Advanced Engineering Informatics 30 (2016) 109–126

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Fast algorithms for mining high-utility itemsets with various discount strategies

Jerry Chun-Wei Lin [a,*], Wensheng Gan [a], Philippe Fournier-Viger [b], Tzung-Pei Hong [c,d], Vincent S. Tseng [e]

[a] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, 518055, China
[b] School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, 518055, China
[c] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, 811, Taiwan
[d] Department of Computer Science and Engineering, National Sun Yat-sen University, 804, Kaohsiung, Taiwan
[e] Department of Computer Science, National Chiao Tung University, Hsinchu, 300, Taiwan

ARTICLE INFO

Article history:
Received 19 May 2015
Received in revised form 8 February 2016
Accepted 11 February 2016

Keywords:
High-utility itemsets
Discount strategies
Downward closure property
Pruning strategies
PNU-list

ABSTRACT

In recent years, mining high-utility itemsets (HUIs) has emerged as a key topic in data mining. It consists of discovering sets of items generating a high profit in a transactional database by considering both purchase quantities and unit profits of items. Many algorithms have been proposed for this task. However, most of them assume the unrealistic assumption that unit profits of items remain unchanged over time. But in real-life, the profit of an item or itemset varies as a function of cost prices, sales prices and sale strategies. Recently, a three-phase algorithm has been proposed to mine HUIs, while considering that each item may have different discount strategies. However, the complete set of HUIs cannot be retrieved based on the traditional TWU model with its defined discount strategies. Moreover, it suffers from the well-known drawbacks of Apriori-based algorithms such as maintaining a huge amount of candidates in memory and repeatedly performing time-consuming database scans. In this paper, a HUI-DTP algorithm for mining HUIs when considering discount strategies of items is introduced. The HUI-DTP is designed as a two-phase algorithm to mine the complete set of HUIs based on a novel downward closure property and a vertical TID-list structure. Furthermore, the HUI-DMiner is an algorithm relying on a compact data structure (Positive-and-Negative Utility-list, PNU-list) and properties of two new pruning strategies to efficiently discover HUIs without candidate generation, while considerably reducing the size of the search space. Moreover, a strategy named Estimated Utility Co-occurrence Strategy which stores the relationships between 2-itemsets is also applied in the improved HUI-DEMiner algorithm to speed up computation. An extensive experimental study carried on several real-life datasets shows that the proposed algorithms outperform the previous best algorithm in terms of runtime, memory consumption and scalability.

1. Introduction

Frequent itemset mining (FIM) or association rule mining (ARM) [2,3,8,14] is a fundamental data mining task, which consists of discovering relationships between items in a transactional database. Many algorithms have also been proposed to efficiently mine frequent itemsets and association rules, usually adopting a level-wise approach [3] or pattern-growth approach [14]. An unrealistic assumption of FIM or ARM is that it only considers binary quantities of items in transactions (a customer may buy one or zero unit of an item). But in real-life, a customer may buy several units of the same item. Furthermore, FIM or ARM does not consider other important factors such as the unit profit of items (all items are considered as having the same unit profit). As a result, many patterns found may be frequent but may generate a very low profit, and thus may be uninteresting for the user. For example, diamond sales may be much less frequent than clothing sales in a department store, but the former has a much larger profit margin than the latter, and may thus be more relevant. Finding associations between items solely based on their occurrence frequency as in traditional FIM or ARM is not suitable to identify these highly profitable itemsets. Factors such as price, quantity and cost should also be considered to analyze and predict customer purchase behavior.

To address some of these limitations, high-utility itemset mining (HUIM) [7,27] was proposed. Under HUIM framework, the

* Corresponding author.
E-mail addresses: jerrylin@ieee.org (J. Chun-Wei Lin), wsgan001@gmail.com (W. Gan), philfv@hitsz.edu.cn (P. Fournier-Viger), tphong@nuk.edu.tw (T.-P. Hong), vtseng@cs.nctu.edu.tw (V.S. Tseng).

http://dx.doi.org/10.1016/j.aei.2016.02.003
1474-0346/© 2016 Elsevier Ltd. All rights reserved.

"utility" of an itemset can be measured in terms of quantity and profit according to users' preferences. For example, a user may be interested in finding itemsets yielding a high profit, while another user may focus on discovering itemsets causing low pollution during the manufacturing process. When the utility of an itemset is no less than a minimum utility threshold, it is considered as a high-utility itemset (HUI); otherwise, it is a low-utility itemset. The discovered information of HUIs can be generally used in various applications, such as decision support systems [5,7,27], as well as a framework of data mining based analysis [9], to aid managers or retailers for take efficient decisions or choose the most profitable business strategies for their companies.

In real-life retail stores, the profit earned from the sale of items depends on the cost prices, tag prices and discount strategies. Different stores may use different discount strategies to sell the same products. Furthermore, the use of discount strategies may vary based on time periods. Traditional way of measuring the utility of itemsets in the presence of different discount strategies used for each item is to consider not only the positive profit generated by the sale of items but also the negative profit. For example, a popular discount strategy is to sell some items at high discount or even to give them away, such as mouse and keyboard are always sold at a high discount or even free with laptop. This may yield negative profit for the vendor. In such scenario, however, these products are often cross-promoted with others having a positive profit margin, thus leading to an overall positive profit. A cross promotion is a common phenomenon in marketing management that targets buyers of a product with an offer to purchase the related products, and the appropriate discount strategies may affect users' shopping behavior.

Discovering HUIs with the constraint that each item may be associated with its own discount strategy is highly desirable since discounting is done in most real-life stores. However, most algorithms for HUIM are not designed to handle items with profit values that can vary under different discount strategies, which thus limits their usefulness in real-life. Mining HUIs while considering the discount strategy associated to each item is more computationally expensive than traditional HUIM. Furthermore, another challenge is that the pruning strategies or properties of traditional HUIM cannot be directly adopted to solve this problem. Chu et al. developed the HUINIV-Mine algorithm to discover HUIs while considering items with negative profits (HUINIV) [10]. It is a two-phase algorithm, that is, it overestimates the utility of itemsets to prune the search space, and then scans the database again to calculate the exact utility of itemsets. It suffers from well-known drawbacks, which lead to high execution time and memory consumption. Fournier-Viger adopted the utility-list structure and EUCP strategy in the FHN algorithm for mining HUIs with negative profit more efficiently [17]. Li et al. proposed a three-phase algorithm to discover HUIs under four discount strategies [19]. Because this three-phase algorithm also overestimates the utilities of itemsets as in the Two-Phase model [24], it also suffers from the same drawbacks. Moreover, this algorithm is not designed to handle negative unit profit. If negative unit profit is introduced, the algorithm fails to mine the complete set of HUIs. It is thus a challenge to design an efficient algorithm to mine the whole set of HUIs when both positive and negative items are used, and each item has its own discount strategy.

In this paper, we address this issue by proposing algorithms to mine HUIs when each item has its own discount strategy. The proposed algorithms are quite different from the previous HUINIV-Mine algorithm. The reasons are shown as follows: (1) the approach for calculating profit of all items in the transaction database is not the same as previous works; (2) both positive (includes zero) and negative items profits are taken into account; (3) the used data structure and proposed approaches are different from HUINIV-Mine; (4) since discount strategy affects users' shopping behavior by evaluating the maximal total profit which bring from derived HUIs and those common HUIs, the managers or retailers can find the optimal discount strategies and decisions to get the total revenue maximization. Hence, HUIM with discount strategies is a more general problem than previous ones. The proposed algorithms can be used as efficient tools for various applications, including decision support systems (DSS) [5,7,27], for managers or retailers to discover more useful and meaningful information in many real-life applications. Moreover, based on the various definitions of "utility" (profit, benefit, weight, risk, etc.), the proposed technologies can be applied to other various domains, such as multi-dimensional data analysis, benefits evaluation, and possibilities and risk assessment in engineering informatics [6]. The contributions of this paper are as follows.

