtree rooted at Node 24.

Case 2: A supplement to the closure property is that although many items are relevant to prefix extensions of an itemset, there is only one high utility itemset among

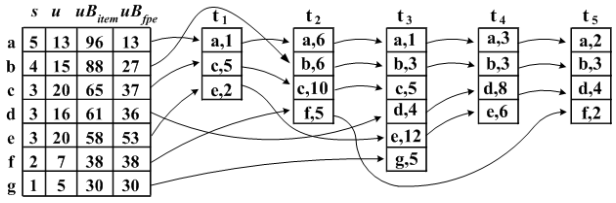

Figure 2. $TS_{caul}(\{\})$: CAUL representing transaction set $TS(\{\})$

all the prefix extensions. Identifying such a case without enumeration also enhances the efficiency.

THEOREM *5:* (Singleton property) For an itemset X and a set W of items with $X \cap W = \emptyset$, if $s(\{i\} \cup X) = s(X)$ for all $i \in W$ and

$$minUtil \leq u(W \cup X) < minUtil + \min_{j \in W} \sum_{t \in TS(X)} u(\{j\}, t)$$

then

$$u(S \cup X) < minUtil, \forall S \subset W. \quad (5)$$

For example, when enumerating {g} by Node 64 in Figure 1, we know items a, b, c, d, and e are relevant, and $s(\{\text{a}\} \cup \{\text{g}\}) = s(\{\text{b}\} \cup \{\text{g}\}) = s(\{\text{c}\} \cup \{\text{g}\}) = s(\{\text{d}\} \cup \{\text{g}\}) = s(\{\text{e}\} \cup \{\text{g}\}) = s(\{\text{g}\}) = 1$, and $u(\{\text{a,b,c,d,e}\} \cup \{\text{g}\}) = 30 = minUtil$. Clearly, the condition of Theorem 5 holds. Thus, we know that {a,b,c,d,e,g} is a high utility itemset and all its proper subsets are not, without creating the rest of the subtree rooted at Node 64.

V. ENABLING TECHNIQUE

What facilitates our approach is a novel yet simple structure, CAUL, namely Chain of Accurate Utility Lists. CAUL maintains the utility information in the transaction set $TS(X)$ for each enumerated itemset X, denoted $TS_{caul}(X)$, by utility lists and a summary table.

For each transaction $t \in TS(X)$, all relevant items in t with their utilities are stored in a utility list. The summary table maintains an entry for each distinct item i relevant in growing prefix extensions of X, which consists of 4 fields:

- s for $s(\{i\} \cup X)$, i.e., the support of $\{i\} \cup X$;
- u for $u(\{i\} \cup X)$, i.e., the utility of $\{i\} \cup X$;
- uB_{item} for $uB_{item}(i, X)$ by Corollary 2;
- uB_{fpe} for $uB_{fpe}(\{i\} \cup X)$ by Theorem 1, Corollary 3.

For the running example, Figure 2 shows $TS_{caul}(\{\})$ for the empty itemset $\{\}$ represented by Node 0 in Figures 1. The first list represents t_1 with its first element storing item a and $u(\text{a}, t_1) = 1$, its second element storing item c and $u(\text{c}, t_1) = 5$, and so on. The occurrences of item a in all the five lists are threaded by the chain starting from the first entry of the summary table. The first summary entry also stores $s(\{\text{a}\})$, $u(\{\text{a}\})$, $uB_{item}(\text{a}, \{\})$, and $uB_{fpe}(\{\text{a}\})$.

CAUL keeps original utility information for each transaction, which targets the root cause with the existing FP-growth based algorithms [3][7][17], that is, they all employ a tree structure to maintain the TWU information instead, and thus can only determine the candidacy of an itemset but not the actual utility of the itemset in the first phase.

More importantly, when depth-first searching the prefix extension tree, for any node N and its parent node P, $TS_{caul}(pat(N))$ is embedded in and can be derived from $TS_{caul}(pat(P))$ by a pseudo projection.

The detail can be found in the full version of the paper.

VI. UTILITY MINING ALGORITHM

This section presents our algorithm, d^{2}HUP, namely Direct Discovery of High Utility Patterns.

d^{2}HUP searches the prefix extension tree in a depth-first manner. When visiting a node N, it first computes utilities and upper bounds for the children of N by building a pseudo CAUL, make a materialized copy of CAUL if a space-time tradeoff is beneficial, outputs each child whose utility is no less than $minUtil$ as a high utility itemset, depth-first searches each child whose upper bound is no less than $minUtil$, and then purges the subtree rooted at N and continues with the next sibling of N.

