Mining High-Utility Itemsets with Both Positive and Negative Unit Profits from Uncertain Databases

Conference Paper *in* Lecture Notes in Computer Science · April 2017

DOI: 10.1007/978-3-319-57454-7_34

5 authors, including:

Wensheng Gan
Jinan University (Guangzhou, China)

Philippe Fournier Viger
Shenzhen University

Mining High-Utility Itemsets with Both Positive and Negative Unit Profits from Uncertain Databases

Wensheng Gan[1], Jerry Chun-Wei Lin[1(✉)], Philippe Fournier-Viger[2], Han-Chieh Chao[1,3], and Vincent S. Tseng[4]

[1] School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China
wsgan001@gmail.com, jerrylin@ieee.org, hcc@ndhu.edu.tw

[2] School of Natural Sciences and Humanities, Harbin Institute of Technology (Shenzhen), Shenzhen, China
philfv@hitsz.edu.cn

[3] Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan

[4] Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
vtseng@cs.nctu.edu.tw

Abstract. Some important limitation of frequent itemset mining are that it assumes that each item cannot appear more than once in each transaction, and all items have the same importance (weight, cost, risk, unit profit or value). These assumptions often do not hold in real-world applications. For example, consider a database of customer transactions containing information about the purchase quantities of items in each transaction and the positive or negative unit profit of each item. Besides, uncertainty is commonly embedded in collected data in real-life applications. To address this issue, we propose an efficient algorithm named HUPNU (mining High-Utility itemsets with both Positive and Negative unit profits from Uncertain databases), the high qualified patterns can be discovered effectively for decision-making. Based on the designed vertical PU$^{\pm}$-list (Probability-Utility list with Positive-and-Negative profits) structure and several pruning strategies, HUPNU can directly discovers the potential high-utility itemsets without generating candidates.

Keywords: Frequent itemset · Uncertainty · Negative unit profit · PU$^{\pm}$-list

1 Introduction

Frequent itemset mining (FIM) [1,3] has become one of the core data mining tasks that is essential to a wide range of applications. However, some important limitations of FIM are that it assumes that each item cannot appear more than once in each transaction and that all items have the same importance (weight, cost, risk, unit profit or value). These assumptions often do not hold

J. Kim et al. (Eds.): PAKDD 2017, Part I, LNAI 10234, pp. 434–446, 2017.
DOI: 10.1007/978-3-319-57454-7_34

in real-world applications. For example, consider a database of customer transactions containing information about the purchase quantities of items in each transaction and the unit profit of each item. All the developed FIM algorithms would discard this information and may thus discover many frequent itemsets generating a low profit. Hence, FIM fails to discover high profit patterns for many real-world applications.

To address this issue, the problem of high-utility itemset mining (HUIM) was developed [6,15]. HUIM considers the case where items can appear more than once in each transaction and where each item has a user-specified "utility" (e.g., unit profit). The goal of HUIM is to discover items/itemsets with their utility in a database is no less than the minimum utility threshold, called the high utility itemsets (HUIs), i.e., itemsets generating a high profit. HUIM plays an important role in a wide range of applications, such as website click stream analysis, cross-marketing in retail stores, and biomedical applications [4,9,14]. The problem of HUIM is more difficult than FIM, the reason is that the well-known *downward-closure property* of the support of an itemset is no longer hold in HUIM. In HUIM, however, the utility of an itemset is neither monotonic or anti-monotonic, it means that a high utility itemset may have its supersets or subsets with lower, equal or higher utility [3]. Thus, it is very difficult to prune the search space in HUIM. Many studies have been carried to develop efficient HUIM algorithms, such as Two-Phase [10], IHUP [4], UP-Growth [14], HUI-Miner [9], and FHM [8], etc.

However, these algorithms are designed under a assumption that all items having positive unit profits in a database, they cannot be applied to handle items having negative unit profits, despite that such items occur in many real-life transaction databases. For example, it is common that retail stores or supermarket sell items at a loss (e.g., printers) to stimulate the sale of other related items (e.g., proprietary printer cartridges). Although giving away a unit of some items results in a loss for supermarkets, they could provide opportunities for cross-selling and could possibly earn more money from the promotion. It was demonstrated that if classical HUIM algorithms are applied on databases containing items with negative unit profits, they can generate an incomplete set of HUIs [7]. The HUINIV-Mine [7] and FHN [12] were developed to handle the problem of HUIM with negative unit profits. In real-life applications, uncertainty is common seen when data is collected from noisy data sources such as RFID, GPS, wireless sensors, and WiFi systems [2,5]. Some algorithms of FIM have been developed to discover useful information in uncertain databases. Since utility and uncertainty are two different measures for an object (e.g., an useful pattern). The utility is a semantic measure (how "utility" of a pattern is based on the user's priori knowledge and goals), while uncertainty is an objective measure (the probability of a pattern is an objective existence). Up to now, most algorithms of HUIM have been extensively developed to handle precise data, but they are not suitable to handle the data with uncertainty. It may be useless or misleading if the discovered results of HUIs with low existential probability [11].