1. Several algorithms of high-utility itemset mining with various discount strategies, are designed to reveal more useful and meaningful HUIs when considering items having various discount strategies. The proposed algorithms are more adapted to real-life situations than traditional HUIM.
2. To the best of our knowledge, this is the first paper successfully solves the addressed mining problem. Two algorithms are developed to mine HUIs when considering items having various discount strategies without losing any HUIs.
3. The HUI-DTP algorithm is proposed as a baseline algorithm. It relies on a level-wise search to mine HUIs and integrates a new downward closure property to prune unpromising candidates without losing any HUIs.
4. The HUI-DMiner algorithm is proposed as an efficient algorithm. It relies on a vertical structure, called Positive-and-Negative Utility-list (PNU-list), to mine HUIs without generating and maintaining candidates in memory. Two efficient pruning strategies are further proposed to reduce the search space, and thus speed up the discovery of HUIs. Furthermore, a structure called the Estimated Utility Co-occurrence Structure (EUCS) is also used to prune the search space.
5. Based on the designed algorithms, the complete set of HUIs can be efficiently discovered. An extensive experimental study carried on several real-life datasets shows that the proposed algorithms largely outperform the previous best algorithm in terms of runtime, memory consumption and scalability.

The remaining of this paper is organized as follows. Related work in high-utility itemset mining and comparative analysis are reviewed in Section 2. The proposed work of high-utility itemset mining with various discount strategies is described in Section 3. The proposed HUI-DTP and HUI-DMiner algorithms are respectively presented in Sections 4 and 5. Experiments are conducted in Section 6. Finally, Section 7 draws conclusions and provides a discussion.

2. Related work

This section briefly reviews related studies on high-utility itemset mining and presents the difference and relationship between this paper and previous works.

2.1. High-utility itemset mining

Traditional ARM has several important limitations. First, it can only discover relationships between items in a binary database, i.e. where items may appear not more than once in each transaction. Second, itemsets are only found based on frequency, and only this information is provided to the user, which is often insufficient in real-life situations. High-utility itemset mining (HUIM) was proposed as an extension of frequent itemset mining that addresses some of these limitations. It considers both the quantities and profits of items, and outputs itemsets that are valuable and profitable itemsets rather than only the frequent ones. An itemset is said to be a high-utility itemset (HUI) if its utility is no less than a given minimum utility threshold. Chan et al. first proposed top-*k* objective-directed data mining to mine top-*k* closed utility patterns based on business objectives [7,27]. The proposed approach can discover not only the frequent itemsets but also those that are HUIs. Yao et al. proposed the fundamental utility model that considers both quantities and profits of items for mining HUIs [28], as well as several mathematical properties of utility constraints and two pruning strategies to efficiently mine HUIs. The Two-Phase model [24] was proposed to mine HUIs by using a pruning property named the Transaction-Weighted Downward Closure (TWDC) property. The first phase of the Two-Phase model consists of first overestimating the utility of itemsets, to be able to prune the search space and obtain a set of candidates. Then, the second phase consists of scanning the database to calculate their exact utility and keep only those that are HUIs. Although this approach improves upon previous work, an important drawback is that numerous candidates need to be explored when the utility is too much overestimated.

Many algorithms have been proposed to mine HUIs using a pattern-growth approach. Ahmed et al. proposed the IHUP algorithm, which uses a tree structure for mining HUIs interactively in incremental databases [5]. Thanks to its tree structure, IHUP avoids the well-known limitations of the level-wise approach, such as having to generate and maintain many patterns into memory. Lin et al. designed a High Utility Pattern (HUP)-tree algorithm to compress the original database into a tree structure, and the HUP-growth mining algorithm was also designed to mine HUIs [20]. Tseng et al. proposed the UP-tree structure and corresponding UP-growth [25] mining algorithms to efficiently mine HUIs. These pattern-growth approaches, however, require time-consuming computations to trace tree nodes in the tree structure. Furthermore, these algorithms are also based on the TWU model and thus suffer from its drawbacks. To avoid the drawbacks of using a tree structure and of the TWU model, Liu et al. proposed the HUI-Miner algorithm and a novel data structure named utility-list, which compresses information from the database [23]. Based on the designed utility-list structure and a novel pruning strategy based on remaining utility, HUIs can be quickly discovered without generating candidates. Fournier-Viger et al. extended HUI-Miner by adding a structure called the Estimated Utility Co-occurrence Structure (EUCS), which keeps information about the utility of 2-itemsets to prune the search space [12]. Other algorithms for mining HUIs are still developed. Recently, many interesting issues on HUIM have been extensively studied, such as new algorithm for mining HUIs without candidate generation [22], HUIs mining in dynamic environment [15,21], on-shelf HUIs mining [13], up-to-date HUIs mining [17], HUIs mining with multiple minimum utility thresholds [16], top-*k* HUIs mining [26], HUIs mining in stream data [30].

Although traditional algorithms for HUIM consider the profits and quantities of items to find itemsets generating a high profit in transactional databases, the profit values of all items in databases are generally assumed to be positive. However, in real-world applications, the profit earned by the sale of an item may be negative. For example, a common strategy for promoting specific products in a supermarket is to package products with some free products to sell them together. It was demonstrated that if traditional HUIM algorithm are applied on database containing negative profit values, some HUIs may be missed. To address this issue, Chu et al. proposed a new two-phase model to mine HUIs with negative profit values (HUINIV) [10]. The HUINIV-Mine algorithm overestimates the utility of candidates as in the Liu et al. Two-Phase model. However, overestimation is only done using positive profit values to avoid missing HUIs. Lan et al. also proposed a novel algorithm [18] to consider both time periods and negative profit values. Fournier-Viger then developed the FHN algorithm to efficiently mine HUIs with negative unit profits, FHN was shown to outperform HUINIV-Mine [11].

Recently, Li et al. proposed a three-phase algorithm to discover HUIs when four discount strategies are considered [19]. The three-phase algorithm, however, fails to mine the complete set of HUIs when items with negative profit values appear in transactions. The reason is that the algorithm overestimates the utility of items to prune the search space. However, when negative profit values are introduced, this overestimation may become an underestimation and HUIs may be pruned (as in the Two-Phase model). Moreover, this algorithm suffers the drawbacks since it is implemented based on Two-Phase model. Thus, it may generate a very large amount of candidates and has to perform time-consuming database scans to calculate the exactly utility of each candidate to discover the promising HUIs.

2.2. Comparative analysis

According to the above descriptions, the addressed problem in this paper has the following main differences compared to the previous algorithms for mining high-utility itemsets. The differences are shown as follows.

(1) The mining environment (w.r.t. conditions) and goal are different. All the above approaches except the three-phase algorithm are designed to discover HUIs without the consideration of various discount strategies with sold goods. However, the discount strategies may affect users' shopping behavior. On the contrast, the designed algorithms aim at considering various discount strategies to mine interesting HUIs which having positive cross-promoted relationships among all goods.
(2) The data structure and mining techniques used in this paper are quite different from the above approaches. On one hand, the developed techniques in traditional HUIM algorithms cannot be directly applied to solve the addressed problem. As the mentioned above, the three-phase algorithm fails to mine the complete set of HUIs and suffers from the drawbacks based on the Two-Phase model. It is thus a challenge and an important research problem to design an appropriate data structure and more efficient technologies for mining HUIs with various discount strategies.
(3) Furthermore, this is the first work successfully solves the high-utility itemset mining problem with various discount strategies.

3. Preliminaries and problem statement

This section introduces preliminaries related to high-utility itemset mining (HUIM) and then defines the problem of mining high-utility itemsets while considering items with discount strategies. The notations used in this paper are shown as follows.

3.1. Notations

I	$I = \{i_1, i_2, \ldots, i_m\}$ is a set of m items
cp_j	The cost price value of an item i_j $(1 \leqslant j \leqslant m)$ is denoted as cp_j
tp_j	The tag price value of an item i_j $(1 \leqslant j \leqslant m)$ is denoted as tp_j
D	A quantitative database $D = \{T_1, T_2, \ldots, T_n\}$ is a set of n transactions, where $T_q \subseteq I$ $(1 \leqslant q \leqslant n)$, and where each item in a transaction is annotated with a quantity
TID	A transaction $T_q \in D$ has a unique identifier q referred as its TID (Transaction ID)
X	An itemset X is a set of k distinct items $\{i_1, i_2, \ldots, i_k\} \subseteq I$, where k is the length of the itemset. Note that for brevity, brackets may be omitted. An itemset X is contained in a transaction T_q if $X \subseteq T_q$
$u(i_j, T_q)$	The utility value of item i_j in transaction T_q
$tu(T_q)$	The sum of the utility values of items in transaction T_q
$q(i_j, T_q)$	The quantity of item i_j in transaction T_q
δ	The predefined minimum utility threshold
$TWU(X)$	The transaction-weighted utility of an itemset X in D

3.2. Preliminaries

Consider the transactional database shown in Table 1, which will be our running example. It contains 10 transactions and 6 items, represented by letters. Each item is associated with a purchased quantity in transactions where it occurs. For example, in the first transaction, three, two and four units of items A, C and E were respectively bought. We next introduce important definitions for the problem of mining HUIs with items having various discount strategies.