Algorithm 1 d^{2}HUP($D, XUT, minUtil$)
1 create the root of prefix extension tree
2 build $TS_{caul}(\{\})$ with Ω in descending order of uB_{item}
3 $N \leftarrow$ the root
4 $W \leftarrow \{i | i \prec pat(N) \wedge uB_{item}(i, pat(N)) \geq minUtil\}$
5 **if** $\forall i \in W$, $s(pat(N)) = s(\{i\} \cup pat(N)) \wedge$ $u(\{i\} \cup pat(N)) \geq minUtil$ **then** output each non-empty subset of $W \cup pat(N)$ as an HUP, **goto** step 12
6 $\Delta \leftarrow$ minimum of $u(\{j\} \cup pat(N)) - u(pat(N)), \forall j \in W$
7 **if** $\forall i \in W$, $s(pat(N)) = s(\{i\} \cup pat(N)) \wedge$ $minUtil \leq u(W \cup pat(N)) < minUtil + \Delta$ **then** output $W \cup pat(N)$ as an HUP, **goto** step 12
8 **foreach** item $i \in W$ **do**
9 **if** $u(\{i\} \cup pat(N)) \geq minUtil$ **then** output $\{i\} \cup pat(N)$
10 **if** $uB_{fpe}(\{i\} \cup pat(N)) \geq minUtil$ **then** create a child node of N for i
11 **end foreach**
12 **while** N is not *null* and has no child **do**
13 $P \leftarrow parent(N)$, delete N, $N \leftarrow P$
14 **end while**
15 **if** N is *null* **then stop**
else $P \leftarrow N$, $N \leftarrow firstChild(N)$
16 build $TS_{caul}(pat(N))$ by projection, **goto** Step 4

The pseudo code of d^{2}HUP is shown in Algorithm 1.

First, d^{2}HUP creates the root of prefix extension tree (line 1), builds $TS_{caul}(\{\})$ by scanning the database D and the external utility table XUT to compute $s(\{i\})$, $u(\{i\})$, $uB_{item}(i, \{\})$, and $uB_{fpe}(\{i\})$ for each item i (line 2), and starts the depth-first search from the root node (line 3).

Second, d^{2}HUP makes the set W of relevant items by Corollary 2 for the node N currently being visited (line 4).

If the closure property holds, d^{2}HUP outputs every prefix extension of $pat(N)$ with relevant items as a high utility itemset (line 5). If the singleton property holds, d^{2}HUP outputs the union of all the relevant items and $pat(N)$ as a high utility itemset (lines 6 - 7).

Third, for each relevant item $i \in W$, d^{2}HUP outputs $\{i\} \cup pat(N)$ as a high utility itemset if $u(\{i\} \cup pat(N)) \geq minUtil$, and creates a child node C of N with $item(C) \leftarrow i$ and $pat(C) \leftarrow \{i\} \cup pat(N)$, if $uB_{fpe}(\{i\} \cup pat(N)) \geq minUtil$ (lines 8 - 11).

Fourth, d^{2}HUP continues with the next node in the depth-first order (lines 12 - 15), and computes $s(\{j\} \cup pat(N))$, $u(\{j\} \cup pat(N))$, $uB_{item}(j, pat(N))$, and $uB_{fpe}(\{j\} \cup pat(N))$ for each item j in $TS_{caul}(\{pat(N)\})$ by projecting from $TS_{caul}(pat(P))$ (line 16).

For the running example, d^{2}HUP only enumerates the nodes 0, 4, 6, 8, 10, 16, 24, 32, and 64 in Figure 1 to find all the high utility itemsets.

VII. Experimental Evaluation

We evaluate the efficiency and scalability of our d^{2}HUP algorithm by comparing with the state-of-the-art algorithms, TwoPhase [13], IHUP$_{TWU}$ [3], and UP$_{UPG}$ [17].

Extensive experiments on sparse and dense, synthetic and real data suggest that our d^{2}HUP is up to 1 order, up to 2 orders, and up to 3 orders of magnitude more efficient than UP$_{UPG}$, IHUP$_{TWU}$, and TwoPhase respectively. This is because each latter algorithm generates a large number of candidates, its first phase already takes more time than d^{2}HUP, and it needs a second phase while d^{2}HUP does not.