In light of these, in this paper, we attempt to design en efficient algorithm to discover high-utility itemsets from uncertain transaction databases by considering both positive and negative unit profits. This algorithm is named **HUPNU** (mining **H**igh-**U**tility itemsets with both **P**ositive and **N**egative unit profits from **U**ncertain databases) to mine HUIs. To the best of our knowledge, it is the first work to address this problem. The contributions of this paper are described below. (1) A vertical list structure, called PU$^{\pm}$-list (Probability-Utility list with Positive-and-Negative profits), is designed to store all the necessary information for the database. (2) A one-phase efficient algorithm named HUPNU is proposed to mine HUIs without multiple time-consuming database scan. It relies on a series of PU$^{\pm}$-lists to directly mine HUIs without generating and testing candidates. (3) Several efficient pruning strategies are further proposed to reduce the search space, a number of unpromising itemsets can be early pruned when constructing the PU$^{\pm}$-list. (4) An extensive experimental study carried on several real-life datasets shows that the complete set of HUIs can be efficiently discovered by the proposed HUPNU algorithm.

2 Preliminaries and Problem Definition

Definition 1. Let I be a set of items (symbols). An *uncertain transaction database* is a set of uncertain transactions $D = \{T_1, T_2, \ldots, T_n\}$ such that for each transaction $T_c \in I$, and T_c has a unique identifier c called its *tid*. As the attribute uncertainty model [2,5], each item i has a unique probability of existence $p(i, T_c)$. Each item $i \in I$ is associated with a positive or negative value $pr(i)$, called its external utility (e.g., unit profit). For each T_c such that $i \in T_c$, a positive number $q(i, T_c)$ is called the internal utility of i (e.g., purchase quantity). Each item i_m in D has a unique profit $pr(i_m)$, they are provided in a profit table and denoted as $ptable$ = $\{pr(i_1)$, $pr(i_2)$, ..., $pr(i_m)\}$.

Table 1. An example uncertain quantitative database.

tid	Transaction (item: quantity, probability)	*TU*	*RTU*
T_1	(b:3, 0.85); (c:1, 1.0); (d:2, 0.70)	14	24
T_2	(a:1, 1.0); (b:1, 0.60); (c:3, 0.75); (e:1, 0.40)	19	19
T_3	(a:1, 0.55); (b:2, 0.60); (c:4, 1.0); (d:1, 0.90); (e:5, 0.40)	34	39
T_4	(b:3, 0.90); (d:1, 0.45)	16	21
T_5	(a:4, 1.0); (c:3, 0.85); (d:2, 0.70); (e:2, 0.45)	23	33

Example 1. Consider the running example w.r.t. Table 1, it contains five transactions ($T_1, T_2, \ldots, T_5$). Transaction T_1 indicates that items (b)1, (c), and (d) appear in T_1 with purchase quantity as 3, 1, and 2, respectively. And assume that the unit profit of (a) to (e) are respectively defined as: $\{pr(a)$:6, $pr(b)$:7, $pr(c)$:1, $pr(d)$:-5, $pr(e)$:3$\}$. Thus, item (d) is sold at loss.

Definition 2. The utility of an item i in a transaction T_c is denoted as $u(i, T_c)$ and defined as $pr(i) \times q(i, T_c)$. The utility of an itemset X (a group of items $X \subseteq I$) in T_c is denoted as $u(X, T_c)$ and defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$. The utility of an itemset X in a database D is denoted as $u(X)$, it can be calculated as $u(X) = \sum_{X \subseteq T_c \wedge T_c \in D} u(X, T_c)$.

Example 2. The utility of item (e) in T_2 is $u(e, T_2) = 3 \times 1 = 3$. The utility of the itemset $\{a, e\}$ in T_2 is $u(\{a, e\}, T_2) = u(a, T_2) + u(e, T_2) = 6 \times 1 + 3 \times 1 = 9$. The utility of the itemset $\{a, e\}$ is $u(\{a, e\}) = (u(a, T_2) + u(e, T_2)) + (u(a, T_3) + u(e, T_3)) + (u(a, T_5) + u(e, T_5)) = (6+3) + (6+15) + (24+6) = 60$. The utility of the itemset $\{a, d, e\}$ is $u(\{a, d, e\}) = (u(a, T_3) + u(d, T_3)) + u(e, T_3)) + (u(a, T_5) + u(d, T_5)) + u(e, T_5)) = (6 + (-5) + 15) + (24 + (-10) + 6) = 36$.