Definition 1 (*Price of a product*). The cost price and tag price of a product i_j are respectively denoted as $cp(i_j)$ and $tp(i_j)$. Depending on the discount strategies (denoted as Sg_x, in which x is the flag of Sg), the relationship between the cost price and tag price can be represented by any of the three following relationships: (1) if $tp(i_j) \times Sg_x \geqslant cp(i_j)$, it indicates that i_j is sold with a positive profit margin; (2) $0 < tp(i_j) \times Sg_x < cp(i_j)$, it indicates that i_j is sold with a negative profit margin; (3) $tp(i_j) = 0$, it indicates that i_j is offered as a free gift for the purpose of sales promotion.

Table 1
A transactional database.

TID	Transaction (item, quantity)
1	$(A, 3); (C, 2); (E, 4)$
2	$(D, 1); (F, 2)$
3	$(A, 1); (B, 3); (C, 1); (D, 3); (F, 1)$
4	$(B, 1); (D, 1); (F, 3)$
5	$(B, 1); (C, 4)$
6	$(A, 2); (B, 6); (C, 3); (D, 4); (F, 1)$
7	$(C, 1); (D, 2); (E, 5)$
8	$(B, 2); (E, 1)$
9	$(B, 2); (D, 1); (F, 1)$
10	$(A, 4); (B, 1); (D, 1); (E, 3)$

Definition 2 (*Price table*). A price table indicates the cost price and tag price of each product (w.r.t. item). In general, the tag price of a product is always higher than its cost price.

For example, consider the price table shown in Table 2. The cost price of item (A) is $cp(A)$ (= 25) and its tag price is $tp(A)$ (= 35).

Based on real-life situations described in [19], three types of discount strategies are formalized and stated as follows.

(1) **Strategy 1** (***Sg*₁**): An item is sold with a discount levels ranging from 0% to 100%. For example, if a customer buys a product (A) with a 25% discount, the utility (profit) of this item is calculated as 75% $\times$ $tp(A)$ – $cp(A)$.
(2) **Strategy 2** (***Sg*₂**): If N units of an item are bought by a customer, s/he will receive M free units of this item. Both N and M are positive integers. For example, if a customer buys product (C) when the discount strategy "buy 2 get 1 free" is in place, the utility (profit) of (C) is calculated as $2 \times tp(C) - 3 \times cp(C)$.
(3) **Strategy 3** (***Sg*₃**): If N units of an item are bought by a customer, s/he will receive m% discount on each additional unit purchased. Both N and m are positive integers. For example, if a customer buys five units of product (B) under the discount strategy "buy 3 and get a 35% discount on each additional unit", the utility of (B) is calculated as $(3 + 2 \times 65\%) \times tp(B) - 5 \times cp(B)$.

Definition 3 (*Discount strategy table*). A discount strategy table indicates the discount strategy in place for each item. Each line contains an item, its discount strategy (Sg_x), and two values v_1 and v_2 representing the parameters of the discount strategy.

For example, consider the discount strategy table shown in Table 3. It indicates that the discount strategy of item (A) is strategy 1, and more precisely that it is sold at a 25% discount.

Definition 4 (*Utility of an item in a transaction*). Let $q(i_j, T_q)$ denotes the quantity of an item i_j in a transaction T_q; $tp(i_j)$ and $cp(i_j)$ respectively be the tag price and cost price of item i_j according to the price table. The utility of an item i_j in a transaction T_q is denoted as $u(i_j, T_q)$, and defined with respect to discount strategies as follows.

(1) **Strategy 1:** The utility value of product i_j in transaction T_q is calculated as:

$$u(i_j, T_q) = q(i_j, T_q) \times [tp(i_j) \times v_1 - cp(i_j)].$$

(2) **Strategy 2:** The utility value of product i_j in transaction T_q is calculated as:

$$u(i_j, T_q) = \left[\left\lfloor \frac{q(i_j, T_q)}{v_1 + v_2} \right\rfloor \times v_1 + q(i_j, T_q)\%(v_1 + v_2)\right] \times tp(i_j) - q(i_j, T_q) \times cp(i_j),$$

Table 2
A price table.

Item	Cost price ($)	Tag price ($)
A	25	35
B	8	10
C	70	128
D	5	10
E	38	50
F	10	18

Table 3
Discount strategy table.

Item	Sg_i	v_1	v_2
A	1	0.75	–
B	1	1.0	–
C	2	2	1
D	1	0	–
E	3	2	0.6
F	1	0.8	–

where $q(i_j, T_q)$ must satisfy the equation $q(i_j, T_q) = n \times (v_1 + v_2) + k$, where n, k are integers, and $n \geqslant 0$, $0 \leqslant k < v_1$.

(3) **Strategy 3:** The utility value of product i_j in transaction T_q is calculated as:

$$u(i_j, T_q) = \begin{cases} [v_1 + (q(i_j, T_q) - v_1) \times v_2] & \\ \quad \times tp(i_j) - q(i_j, T_q) \times cp(i_j), & \text{if } q(i_j, T_q) > v_1, \\ q(i_j, T_q) \times [tp(i_j) - cp(i_j)], & \text{otherwise}. \end{cases}$$

For strategy 1, the utility of (B) in transaction T_3 is calculated as $u(B, T_3) = 3 \times (10 \times 1 - 8) = 6$, because no discount strategy is in place for item (B). The utility of (A) in transaction T_1 is calculated as $u(A, T_1) = 3 \times (35 \times 0.75 - 25) = 3.75$, which indicates that (A) is sold at a 25% discount. The utility of (D) in transaction T_3, is calculated as $u(D, T_3) = 3 \times (10 \times 0 - 5) = -15$ and indicates that (D) was offered as a free gift.

For strategy 2, assume that four units of (C) are purchased in transaction T_5. Thus, the utility of (C) in transaction T_6 under discount strategy 2 (buy 2 get 1 free) is calculated as $u(C, T_6) = \left[\left\lfloor \frac{6}{2+1} \right\rfloor \times 2 + 6\%(2+1)\right] \times 128 - 6 \times 70 = 92$.

For strategy 3, the utility of (E) in transaction T_1 is $u(E, T_1) = [2 + (4 - 2) \times 0.6] \times 50 - 4 \times 38 = 8$ since four units of (E) have been purchased in transaction T_1, which is larger than the parameter of discount strategy 3 (buy 2, get a 40% discount on additional units of the items $(4 > 2)$). The utility of (E) in transaction T_8 is calculated as $u(E, T_8) = 1 \times (50 - 38) = 12$ since one unit of (E) is purchased in transaction T_8, which is smaller than the parameter of discount strategy 3 $(1 < 2)$.

Definition 5 (*Utility of an itemset in a transaction*). The utility of an itemset $X = \{i_1, i_2, \ldots, i_j\}$ in a transaction T_q is denoted as $u(X, T_q)$ and is defined as:

$$u(X, T_q) = \sum_{i_j \in X \wedge X \subseteq T_q} u(i_j, T_q).$$

For example, the utility of (AC) in transaction T_1 is calculated as $u(AC, T_1) = u(A, T_1) + u(C, T_1) = 3.75 + 116 = 119.75$.

Definition 6 (*Utility of an itemset in a database*). The utility of an itemset X in a database D is denoted as $u(X)$, and defined as:

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in D} u(X, T_q).$$

For example, the utility of (A) in the running example is calculated as $u(A) = u(A, T_1) + u(A, T_3) + u(A, T_6) + u(A, T_{10}) = 3.75 + 1.25 + 12.5 + 5 = 22.5$. The utility of (AC) in the database is calculated as $u(AC) = u(AC, T_1) + u(AC, T_3) + u(AC, T_6) = 119.75 + 59.25 + 48.5 = 227.5$.

Using the above definitions, we can see that the utility of an itemset in a database will vary depending on the discount strategies used, which is quite different from how utility was defined in traditional HUIM and HUIM with negative profit values. The proposed algorithms considers how the profit is influenced by the discount strategies, which is more realistic and thus more useful for real-life applications.