The detail can be found in the full version of the paper.

VIII. Conclusion

This paper proposes a new algorithm, d^{2}HUP, for utility mining with the itemset share framework, which directly discovers high utility itemsets without candidate generation. d^{2}HUP outperforms the state-of-the-art algorithms over one order of magnitude. The novelty lies in the following.

- A novel data structure, CAUL, is proposed, which targets the root cause of candidate generation with the existing approaches, and is our enabling technique.
- A high utility itemset growth approach is presented, which integrates an itemset enumeration strategy by prefix extensions, and strong pruning based on utility upper bounding.
- The efficiency of our approach is enhanced significantly by recursive irrelevant item filtering with sparse data, and by the lookahead strategy with dense data.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (61272306), the Zhejiang Provincial Natural Science Foundation of China (LY12F02024), and the Zhejiang Provincial Human Resources and Social Security Bureau of China (2011-443-3).

References

[1] R. Agarwal, C. Aggarwal, V. Prasad. *Depth first generation of long patterns*. In SIGKDD, pp.108–118, 2000.

[2] R. Agrawal and R. Srikant. *Fast algorithms for mining association rules*. In VLDB, pp.487–499, 1994.

[3] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, Y.-K. Lee. *Efficient tree structures for high utility pattern mining in incremental databases*. IEEE TKDE, 21(12):1708–1721, 2009.

[4] D. Burdick, M. Calimlim, J. Gehrke. *MAFIA: A maximal frequent itemset algorithm for transactional databases*. In ICDE, pp. 443-452, 2001.

[5] C. H. Cai, A. W. C. Fu, C. H. Cheng, W. W. Kwong. M*ining association rules with weighted items*. In Int'l Database Engineering & Applications Symposium. IEEE, pp.68–77, 1998.

[6] R. Chan, Q. Yang, Y. Shen. *Mining high utility itemsets*. In ICDM, pp.19–26. IEEE, 2003.

[7] A. Erwin, R. P. Gopalan, N. R. Achuthan. *Efficient mining of high utility itemsets from large datasets*. In PAKDD, 2008.

[8] L. Geng and H. J. Hamilton. *Interestingness measures for data mining*. ACM Computing Surveys, 38(3):9, 2006.

[9] J. Han, J. Pei, Y. Yin. *Mining frequent patterns without candidate generation*. In SIGMOD, pp.1–12. ACM, 2000.

[10] R. J. Hilderman, C. L. Carter, H. J. Hamilton, N. Cercone. *Mining market basket data using share measures and characterized itemsets*. In PAKDD, pp.72–86, 1998.

[11] Y.-C. Li, J.-S. Yeh, C.-C. Chang. *Isolated items discarding strategy for discovering high utility itemsets*. Data & Knowledge Engineering, 64(1):198–217, 2008.

[12] T. Y. Lin, Y. Y. Yao, E. Louie. *Value added association rules*. In *PAKDD*, pp.328–333, 2002.

[13] Y. Liu, W. Liao, A. Choudhary. *A fast high utility itemsets mining algorithm*. In Utility-Based Data Mining in KDD, 2005.

[14] S. Lu, H. Hu, F. Li. *Mining weighted association rules*. Intelligent Data Analysis, 5(3):211–225, 2001.

[15] Y. Shen, Q. Yang, Z. Zhang. *Objective-oriented utility-based association mining*. In ICDM, pp.426-433, 2002.

[16] P. N. Tan, V. Kumar, J. Srivastava. *Selecting the right objective measure for association analysis*. Information Systems, 29(4):293–313, 2004.

[17] V. S. Tseng, C.-W. Wu, B.-E. Shie, P. S. Yu. *Up-growth: An efficient algorithm for high utility itemset mining*. In SIGKDD, pp.253–262. ACM, 2010.

[18] H. Yao, H. J. Hamilton, C. J. Butz. *A foundational approach to mining itemset utilities from databases*. In SDM, 2004.

[19] H. Yao, H. J. Hamilton, L. Geng. *A unified framework for utility-based measures for mining itemsets*. In Utility-Based Data Mining in SIGKDD, pp.28–37. ACM, 2006.

[20] M. J. Zaki and C. Hsiao. *Charm: An efficient algorithm for closed itemset mining*. In SDM, pp.457-473. SIAM, 2002.