Definition 3. The probability of an itemset X (a group of items $X \subseteq I$) in T_c is denoted as $p(X, T_c)$ and defined as $p(X, T_c) = \prod_{i \in X} p(i, T_c)$. The probability of X in D is denoted as $Pro(X)$ and defined as $Pro(X) = \sum_{T_c \in D}(\prod_{i \in X} p(i, T_c))$.

Example 3. The probability of item (e) in T_2 is $p(e, T_2) = 0.40$. The probability of the itemset $\{a, e\}$ in T_2 is $p(\{a, e\}, T_2) = p(a, T_2) \times p(e, T_2) = 1.0 \times 0.40 = 0.40$. The probability of item (e) in D is $Pro(e) = 1.25$. The probability of the itemset $\{a, d, e\}$ in D is $p(\{a, d, e\}) = p(ade, T_3) + p(ade, T_5) = 0.198 + 0.315 = 0.513$.

Definition 4. An itemset X in an uncertain database D is said to be a potential high-utility itemset (PHUI) if it satisfies the following two conditions: (1) $u(X) \geq minUtil$, and (2) $Pro(X) \geq minPro \times |D|$. A PHUI is thus an itemset having both a high expected/potential probability and a high utility value.

The problem of mining high-utility itemsets with both positive and negative unit profits from uncertain databases is to discover all potential high-utility itemsets (having a high expected/existential probability and a high utility) in an uncertain database where external utility values may be positive or negative.

Example 4. If the user-specified $minPro = 0.20$ and $minUtil = 20$, ten PHUIs should be found in the running example database. They are ($\{a\}$:36, 2.55; $\{b\}$:63, 2.95; $\{e\}$:24, 1.25; $\{a, c\}$:46, 2.15; $\{a, e\}$:60, 1.07; $\{b, c\}$:52, 1.90; $\{b, d\}$:36, 1.54; $\{c, e\}$:34, 1.0825; $\{a, c, d\}$:22, 1.09; $\{b, c, d\}$:27, 1.135). $\{\{\text{a}\}: 36, 2.55\}$ means that the utility of $\{a\}$ is 36, and its expected probability is 2.55.

3 Proposed HUPNU Algorithm

3.1 Properties of Positive and Negative Unit Profits

According to the previous studies, the utility measure is not monotonic or anti-monotonic [9,10,14]. In other words, an itemset may have a utility lower, equal or higher than those of any of its subsets. To handle the problem for mining HUIs with both positive and negative unit profits, the HUINIV-Mine [7] and FHN [12] algorithms were developed by redefining the notion of transaction utility (TU) and the TWU measure [10] as follows.

Definition 5. The $TU(T_c) = \sum_{i \in T_c} u(i, T_c)$, but the *redefined transaction utility* (RTU) of T_c is defined as $RTU(T_c) = \sum_{i \in T_c \wedge pr(i)>0} u(i, T_c)$. The *redefined transaction-weighted utilization* ($RTWU$) of X is defined $RTWU(X) = \sum_{X \subseteq T_c \wedge T_c \in D} RTU(T_c)$. Thus, $RTWU(X) \geq u(X)$.

Example 5. Table 1 shows the TU and RTU of five transactions. Consider itemsets $\{a, e\}$ and $\{a, d, e\}$, the $RTWU(\{a, e\}) = 91$ and $RTWU(\{a, d, e\}) = 71$, which are over-estimations of $u(\{a, e\}) = 60$ and $u(\{a, d, e\}) = 36$.

Let $pu(X)$ and $nu(X)$ respectively denotes the sum of positive utilities and negative utilities of items in X in a transaction (or in a database). Since $u(X) = pu(X) + nu(X)$, the relationship $nu(X) \leq u(X) \leq pu(X)$ holds [12]. Thus, both $u(X)$ and $nu(X)$ cannot be used to overestimate the utility of an itemset. Although $pu(X)$ for an itemset is an upper-bound on utility, it still does not hold the downward closure of extensions with positive or negative items.

3.2 Probability-Utility List with Positive-and-Negative Profits

Definition 6. In the designed HUPNU algorithm, we define the total processing order $\succ$ such that (1) items are sorted in $RTWU$-ascending order, and (2) negative items always succeed all positive items.