Definition 7 (*High-utility itemset, HUI*). Let δ be a minimum utility threshold set by the user (a positive integer). Note that this user-specified threshold is based on users' background knowledge and goals. It is somewhat similar to how to set the minimum support and confident thresholds in association rule mining task. An itemset X in a database D is called a high-utility itemset (HUI) if $u(X) \geqslant \delta$. Otherwise, it is said to be a low-utility itemset. The set of all high-utility itemsets in a database is thus denoted as HUI and defined as $HUI \leftarrow \{X | u(X) \geqslant \delta\}$.

For example, assume that the minimum utility threshold δ is set to 100. An item (A) is considered low-utility in D since $u(A) = 22.5 < \delta(= 100)$. The itemset (AC) is a HUI in D since $u(AC) = 227.5 > \delta$. Table 4 shows the complete set of HUIs for the database of the running example under the discount strategies of Table 3. The problem statement of HUIM with items having various discount strategies is the following.

3.3. Problem statement

Let D be a transactional quantitative database, a price table indicating the cost price and tag price of each item, a discount strategy table, and a user-defined minimum utility threshold δ. The problem of mining HUIs from database D with items having various discount strategies is to discover all high-utility itemsets in D.

Due to discount strategies affect users' shopping behavior, the designed algorithms are helpful to analyze the effect of cross-promotion by the derived various HUIs under different discount strategies. With evaluating the maximal total profit which bring from derived HUIs and those common HUIs, the managers or retailers can find the optimal discount strategies and decisions for goods to get the total revenue maximization. For example, consider the case when (A, B, C, D) is a cross-promotion pattern, (A), (B), (C) have positive value and (D) has negative value. This is a case that an itemset (ABC) is a HUI as a result of cross-promotion with (D), if the itemset $(ABCD)$ is still a HUI under the cross-promoted discount strategies, we can conclude that (ABC) and (D) have a strong positive cross-promoted relationship; otherwise, if the itemset $(ABCD)$ is not a HUI, it indicates that there is a negative cross-promoted relationship between (ABC) and (D), it would not helpful to perform the cross-promotion between (ABC) and (D). Hence, the designed algorithms are helpful to find the optimal discount strategies and decisions for the managers or retailers.

4. The HUI-DTP algorithm

In this section, a two-phase naive algorithm named HUI-DTP is presented to mine HUIs while considering items having various

Table 4
Final derived HUIs for the running example.

Itemset	Utility
(C)	382.0
(AC)	227.5
(BC)	228.0
(CD)	117.0
(CE)	182.0
(CF)	112.8
(ABC)	125.75
(ACE)	127.75
(ACF)	116.55
(BCF)	130.8
$(ABCF)$	134.55

discount strategies. To prune the search space and avoid the combinatorial explosion, HUI-DTP relies on a modified downward closure property.

4.1. Proposed downward closure property

Frequent itemset mining algorithms rely on a property called downward closure (DC) introduced in the Apriori algorithm to prune candidate patterns, and thus reduce the time to mine association rules. This property states that for any itemset X and any proper superset Y of X, the support (i.e. frequency) of Y is less than or equal to the frequency of X (i.e., the support is anti-monotonic). This property is very powerful for pruning the search space. However, it is not possible to directly rely on such a property to prune the search space in traditional HUIM since utility is not anti-monotonic. For example, as shown in Table 4, the itemset (E) is not a HUI, but its supersets (CE) and (ACE) are HUIs. Because the DC property cannot be directly used in HUIM, the transaction-weighted downward closure (TWDC) property was introduced as part of the Two-Phase model [24].

Definition 8 (*Utility of a transaction, tu*). The transaction utility of a transaction $T_q = \{i_1, i_2 \ldots i_m\}$ is denoted as $tu(T_q)$, and is defined as:

$$tu(T_q) = \sum_{j=1}^{m} u(i_j, T_q).$$

in which m is the number of distinct items in T_q. For example, the transaction utility of T_1 is calculated as $tu(T_1) = u(A, T_1) + u(C, T_1) + u(E, T_1)(= 3.75 + 116 + 8)\ (= 127.75)$.

Definition 9 (*Total utility of a database*). The total utility of a database D is denoted as TU, which can be defined as:

$$TU = \sum_{T_q \in D} tu(T_q).$$

For example, the total utility of D is the sum of all transaction utilities in D, which can be calculated as: $TU = (127.75 + 3.8 + 54.65 + 10.2 + 106 + 44.9 + 48 + 16 + 3.4 + 18)\ (= 429)$.

Definition 10 (*Transaction-weighted utility, twu*). The transaction-weighted utility (TWU) of an itemset X is the sum of all transaction utilities $tu(T_q)$ containing X, which is defined as:

$$twu(X) = \sum_{X \subseteq T_q \wedge T_q \in D} tu(T_q).$$

Definition 11 (*High transaction-weighted utilization itemset, HTWUI*). An itemset X is called a high transaction-weighted utilization itemset (HTWUI) if its *twu* value is no less than the minimum utility threshold as:

$$HTWUI \leftarrow \{X | twu(X) \geqslant \delta\}.$$

Property 1 (*Anti-monotonicity*). *The transaction-weighted utility measure is anti-monotonic [24], that is, if X and Y are two itemsets and $X \subseteq Y$, then $twu(X) \geqslant twu(Y)$.*

Property 2 (*Pruning strategy with downward closure property*). *The transaction-weighted downward closure (TWDC) property states that if an itemset X is not a HTWUI, any subsets of X are not HTWUI.*

Because the *twu* value is always an upper-bound on the utility of itemsets in traditional HUIM, the TWDC property can be used to prune the search space [24]. The three-phase algorithm proposed in [19] to mine HUIs while considering items having various discount strategies is based on the TWU model. However, it fails to discover the complete set of HUIs when items with negative profit values appear in the database. This is an important limitation because items/itemsets with negative profit values are common in real-life when discount strategies are used. For instance, consider the discount strategy table shown in Table 3, and the database shown in Table 5 where the utility of each item and the *tu* of each transaction have been calculated. If the three-phase algorithm is run on this database, it will underestimate the utility of the itemsets containing negative profit values, because in that case, the TWU of an itemset is no longer an upper bound on its utility. Thus, some HUIs may not be discovered. For example, the transaction-weighted utility of (CF) is calculated as $twu(CF) = tu(T_3) + tu(T_6) = 54.65 + 44.9 = 99.55$, which is less than the minimum utility threshold $\delta(= 100)$. The itemset (CF) is thus regarded as a low transaction-weighted utilization itemset, meaning that (CF) is a low-utility itemset as well as all its supersets. However, this is incorrect since $u(CF) = u(CF, T_3) + u(CF, T_6) = 62.4 + 50.4 = 112.8 > 100$ and thus (CF) should be considered as a HUI, as well as its supersets (ACF), (BCF), and $(ABCF)$, as shown in Table 4. Thus, it is insufficient to adopt the Two-Phase model for mining HUIs while considering items having various discount strategies when an item/itemset may have a negative profit value. Based on the observation of Table 4, the following properties can be stated.

Observation 1. The traditional TWU model may not find the complete set of HUIs when the database contains items having negative utility values.

Proof. Let $X \in D$ be any itemset and the notation $(T_q \setminus X)$ denotes the set of all items in T_q that are not element of X. We have that $u(T_q \setminus X) = \sum_{i_j \in (T_q \setminus X)} u(i_j, T_q)$, where each item $i_j \in (T_q \setminus X)$ may have a positive or negative utility value. Thus, if $u(T_q \setminus X) > 0$, $u(X) \leqslant u(X) + u(T_q \setminus X) = tu(T_q)$. Moreover, if $u(T_q \setminus X) \leqslant 0$, $u(X) > u(X) + u(T_q \setminus X) = tu(T_q)$. If we assume that $u(X, T_q) > tu(T_q)$ holds for all transactions, then the following formula based on the traditional TWU model is insufficient to discover the complete set of HUIs when considering items have various discount strategies:

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in D} u(X, T_q) \leqslant \sum_{X \subseteq T_q \wedge T_q \in D} tu(T_q) = twu(X).$$

It can be seen that some HUIs would not be found based on the transaction-weighted utility of the traditional TWU model. □

Table 5
The profit, *tu* and *ubtu* for the running example.