Definition 7. The PU$^{\pm}$-list of an itemset X in an uncertain database D is denoted as *X.PUL*. It consisted of a set of tuples, $<tid, pro, pu, nu, rpu>$ for each transaction T_{tid} containing X. For each tuple, (1) The *tid* element is the transaction identifier; (2) The *pro* element is the existential probability of X in T_{tid}, i.e., $pro(X, T_{tid}) \geq 0$; (3) The *pu* element is the positive utility of X in T_{tid}, i.e., $u(X, T_{tid}) \geq 0$; (4) The *nu* element is the negative utility of X in T_{tid}, i.e., $u(X, T_{tid}) < 0$; (5) The *rpu* element is defined as $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid}) \geq 0$, such that only positive utility values of the remaining items.

Example 6. The search space of HUPNU can be represented as a PU$^{\pm}$-list based Set-enumeration tree [13], we named it as PU$^{\pm}$-tree. Since $\{RTWU(a)$: 91; $RTWU(b)$: 103; $RTWU(c)$: 115; $RTWU(d)$: 117; $RTWU(e)$: 91;$\}$, the designed processing order $\succ$ in PU$^{\pm}$-list is $\{a \succ e \succ b \succ c \succ d\}$, we have $\{a\}.PUL = \{(T_2, 1.0, 6, 0, 13), (T_3, 0.55, 6, 0, 33), (T_5, 1.0, 24, 0, 9)\}$; $\{d\}.PUL = \{(T_2, 0.70, 0, -10, 0), (T_3, 0.090, 0, -5, 0), (T_4, 0.45, 0, -5, 0), (T_5, 0.70, 0, -10, 0)\}$; $\{a, d\}.PUL = \{T_3, 0.495, 6, -5, 0), (T_5, 0.70, 24, -10, 0)\}$.

Definition 8. Let *SUM(X.iu)*, *SUM(X.pu)*, *SUM(X.nu)*, and *SUM(X.rpu)* are respectively the sum of the utilities, the sum of *pu* values, the sum of *nu* values and the sum of *rpu* in the PU$^{\pm}$-list of X, that are: $SUM(X.pu) = \sum_{X \in T_c \wedge T_c \subseteq D} X.pu(T_c)$; $SUM(X.nu) = \sum_{X \in T_c \wedge T_c \subseteq D} X.nu(T_c)$; $SUM(X.rpu) = \sum_{X \in T_c \wedge T_c \subseteq D} X.rpu(T_c)$; $SUM(X.iu) = SUM(X.pu) + SUM(X.nu)$.

Lemma 1. Based on the PU$^{\pm}$-list, given two itemsets X and Y in a subtree in the PU$^{\pm}$-tree, if (1) $SUM(X.pu) + SUM(X.rpu)$ - $\sum_{\forall T_c \in D, X \subseteq T_c \bigwedge Y \nsubseteq T_c}(X.pu + X.rpu) < minUtil$, or (2) $SUM(X.pro)$ - $\sum_{\forall T_c \in D, X \subseteq T_c \bigwedge Y \nsubseteq T_c}(X.pro) < minPro \times |D|$, then neither $\{X, Y\}$ nor any of X it extensions will be a PHUI.

Strategy 1 (PU-Prune strategy). *Let X be a node of the PU$^{\pm}$-tree, and Y be the right sibling node of X. If $SUM(X.pu) + SUM(X.rpu)$ - $\sum_{\forall T_c \in D, X \subseteq T_c \bigwedge Y \nsubseteq T_c}(X.pu + X.rpu) < minUtil$, or $SUM(X.pro)$ - $\sum_{\forall T_c \in D, X \subseteq T_c \bigwedge Y \nsubseteq T_c}(X.pro) < minPro \times |D|$, then $\{X, Y\}$ and any of X its child nodes is not a PHUI. The construction of the PU$^{\pm}$-lists of X its children is unnecessary to be performed.*

Based on the PU-Prune strategy, a huge number of unpromising k-itemset ($k \geq 2$) can be pruned. The PU$^{\pm}$-list construct procedure with PU-Prune strategy is given in Algorithm 1. Thus, PU$^{\pm}$-list for k-itemsets ($k > 1$) can be easily constructed from PU$^{\pm}$-lists of (k-1)-itemsets without scanning the database.

```
Input: P: a pattern, Px: the extension of P with an item x, Py: the extension
       of P with an item y
output: The PU±-list of Pxy
Pxy.PUL ← ∅;
set Probability = SUM(X.pro), Utility = SUM(X.pu) + SUM(X.rpu);
foreach tuple ex ∈ Px.PUL do
    if ∃ey ∈ Py.PUL and ex.tid = exy.tid then
        if P.PUL ≠ ∅ then
            Search element e ∈ P.PUL such that e.tid = ex.tid.;
            exy ←< ex.tid, ex.pro × ey.pro/e.pro, ex.pu + ey.pu − e.pu, ex.nu +
              ey.nu − e.nu, ey.rpu >;
        else
            exy ←< ex.tid, ex.pro×ey.pro, ex.pu+ey.pu, ex.nu+ey.nu, ey.rpu >;
        Pxy.PUL ← Pxy.PUL ∪ {exy};
    else
        Probability = Probability − ex.pro, Utility = Utility − ex.pu − ex.rpu;
        if Probability < minPro × |D|||Utility < minUtil then
            return null;
return Pxy.PUL
```