TID	Transaction (item, profit)	*tu*	*ubtu*
1	$(A, 3.75)$; $(C, 116.0)$; $(E, 8.0)$	127.75	127.75
2	$(D, -5.0)$; $(F, 8.8)$	3.8	8.8
3	$(A, 1.25)$; $(B, 6)$; $(C, 58)$; $(D, -15)$; $(F, 4.4)$	54.65	69.65
4	$(B, 2)$; $(D, -5.0)$; $(F, 13.2)$	10.2	15.2
5	$(B, 2)$; $(C, 104.0)$	106	106
6	$(A, 2.5)$; $(B, 12)$; $(C, 46)$; $(D, -20)$; $(F, 4.4)$	44.9	64.9
7	$(C, 58)$; $(D, -10)$; $(E, 0)$	48.0	58
8	$(B, 4)$; $(E, 12)$	16	16
9	$(B, 4)$; $(D, -5)$; $(F, 4.4)$	3.4	8.4
10	$(A, 5)$; $(B, 2)$; $(D, -5)$; $(E, 16)$	18	23

Property 3. *In the context of HUIM with discount strategies, a HUI may contain item(s) having negative utility values.*

Proof. Let be an itemset X that is a HUI. The itemset X may contain item(s) having a negative utility but still have an overall utility higher or equal to the minimum utility threshold, because of the sum of the utilities of all items in X.

For example, the itemset (CD) shown in Table 4 is a HUI but the utility of its subset (D) is -45 in the database. □

Property 4. *In the context of HUIM with discount strategies, for a given transactional database, the derived HUIs must contain at least an item having a positive utility.*

Proof. Let an itemset X be a HUI. Then, the itemset X must contain at least one item i such that $u(i) > 0$. Otherwise, X has a negative utility and cannot be a HUI. □

To solve the drawbacks of the straightforward TWU model when applied in the proposed algorithms, we propose a new downward closure property, called upper-bound transaction-weighted downward closure (*UBTWDC*) property, described below.

Definition 12 (*Upper-bound transaction utility, ubtu*). The upper-bound transaction utility of a transaction T_q is the sum of the positive utilities of the items in T_q such that $u(i_j, T_q) > 0$. Thus,

$$ubtu(T_q) = \sum_{i_j \in T_q \wedge T_q \in D} u(i_j, T_q), \;\; u(i_j, T_q) > 0.$$

For example, consider Table 5. For each transaction, it shows the profit generated by each item, the transaction utility and the proposed upper-bound transaction utility. For instance, consider transaction T_4. Item (D) has a negative utility (-5) in that transaction. Moreover transaction T_4 has an upper-bound transaction utility equal to $ubtu(T_4) = (2 + 13.2) = 15.2$, which is larger than the utility of T_4 which is $tu(T_4) = 10.2$.

Definition 13 (*Upper-bound transaction-weighted utility, ubtwu*). The upper-bound transaction-weighted utility (UBTWU) of an itemset X is the sum of the upper-bound transaction utilities of transactions containing X, that is:

$$ubtwu(X) = \sum_{X \subseteq T_q \wedge T_q \in D} ubtu(T_q).$$

For example, the upper-bound transaction-weighted utility of an itemset (A) is calculated as $ubtwu(A) = ubtu(T_1) + ubtu(T_3) + ubtw(T_6) + ubtu(T_{10}) = 127.75 + 69.65 + 64.9 + 23 = 285.3$. The upper-bound transaction-weighted utility of an itemset (AB) is calculated as $ubtwu(AB) = ubtu(T_3) + ubtw(T_6) + ubtu(T_{10}) = 157.55$.

Property 5 (*Overestimation of ubtwu*). *The upper-bound transaction-weighted utility of an itemset X according to the proposed model is always an overestimation of the utility of X, i.e.* $ubtwu(X) \geqslant u(X)$.

Proof. The traditional TWU model relies on the transaction-weighted utility (*twu* value), which considers both the positive and negative utilities of items in a transaction. As previously demonstrated, this may lead to underestimating the utilities of itemsets. The proposed upper-bound transaction utility (*ubtu*) only considers the positive utilities of items in a transaction. Because of this, the upper-bound transaction-weighted utility (*ubtwu*) of an itemset is always an upper-bound on its utility. □

Theorem 1 (*Upper-bound transaction-weighted downward closure property, UBTWDC*). *If an itemset X is a HTWUI, all subsets of X are also HTWUI, which indicates that* $ubtwu(X^k) \leqslant ubtwu(X^{k-1})$.

Proof. For any itemset Z, let the notation $TIDs(Z)$ denote the set of transactions containing Z. Let X^k be a k-itemset and X^{k-1} be a $(k$-$1)$-itemset that is a subset of X^k. Assume that X^k and X^{k-1} are HTWUI. Since $X^{k-1} \subseteq X^k$ and the set of $TIDs(X^k \subseteq T_q) \subseteq TIDs(X^{k-1} \subseteq T_q)$, it follows that:

$$ubtwu(X^k) = \sum_{X^k \subseteq T_q \wedge T_q \in D} ubtu(T_q) \leqslant \sum_{X^{k-1} \subseteq T_q \wedge T_q \in D} ubtu(T_q) = ubtwu(X^{k-1}).$$

Therefore, if X^k is a HTWUI, any subset X^{k-1} is also a HTWUI.

The UBTWDC property indicates that any superset of a non-HTWUI is not a HTWUI. That is, only the combinations of high upper-bound transaction-weighted utility $(k$-$1)$-itemsets could be added into the candidate set C_k at each level. According to Theorem 1, we can utilize the UBTWDC property in proposed HTWUIs in Phase I. By holding the UBTWDC property, those overestimated unpromising itemsets will be pruned effectively in Phase I, so that the search space is small. □

Theorem 2 (*HUIs $\subseteq$ HTWUIs*). *Based on the developed upper-bound transaction-weighted utilization downward closure (UBTWDC) property of HUIM with discount strategies, we can demonstrate that HUIs $\subseteq$ HTWUIs, and thus that if an itemset is not a HTWUI, then none of its supersets are HUIs.*

Proof. Let X be any itemset such that $X \subseteq T_q$ for a transaction T_q. The utility of X in T_q is calculated as $u(X, T_q) = \sum_{i_j \in X \wedge X \subseteq T_q} u(i_j, T_q)$, where $u(i_j, T_q)$ may be a positive or negative. Based on definition 3, it can be shown that $u(X, T_q) \leqslant ubtu(T_q)$, and thus that:

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in D} u(X, T_q) \leqslant \sum_{X \subseteq T_q \wedge T_q \in D} ubtu(T_q) = ubtwu(X).$$

Hence, we obtain $u(X) \leqslant ubtwu(X)$. Thus, if an itemset is not a HTWUI, it is not a HUI either. Furthermore, using Theorem 1, we can determine that none of its supersets are HUIs. □

4.2. Proposed HUI-DTP algorithm

In this section, a level-wise HUI-DTP algorithm is proposed to mine HUIs based on its novel downward closure property. The HUI-DTP algorithm consists of two main phases. In the first phase, it uses an Apriori-like search strategy to identify HTWUIs using the *ubtwu* upper-bound at each level. After that, a second phase is performed. It consists of calculating the exact utility of each HTWUI by performing an additional database scan and to output only those that are HUIs. To avoid scanning the full database to calculate the utility of each HTWUI, itemsets are annotated with a vertical TID-list structure [29]. It allows quickly determining the transactions containing an itemset, thus improving the efficiency of the HUI-DTP algorithm. Based on the above definitions and theorems, the pseudo-code of the HUI-DTP algorithm is given in Algorithm 1.

Algorithm 1. HUI-DTP Algorithm.

```
INPUT: $D$, a transactional database; $ptable$, the price table;
  $stable$, the discount strategy table; $\delta$, the user-specified
  minimum utility threshold.
OUTPUT: The set of high-utility itemsets (HUIs).
/* Phase I: scan database, ptable and stable to calculate the
  candidates HTWUIs */
1.    for $T_q \in D \wedge$ each $i_j \in T_q$ do
2.      calculate $ubtu(i_j)$ and construct TID-list($i_j$).
3.    end for
4.    for each $i_j \in D$ do
5.      calculate $ubtwu(i_j)$;
6.      if $ubtwu(i_j) \geqslant \delta$ then
7.        $HTWUI^1 \leftarrow HTWUI^1 \cup i_j$.
8.      end if
9.    end for
10.   set $k \leftarrow 2$.
11.   while $HTWUI^{k-1} \neq \emptyset$ do
12.     $C_k = Apriori(HTWUI^{k-1})$;
13.     for each $k$-itemset $X \in C_k$ do
14.       calculate $ubtwu(X)$ by TID-list($X$);
15.       if $ubtwu(X) \geqslant \delta$ then
16.         $HTWUI^k \leftarrow HTWUI^k \cup X$.
17.       end if
18.     end for
19.     $k \leftarrow k + 1$.
20.   end while
21.   $HTWUIs \leftarrow \bigcup HTWUI^k$.
      /* Phase II: scan database once to find HUIs from HTWUIs */
22.   for each $k$-itemset $X \in HTWUIs$ do
23.     scan $D$ to calculate $u(X)$ by TID-list($X$);
24.     if $u(X) \geqslant \delta$ then
25.       $HUIs \leftarrow HUIs \cup X$.
26.     end if
27.   end for.
28.   return $HUIs$
```