Algorithm 1. The PU$^{\pm}$-list construct procedure with PU-Prune

3.3 Proposed Pruning Strategies

Based on the PU$^{\pm}$-list and the properties of probability and utility, several pruning strategies are designed in HUPNU to early prune unpromising itemsets. Assume a (k-1)-itemset w.r.t. a node in the Set-enumeration PU$^{\pm}$-tree be $X^{k-1} (k \geq 2)$, and any of its child nodes be denoted as X^k.

Theorem 1 (downward closure property of *RTWU* and *probability*). *In the $PU^{\pm}$-tree, the $Pro(X^{k-1}) \geq Pro(X^k)$ and $RTWU(X^{k-1}) \geq RTWU(X^k)$.*

Proof. Since $p(X, T_c) = \prod_{i \in X} p(i, T_c)$ for any T_c in D, it can be found that: $p(X^k, T_c) \leq p(X^{k-1}, T_c)$. X^{k-1} is subset of X^k, the *tids* of X^k is the subset of the *tids* of X^{k-1}, thus, $Pro(X^k) = \sum_{X^k \subseteq T_c \wedge T_c \in D} p(X^k, T_c) \leq \sum_{X^{k-1} \subseteq T_c \wedge T_c \in D} p(X^{k-1}, T_c) = Pro(X^{k-1})$. It can be found that $Pro(X^{k-1}) \geq Pro(X^k)$. Besides, $X^{k-1} \subseteq X^k$, $RTWU(X^k) = \sum_{X^k \subseteq T_c \wedge T_c \in D} tu(T_c) \leq \sum_{X^{k-1} \subseteq T_c \wedge T_c \in D} tu(T_c) = RTWU(X^{k-1})$.

Lemma 2 (probability upper-bound of PHUI). *The sum of all the probabilities of any node in the $PU^{\pm}$-tree is no less than the sum of the probabilities of any of its child nodes.*

Strategy 2. *After the first database scan, we can obtain the RTWU and probability value of each 1-item. If the RTWU of a 1-item and the sum of the probabilities of an item do not satisfy the two conditions of PHUI, this item can be directly pruned, and none of its supersets is a desired PHUI.*

Strategy 3. *When traversing the $PU^{\pm}$-tree based on a depth-first search strategy, if the sum of all the probabilities of a tree node X w.r.t. $Pro(X)$ in its constructed $PU^{\pm}$-list is less than $minPro \times |D|$, then none of the child nodes of this node is a desired PHUI.*

Lemma 3 (utility upper-bound of PHUI). *For any node X in the search space w.r.t. the $PU^{\pm}$-tree, the sum of $SUM(X.pu)$ and $SUM(X.rpu)$ in the $PU^{\pm}$-list of X is larger than or equal to utility of any one of its children.*

Thus, the sum of utilities of X^k in D w.r.t $u(X^k)$ is always less than or equals to the sum of $SUM(X^{k-1}.pu)$ and $SUM(X^{k-1}.rpu)$, it ensures that the downward closure of transitive extensions with positive or negative items. Based on these upper-bounds, we can use the following pruning conditions.

Strategy 4. *When traversing the $PU^{\pm}$-tree based on a depth-first search strategy, if the sum of $SUM(X^{k-1}.pu)$ and $SUM(X^{k-1}.rpu)$ of any node X is less than minUtil, any of its child node is not a PHUI, they can be regarded as irrelevant and be pruned directly.*

Strategy 5. *After constructing the $PU^{\pm}$-list of an itemset, if $X.PUL$ is empty or the $Pro(X)$ value is less than $minPro \times |D|$, X is not a PHUI, and none of X its child nodes is a PHUI. The construction of the $PU^{\pm}$-lists for the child nodes of X is unnecessary to be performed.*

We further extend the Estimated Utility Co-occurrence Pruning (EUCP) strategy [8] in the HUPNU algorithm, a structure named Estimated Utility Co-occurrence Structure (*EUCS*) is built. *EUCS* is a matrix that stores the *RTWU* values of the 2-itemsets, details can be referred to [8].

Strategy 6. *Let X be an itemset (node) encountered during the depth-first search of the Set-enumeration $PU^{\pm}$-tree. If the RTWU of a 2-itemset $Y \subseteq X$ according to the constructed EUCS is less than the minimum utility threshold, X would not be a PHUI; none of its child nodes is a PHUI. The construction of the $PU^{\pm}$-lists of X and its children is unnecessary to be performed.*

3.4 Main Procedure of HUPNU

As shown in Algorithm 2, the main procedure of the proposed HUPNU algorithm first scans the uncertain database to calculate the $RTWU$ (with the redefined RTU) and $Pro(i)$ of each item (Line 1). Then, it finds the set I^* of all items that not only having a existence probability no less than $minPro \times |D|$, but also having a $RTWU$ no less than $minUtil$, other items are ignored since they cannot be part of a potential HUI (Line 2). A second database scan is then performed (Line 4) after sorting the set of I^* in the designed order as $\succ$ (Line 3). During this database scan, items in transactions are reordered according to the total order $\succ$, the PU$^{\pm}$-list of each 1-item $i \in I^*$ is built and the structure named EUCS is built simultaneously. After that, the depth-first search exploration starts by calling the recursive procedure *Search* with the empty itemset $\emptyset$, the set of single items I^*, $minPro$, $minUtil$ and the EUCS (Line 5).

Input: D: an uncertain transaction database; $minPro$, a minimum potential probability threshold; $minUtil$: a minimum utility threshold; *ptable*: a profit-table
output: The set of potential high-utility itemsets ($PHUIs$)

1 Scan D to calculate the $RTWU$ and $Pro(i)$ of single item;
2 $I^* \leftarrow$ each item i such that $Pro(i) \geq minPro \times |D| \wedge RTWU(i) \geq minUtil$;
3 Sort the set of I^* in the designed order as $\succ$;
4 Scan D again to built the PU$^{\pm}$-list for each item $i \in I^*$ and built the $EUCS$ structure;
5 call `Search` **(**$\emptyset$**,** I^***,** $minPro$**,** $minUtil$**,** $EUCS$**)**;
6 **return** $PHUIs$

Algorithm 2. The HUPNU algorithm

As shown in Algorithm 3, the search procedure operates as follows. For each extension Px of P, if the probability of Px is no less than $minPro \times |D|$, and the sum of the actual utilities values of Px in the PU$^{\pm}$-list (denoted as $SUM(X.pu) + SUM(X.nu)$) is no less than $minUtil$, then Px is a PHUI and be output (Lines 2 to 3). Then, it uses the pruning strategies 3 and 4 to determine whether the extensions of Px would be the PHUIs and should be explored (Line 4). This is performed by merging Px with all extensions Py of P such that $y \succ x$ and $RTWU(\{x, y\}) \geq minUtil$ (Line 7, pruning strategy 6), to form extensions of the form Pxy containing $|Px|+1$ items. The PU$^{\pm}$-list of Pxy is then constructed by calling the *Construct* procedure to join the PU$^{\pm}$-lists of P, Px and Py (Lines 8 to 11). Only the promising PU$^{\pm}$-lists would be explored in next extension (Line 11, pruning strategy 5). Then, a recursive call to the *Search* procedure with Pxy is done to calculate its utility and explore its extension(s) (Line 12).

Input: P: an itemset, *ExtensionsOfP*: a set of extensions of P, the $minPro$ threshold, the $minUtil$ threshold, the $EUCS$ structure
Output: The set of potential high-utility itemsets (*PHUIs*)

1 **foreach** *itemset* $Px \in ExtensionsOfP$ **do**
2   **if** $SUM(Px.pro) \geq minPro \times |D| \wedge SUM(Px.pu) + SUM(Px.nu) \geq minUtil$ **then**
3   output Px as a $PHUI$;
4   **if** $SUM(Px.pro) \geq minPro \times |D| \wedge SUM(Px.pu) + SUM(Px.rpu) \geq minUtil$ **then**
5         $ExtensionsOfPx \leftarrow \emptyset$;
6   **foreach** *itemset* $Py \in ExtensionsOfP$ *such that* $y \succ x$ **do**
7   **if** $RTWU(\{x, y\}) \geq minUtil$ **then**
8                 $Pxy \leftarrow Px \cup Py$;
9                 $Pxy.PUL \leftarrow$ `Construct` (P, Px, Py);
10   **if** $Pxy.PUL \neq \emptyset \wedge SUM(Pxy.pro) \geq minPro \times |D|$ **then**
11                     $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
12   call `Search` (Px, *ExtensionsOfPx*, $minPro$, $minUtil$);
13 **return** *PHUIs*