The proposed HUI-DTP algorithm takes as input (1) a transactional database D, (2) the price table *ptable*, (3) the discount strategy table *stable*, and (4) the user-specified minimum utility threshold δ. It first scans the database to calculate the *ubtu* of each transaction, and construct the TID-list of each 1-itemset (Lines 1–3). The *ubtwu* values of all 1-items are also calculated (Line 5) to find the high transaction-weighted utilization 1-items (HTWUI^1, Lines 6–8). The discovered set HTWUI^k (k is initially set as 1) is then used to generate a new set of candidate itemsets C_{k+1} to be used for discovering HTWUI^{k+1}, a process that is repeated in a level-wise way (Lines 10–21). The TID-list structure is used to directly determine the transactions containing itemsets HTWUI^{k+1} without scanning the database (Line 14). The first phase of the HUI-DTP algorithm is completed when no candidates can be generated. In the second phase, an additional database scan is performed to calculate the actual utilities of the remaining HTWUIs, thus revealing the complete set of HUIs (Lines 22–28).

Based on the proposed UBTWDC property and Theorem 1, it can be easily seen that the proposed HUI-DTP algorithm is correct and complete, to find the complete set of HUIs. The algorithm is complete because it only prunes itemsets that are not HTWUIs and the *ubtwu* is an upper bound on the exact utility of itemsets. The algorithm is correct because the exact utility of each HTWUI is computed by scanning the database and only high-utility itemsets are output.

4.3. An illustrated example of HUI-DTP algorithm

In this section, an example is given to illustrate the proposed HUI-DTP algorithm for mining HUIs. Consider the database, price table and discount strategy table respectively shown in Tables 1–3, and that the minimum utility threshold δ is set to 100. After the first database scan, the *ubtu* value of each transaction and the TID-list of each 1-item have been obtained, as shown in Table 5. The *ubtwu* value of each item is then calculated. For instance, consider an item (A). It appears in transactions T_1, T_3, T_6, and T_{10}, the *ubtwu* value of (A) is calculated as $ubtwu(A) = ubtu(T_1) + ubtu(T_3) + ubtu(T_6) + ubtu(T_{10}) = 127.75 + 69.65 + 64 + 23 = 285.30 > 100$. Thus, (A) is considered as a HTWUI^1. The set of discovered HTWUI^1 is shown in Table 6.

After that, k is set to 2. The discovered HTWUI^1 are then used to generate the set of candidates C_2 and obtain HTWUI^2 based on the $Apriori(HTWUI^{k-1})$ function. The set C_2 is shown in Table 7.

The original database is then rescanned to calculate the *ubtwu* values of the generated 2-itemsets of C_2 and obtain HTWUI^2. The result is shown in Table 8.

Then, the other HTWUI^k are obtained by continuing this recursive and level-wise generation of itemsets. The full set of HTWUIs is shown in Table 9. Finally, an additional database scan is performed to find the actual HUIs from the HTWUIs. The final result (the complete set of HUIs) is shown in Table 4.

5. The HUI-DMiner algorithm

In the past, a vertical data structure called utility-list [23] was proposed in the HUI-Miner algorithm to maintain important information from the original database during the mining process. However, the utility-list structure cannot be applied for mining HUIs when considering items with various discount strategies because HUI-Miner does not address the mining problem. In this section, a faster algorithm named HUI-DMiner is proposed for mining HUIs while considering items with various discount strategies. HUI-DMiner introduces a new data structure named Positive-and-Negative Utility-list (PNU-list), which is quite different from the utility-list structure in HUI-Miner [23]. The PNU-list structure is a compact data structure that is used to annotate each item(set). It can be built with a single database scan, and it stores the information required for mining all supersets of an itemset without having to scan the database again.

5.1. Positive-and-Negative Utility-list: PNU-list

The utility-list structure [23] is defined as a vertical structure that is associated to an itemset X. It contains an entry called "element" for each transaction T_q where X appears. Details of the utility-list structure are given below.

Table 6
HTWUIs^1.

1-itemset	*ubtwu*	TID-list
(A)	285.3	{1, 3, 6, 10}
(B)	303.15	{3, 4, 5, 6, 8, 9, 10}
(C)	426.3	{1, 3, 5, 6, 7}
(D)	247.95	{2, 3, 4, 6, 7, 9, 10}
(E)	224.75	{1, 7, 8, 10}
(F)	166.95	{2, 3, 4, 6, 9}

Table 7
The set C_2, obtained from HTWUI[1].

2-itemset	*ubtwu*	TID-list
(*AB*)	156	{3, 6, 10}
(*AC*)	260	{1, 3, 6}
(*AD*)	156	{3, 6, 10}
(*AE*)	150	{1, 10}
(*AF*)	133	{3, 6}
(*BC*)	239	{3, 5, 6}
(*BD*)	179	{3, 4, 6, 9, 10}
(*BE*)	39	{8, 10}
(*BF*)	156	{3, 4, 6, 9}
(*CD*)	191	{3, 6, 7}
(*CE*)	185	{1, 7}
(*CF*)	133	{3, 6}
(*DE*)	81	{7, 10}
(*DF*)	164	{2, 3, 4, 6, 9}

Table 8
HTWUI[2].

2-itemset	*ubtwu*	TID-list
(*AB*)	156	{3, 6, 10}
(*AC*)	260	{1, 3, 6}
(*AD*)	156	{3, 6, 10}
(*AE*)	150	{1, 10}
(*AF*)	133	{3, 6}
(*BC*)	239	{3, 5, 6}
(*BD*)	179	{3, 4, 6, 9, 10}
(*BF*)	156	{3, 4, 6, 9}
(*CD*)	191	{3, 6, 7}
(*CE*)	185	{1, 7}
(*CF*)	133	{3, 6}
(*DF*)	164	{2, 3, 4, 6, 9}

Definition 14. Given an itemset X and a transaction (or itemset) T such that $X \subseteq T$, the set of all items from T that are not in X is denoted as $T \setminus X$, and the set of all the items appearing after X in T is denoted as T/X. Thus, $T/X \subseteq T \setminus X$.

For example, consider transaction T_3 in Table 1. If $X = \{BC\}$, then, $T_3 \setminus X = \{A, D, F\}$, and $T_3/X = \{D, F\}$.

Definition 15 (*Utility-list*). For an itemset X, the utility-list of X in a database D contains an entry (element) for each transaction T_q where X appears ($X \subseteq T_q \in D$). An entry contains three fields respectively indicating: (1) the transaction identifier of T_q (w.r.t. ***tid***), (2) the utility of X in T_q (*w.r.t.* ***iu***), i.e. $u(X, T_q)$, and (3) the remaining utility of X in T_q (w.r.t. ***ru***), defined as $X.ru(T_q) = \sum_{i_j \in (T_q/X)} u(i_j, T_q)$ [23].

Table 9
The whole set of HTWUIs for the running example.

k-itemset	*twu*	TID-list	*k*-itemset	*twu*	TID-list
(*A*)	285.3	{1, 3, 6, 10}	(*ABC*)	134.55	{3, 6}
(*B*)	303.15	{3, 4, 5, 6, 8, 9, 10}	(*ABD*)	157.55	{3, 6, 10}
(*C*)	426.3	{1, 3, 5, 6, 7}	(*ABF*)	134.55	{3, 6}
(*D*)	247.95	{2, 3, 4, 6, 7, 9, 10}	(*ACD*)	134.55	{3, 6}
(*E*)	224.75	{1, 7, 8, 10}	(*ACE*)	127.75	{1}
(*F*)	166.95	{2, 3, 4, 6, 9}	(*ACF*)	134.55	{3, 6}
(*AB*)	157.55	{3, 6, 10}	(*ADF*)	134.55	{3, 6}
(*AC*)	262.3	{1, 3, 6}	(*BCD*)	134.55	{3, 6}
(*AD*)	157.55	{3, 6, 10}	(*BCF*)	134.55	{3, 6}
(*AE*)	150.75	{1, 10}	(*BDF*)	158.15	{3, 4, 6, 9}
(*AF*)	134.55	{3, 6}	(*CDF*)	134.55	{3, 6}
(*BC*)	240.55	{3, 5, 6}	(*ABCD*)	134.55	{3, 6}
(*BD*)	181.15	{3, 4, 6, 9, 10}	(*ABCF*)	134.55	{3, 6}
(*BF*)	158.15	{3, 4, 6, 9}	(*ABDF*)	134.55	{3, 6}
(*CD*)	192.55	{3, 6, 7}	(*ACDF*)	134.55	{3, 6}
(*CE*)	185.75	{1, 7}	(*BCDF*)	134.55	{3, 6}
(*CF*)	134.55	{3, 6}	(*ABCDF*)	134.55	{3, 6}
(*DF*)	166.95	{2, 3, 4, 6, 9}			

Based on the utility-list structure, HUI-Miner performs a depth-first search in the enumeration tree of itemsets and uses a pruning criterion to avoid exploring branches. Although this approach works fine to discover HUIs, it cannot be applied in the HUI mining framework with discount strategies.