Algorithm 3. The *Search* procedure

4 Experimental Study

In this section, we evaluated the performance of the proposed HUPNU algorithm. Experiments were implemented in Java and performed on a computer with a third generation 64 bit Core i5 processor running Windows 7 operating system and 4 GB of free RAM. In the literature, note that there is none study which is related to the task of mining HUIs from uncertain database with both positive and negative profits. We compared the performance of HUPNU with the proposed several pruning strategies. Note that the HUPNU$_{P1}$ adopts the pruning strategies 2, 3 an 4, the HUPNU$_{P2}$ adopts the pruning strategies 1, 2, 3 an 4, the HUPNU$_{P123}$ adopts pruning strategies 1, 2, 3, 4 and 5, while the HUPNU$_{P1234}$ adopts all pruning strategies including EUCP strategy.

All memory measurements were done using the Java API. Experiments were carried on four real-life datasets, kosarak, accidents, psumb and mushroom which having varied characteristics. The $\#transactions$, $\#distinctitems$, $avg.length$ and $max.length$ of these four datasets are respectively as: 990002, 41270, 8.09, 2498; 340183, 468, 33.8, 51; 49046, 2113, 74, 74; 8124, 119, 23, 23. For all datasets, external utilities for items are generated between -1,000 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, similarly to the settings of [8,12,14]. In addition, due to the attribute uncertainty property, a unique probability value in the range of (0.0, 1.0] was assigned to each item in every transaction in these datasets.

4.1 Runtime Performance

The comparison of execution times with various $minUtil$ threshold and various $minPro$ are shown in Fig. 1 for all datasets. From Fig. 1, it can be observed that the runtime of all the algorithms is decreased along with the increasing of $minUtil$ with a fixed $minPro$, or with the increasing of $minPro$ with a fixed $minUtil$. In particular, the proposed improved algorithms are generally up to almost one or two orders of magnitude faster than the baseline one on all datasets. Among the four version algorithms, HUPNU$_{P1234}$ which adopts all pruning strategies has the best performance. It is reasonable since HUPNU$_{P1234}$ uses six pruning strategies to early prune unpromising itemsets and search space, which can avoid the costly join operations of a huge number of PU$^{\pm}$-lists for mining PHUIs. When the $minUtil$ or $minPro$ is set quite low, longer desired patterns are discovered, and thus more computations w.r.t. runtime are needed to process, especially in a dense dataset. Based on the PU$^{\pm}$-list, the four HUPNU algorithms directly determine the PHUIs from the Set-enumeration tree without candidate generation, it can effectively avoid the time-consuming dataset scan. Moreover, the six pruning strategies help to prune a huge of unpromising

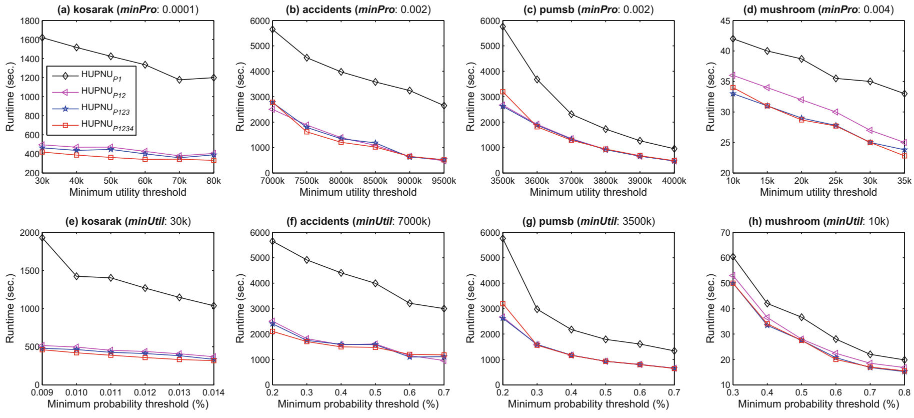

Fig. 1. Comparisons of runtime.

itemsets and to greatly reduce the computations than the baseline one. Moreover, the less memory usage is required, but we omit the detailed memory usage results due to space constraint. We can see this trend more clearly when the *minUtil* or *minPro* is set quite low. Thus, they can lead to a more compact search space and obtain the effectiveness and efficiency for mining PHUIs.

4.2 Scalability Analysis with Memory Usage and Patterns

As shown in Fig. 2, the scalability of the four algorithms is compared in the real-life dataset BMS-POS with different scales, which is set $minPro = 0.0001$, $minUtil = 10$k, and data size is set varying from 100k to 500k. It can be observed that the runtime of all compared algorithms is linear increased along with the increasing of dataset size. The runtime of HUPNU$_{P123}$ is close to that of HUPNU$_{P12}$, but significantly faster than that of HUPNU$_{P1}$. Specially, HUPNU$_{P1234}$ performs the best, and the gap of runtime among them grows wider with the increasing of dataset size. With the increasing of dataset size, the runtime of algorithms are linearly increasing as well. Figure 2(b) shows the memory usages of four algorithms which indicates the linearity in term of dataset size. In addition, HUPNU$_{P1}$ requires the most memory usage, HUPNU$_{P123}$ and HUPNU$_{P1234}$ have the similar performance on memory usage, they consume the least memory. To show the effect of the developed pruning strategies, the number of potential nodes (visited nodes in the PU$^{\pm}$-tree, denoted as N_1, N_2, N_3, and N_4) and the final derived PHUIs are further evaluated as shown in Fig. 2(c). It can be observed that $N_1 > N_2 > N_3 > N_4$, the larger dataset size is, the bigger gap among them is.

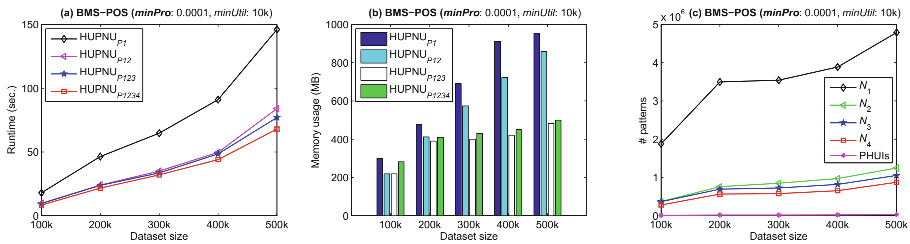

Fig. 2. Scalability test.

5 Conclusion

In this paper, we proposed an algorithm named HUPNU (mining High-utility itemsets with both Positive and Negative unit profits from Uncertain databases), it is the first work to address this problem. A novel vertical list structure, called PU$^{\pm}$-list (probability-utility list with positive-and-negative profits), is designed for HUPNU to mine potential high-utility itemsets (PHUIs) without generating candidates. Several efficient pruning strategies are further developed to reduce the search space and speed up computation. Experiments carried on several real-life datasets shows that the complete set of PHUIs can be efficiently discovered by the proposed HUPNU algorithm. HUPNU is quite efficient in terms of runtime and scalability, and the designed pruning strategies are acceptable.

Acknowledgement. This research was partially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61503092 and by the Tencent Project under grant CCF-Tencent IAGR20160115.

References

1. Frequent Itemset Mining Dataset Repository. http://fimi.ua.ac.be/data/
2. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. IEEE Trans. Knowl. Data Eng. **21**(5), 609–623 (2009)
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the International Conference on Very Large Databases, pp. 487–499 (1994)
4. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Lee, Y.K.: Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans. Knowl. Data Eng. **21**(12), 1708–1721 (2009)
5. Bernecker, T., Kriegel, H.P., Renz, M., Verhein, F., Zuefl, A.: Probabilistic frequent itemset mining in uncertain databases. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 119–128 (2009)
6. Chan, R., Yang, Q., Shen, Y.: Mining high utility itemsets. In: IEEE International Conference on Data Mining, pp. 19–26 (2003)
7. Chu, C.J., Tseng, V.S., Liang, T.: An efficient algorithm for mining high utility itemsets with negative item values in large databases. Appl. Math. Comput. **215**, 767–778 (2009)
8. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS (LNAI), vol. 8502, pp. 83–92. Springer, Cham (2014). doi:10.1007/978-3-319-08326-1_9
9. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: 21st ACM International Conference on Information and Knowledge Management, pp. 55–64 (2012)
10. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). doi:10.1007/11430919_79
11. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P., Tseng, V.S.: Mining potential high-utility itemsets over uncertain databases. In: ACM ASE BigData & Social Informatics, p. 25 (2015)
12. Fournier-Viger, P.: FHN: efficient mining of high-utility itemsets with negative unit profits. In: Luo, X., Yu, J.X., Li, Z. (eds.) ADMA 2014. LNCS (LNAI), vol. 8933, pp. 16–29. Springer, Cham (2014). doi:10.1007/978-3-319-14717-8_2
13. Rymon, R.: Search through systematic set enumeration. Technical reports (CIS), pp. 539–550 (1992)

14. Tseng, V.S., Shie, B.E., Wu, C.W., Yu, P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. **25**(8), 1772–1786 (2013)
15. Yao, H., Hamilton, H.J., d Butz C.J.: A foundational approach to mining itemset utilities from databases. In: SIAM International Conference on Data Mining, pp. 211–225 (2004)