Observation 2. The HUI-Miner algorithm may not discover the complete set of HUIs using the utility-list structure if it is applied on a database containing item(s) having negative profit value(s).

Proof. Let X^{k-1} be an itemset of length $k-1$, and X^k be an itemset of length k that is a superset of X^{k-1}. Let the notation *X.IU* and *X.RU* respectively denotes the sum of *iu* and *ru* values in the utility-list of an itemset X. It can be shown that $u(X)$ = *X.IU* for any itemset X. Since $X^k.\ iu(T_q) \leqslant X^{k-1}.\ iu(T_q) + X^{k-1}.\ ru(T_q)$ is obtained in each transaction, HUI-Miner utilizes the inequation $X^k.\ IU \leqslant X^{k-1}.\ IU + X^{k-1}.\ RU$ to prune the search space. However, it cannot find the complete set of HUIs in the proposed framework. We demonstrate this with an example. Consider transaction T_6 in Table 5. We have that $AB.iu(T_6) + AB.ru(T_6) = (2.5 + 12) + (46 - 20 + 4.4) = 44.9$. However, the inequation $X^k.\ iu(T_q) \leqslant X^{k-1}.\ iu(T_q) + X^{k-1}.\ ru(T_q)$ does not hold since $ABC.iu(T_6) = (2.5 + 12 + 46) = 60.5 > 44.9$. Besides, the upper-bound used by the pruning strategy in HUI-Miner is less than the original *twu* upper-bound in Two-Phase, as mentioned in observation 1. But the *twu* value is not sufficient for finding the complete set of HUIs. Thus, this observation holds. □

To be able to mine HUIs while considering items with various discount strategies, we designed a new data structure, called Positive-and-Negative Utility-list (PNU-list), which is quite different from the previous utility-list structure [23]. Like utility-lists, PNU-lists are vertical compact data structures that are each associated to an itemset X.

Definition 16 (*Positive-and-Negative Utility-list, PNU-list*). The PNU-list of an itemset X is denoted as *X.PNUL*. It contains an entry (element) for each transaction T_q where X appears ($X \subseteq T_q \in D$). An element consists of four fields: (1) the transaction identifier of T_q (denoted ***tid***), (2) the utility of X in T_q (denoted ***iu***), denoted as $X.iu(T_q)$, (3) the negative utility of X in T_q (denoted ***nu***), which is defined as $X.nu(T_q) = \sum_{i_j \in T_q} u(i_j, T_q), u(i_j, T_q) < 0$, and (4) the remaining positive utility of X in T_q (denoted ***rpu***), defined as $X.rpu(T_q) = \sum_{i_j \in (T_q/X) \wedge u(i_j, T_q) > 0} u(i_j, T_q)$, such that only positive utility values of the remaining items are included in ***rpu***.

For example, the PNU-lists of 1-items are shown in Fig. 1. Note that before PNU-lists are constructed, it is assumed that items in transactions have been sorted according to the ascending order of *ubtwu* values. In our example, this order is $\{F < E < D < A < B < C\}$. To build these initial PNU-lists, two databases scans are required, one to calculate the *ubtwu* of each 1-item and another one to sort transactions and create PNU-lists.

PNU-lists for k-itemsets ($k > 1$) can be constructed from PNU-lists of $(k-1)$ itemsets without scanning the database. The proposed HUI-DMiner algorithm performs this construction during its depth-first search using the construction algorithm shown in Algorithm 2. Let the notation X_a denotes the itemset $X \cup \{a\}$, where a is an item and X is an itemset. The PNU-list construction procedure takes as input the PNU-lists of some itemsets X, X_a and X_b, and outputs the PNU-list of X_{ab}.

Algorithm 2. PNU-list construction.

```
INPUT: $X.PNUL$, the PNU-list of an itemset $X$;
       $X_a.PNUL$, the PNU-list of an itemset $X_a$;
       $X_b.PNUL$, the PNU-list of an itemset $X_b (X_a \neq X_b)$.
OUTPUT: $X_{ab}.PNUL$, the PNU-list of an itemset $X_{ab}$.
 1.  $X_{ab}.PNUL \leftarrow \emptyset$.
 2.  for each element $E_a \in X_a.PNUL$ do
 3.    if $\exists E_b \in X_b.PNUL \wedge E_a.tid = E_b.tid$ then
 4.      if $X.PNUL \neq \emptyset$ then
 5.        find element $E \in X.PNUL$ such that $E.tid = E_a.tid$;
 6.        $E_{ab} \leftarrow < E_a.tid, E_a.iu + E_b.iu - E.iu, E_a.nu + E_b.nu - E.nu, E_b.rpu >$.
 7.      else
 8.        $E_{ab} \leftarrow < E_a.tid, E_a.iu + E_b.iu, E_a.nu + E_b.nu, E_b.rpu >$.
 9.      end if
10.      $X_{ab}.PNUL \leftarrow X_{ab}.PNUL \cup E_{ab}$.
11.    end if
12.  end for
13.  return $X_{ab}.PNUL$.
```

For optimization purposes, the "find" operation in the construction procedure can be optimized using the binary search (if entries in PNU-lists are arranged according to the order of increasing transactions identifiers). The constructed PNU-lists for 2-itemsets with (F) as prefix item are shown in Fig. 2.

To construct the PNU-lists of k-itemset $\{i_1, \ldots, i_{k-1}, i_k\}$ ($k \geqslant 3$), the construction procedure is applied on pairs of ($k-1$)-itemsets sharing the same prefix of $k-2$ items. For example, the PNU-list of (FDA) can be built by applying the construction procedure using the PNU-lists of (FD) and (FA). The resulting PNU-list for (FDA) is shown in Fig. 3.

Definition 17 (*Sum of the utilities of an itemset*). For an itemset X, let $X.IU$ is the sum of iu values in the PNU-list of X, that is:

$$X.IU = \sum_{X \subseteq T_q \wedge T_q \in D} X.iu(T_q).$$

Definition 18 (*Sum of the negative utilities of an itemset*). For an itemset X, let $X.NU$ is the sum of nu values in the PNU-list of X, that is:

$$X.NU = \sum_{X \subseteq T_q \wedge T_q \in D} X.nu(T_q).$$

Definition 19 (*Sum of the remaining positive utilities of an itemset*). For an itemset X, let $X.RPU$ is the sum of rpu values in the PNU-list of X, that is:

$$X.RPU = \sum_{X \subseteq T_q \wedge T_q \in D} X.rpu(T_q).$$

Definition 20 (*Sum of positive utilities of an itemset*). For an itemset X, let $X.PU$ is the sum of the positive profit values in X, defined as:

$$X.PU = X.IU - X.NU.$$

For example, the PNU-list of itemset (FD) appears in transactions having respectively 2, 3, 4, 6 and 9 as TIDs. According to this PNU-list, we have $FD.IU = 3.8 + (-10.6) + 8.2 + (-15.6) + (-0.6) = (-14.8)$; $FD.NU = (-5) + (-15) + (-5) + (-20) + (-5) = (-50)$; $FD.RPU = 0 + 65.25 + 2 + 60.5 + 4 = 131.75$; and $FD.PU = FD.IU - FD.NU = (-14.8) - (-50) = 35.2$.

5.2. Pruning strategies

The HUI-DMiner algorithm mines HUIs using a depth-first search and utilizes the PNU-list structure to calculate the utility of each itemset. The search space of the proposed HUI-DMiner algorithm to mine HUIs can be regarded as an enumeration tree. For any itemset encountered during the search, HUI-DMiner verifies two pruning conditions to determine if child nodes of the current itemset should be explored. These pruning conditions greatly reduce the size of the search space by detecting branches of the search tree that will only lead to low-utility itemsets early. We next introduce these pruning conditions.

Property 6 (*Extended itemsets with positive utilities*). *Let X be an itemset and Y be an item/itemset having a positive utility value such that $Y \notin X$. It follows that $u(X \cup Y) < u(X)$ or $u(X \cup Y) \geqslant u(X)$.*

Rationale. As mentioned previously, the well-known support downward closure property cannot be applied in HUIM since the utility is not anti-monotonic. It is clear that if an itemset Y is appended to an itemset $X = \{i_1, i_2, \ldots, i_k\}$, the resulting itemset $(X \cup Y)$ can only appear in as much as or less transactions than X. Thus, appending an itemset Y having a positive utility value to X, will result in an itemset $(X \cup Y)$ having a utility that may be lower, equal or higher than the utility of X.

Property 7 (*Relationship between positive utility and negative utility within an itemset*). *For any itemset X, let $pu(X)$ and $nu(X)$ respectively denote the sum of positive utilities and the sum of negative utilities of items in X, such that $u(X) = pu(X) + nu(X)$. The following relationship holds: $nu(X) \leqslant u(X) \leqslant pu(X)$.*

Rationale. Items with positive utility values in X can only be used to increase the utility of X, while items with negative utility values can only be used to decrease the utility of X. Thus, the property $nu(X) \leqslant u(X) \leqslant pu(X)$ holds.

(F)			
2	8.8	0	0
3	4.4	0	65.25
4	13.2	0	2
6	4.4	0	60.5
9	4.4	0	4

(E)			
1	8	0	119.75
7	0	0	58
8	12	0	4
10	16	0	7

(D)			
2	-5	-5	0
3	-15	-15	65.25
4	-5	-5	2
6	-20	-20	60.5
7	-10	-10	58
9	-5	-5	4
10	-5	-5	7

tid *iu* *nu* *rpu*

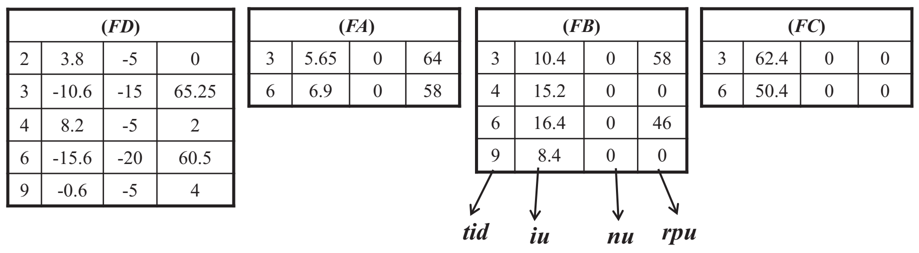
(A)			
1	3.75	0	116
3	1.25	0	64
6	2.5	0	58
10	5	0	2

(B)			
3	6	0	58
4	2	0	0
5	2	0	104
6	12	0	46
8	4	0	0
9	4	0	0
10	2	0	0

(C)			
1	116	0	0
3	58	0	0
5	104	0	0
6	46	0	0
7	58	0	0

$ubtwu(F) < ubtwu(E) < ubtwu(D) < ubtwu(A) < ubtwu(B) < ubtwu(C)$

Fig. 1. Constructed PNU-list of 1-items.



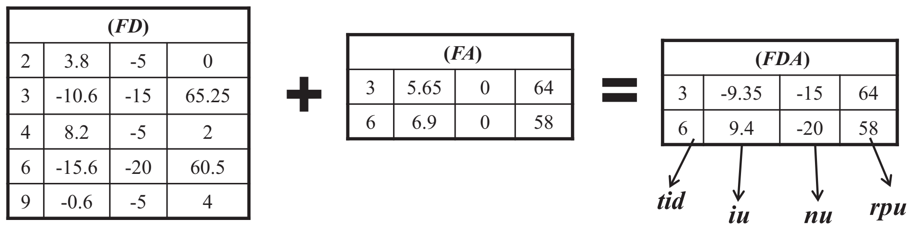
(FD)			
2	3.8	-5	0
3	-10.6	-15	65.25
4	8.2	-5	2
6	-15.6	-20	60.5
9	-0.6	-5	4

(FA)			
3	5.65	0	64
6	6.9	0	58

(FB)			
3	10.4	0	58
4	15.2	0	0
6	16.4	0	46
9	8.4	0	0

(FC)			
3	62.4	0	0
6	50.4	0	0

Fig. 2. Constructed PNU-list of 2-itemsets having (*F*) as prefix item.



(FD)			
2	3.8	-5	0
3	-10.6	-15	65.25
4	8.2	-5	2
6	-15.6	-20	60.5
9	-0.6	-5	4

(FA)			
3	5.65	0	64
6	6.9	0	58

(FDA)			
3	-9.35	-15	64
6	9.4	-20	58

Fig. 3. Constructed PNU-list of the 3-itemset (*FDA*).

Based on the above properties and the designed PNU-list structure, two efficient pruning strategies are presented below to prune the search space.

Lemma 1 (*Anti-monotonicity of unpromising itemsets*). *For any node X in the search space (w.r.t. an enumeration tree), the sum of X.PU and X.RPU in the PNU-list of X is larger than or equal to the utility of any one of its descendents.*

Proof. Let X^{k-1} be an itemset of length $k-1$, and X^k be an itemset of length k that is a superset of $X^{k-1.}$ Assume that X^k is a descendent of X^{k-1} in the enumeration tree, meaning that X^{k-1} is the prefix of X^k. Let the set of items in X^k but not in X^{k-1} be denoted as $(X^k - X^{k-1}) = (X^k / X^{k-1})$. Note that *pu* is set to a positive utility. For any transaction $X \subseteq T_q$:

$$\because X^{k-1} \subset X^k \subseteq T_q \Rightarrow (X^k/X^{k-1}) \subseteq (T_q/X^{k-1})$$

$$\begin{aligned}
\therefore \text{ in } T_q, X^k.iu(T_q) &= X^{k-1}.iu(T_q) + (X^k/X^{k-1}).iu(T_q)\\
&= X^{k-1}.iu(T_q) + \sum_{z\in(X^k/X^{k-1})} z.iu(T_q)\\
&\leqslant X^{k-1}.iu(T_q) + \sum_{z\in(X^k/X^{k-1})} z.pu(T_q)\\
&\leqslant X^{k-1}.iu(T_q) + \sum_{z\in(T_q/X^{k-1})} z.pu(T_q)\\
&= X^{k-1}.iu(T_q) + X^{k-1}.rpu(T_q)\\
&\leqslant X^{k-1}.pu(T_q) + X^{k-1}.rpu(T_q).
\end{aligned}$$

$$\therefore \text{ in } T_q, X^k.iu(T_q) \leqslant X^{k-1}.pu(T_q) + X^{k-1}.rpu(T_q)$$

$$\because X^{k-1} \subset X^k \Rightarrow X^k.tids \subseteq X^{k-1}.tids$$

$$\begin{aligned}
\therefore \text{ in } D, X^k.IU &= \sum_{T_q\in X^k.tids} X^k.iu(T_q)\\
&\leqslant \sum_{T_q\in X^k.tids} (X^{k-1}.pu(T_q) + X^{k-1}.rpu(T_q))\\
&\leqslant \sum_{T_q\in X^{k-1}.tids} (X^{k-1}.pu(T_q) + X^{k-1}.rpu(T_q))\\
&= X^{k-1}.PU + X^{k-1}.RPU.
\end{aligned}$$

$$\therefore \text{ in } D, X^k.IU \leqslant X^{k-1}.PU + X^{k-1}.RPU.$$

Thus, the sum of utilities of X^k in D is always less than or equals to the sum of X^{k-1}. *PU* and X^{k-1}. *RPU* of X^{k-1}. □

Pruning strategy 1. *Let X be an itemset (node) encountered during the depth-first search of the enumeration tree. If the sum of X.PU and X.RPU of the itemset/node X according its constructed PNU-list structure is less than the minimum utility value, then none of thedescendent node X is a HUI, and this part of the search space can be pruned.*

Rationale. This pruning condition directly follows from Lemma 1. According to Lemma 1, if the sum of *X.PU* and *X.RPU* is less than the minimum utility value for a given itemset (node) *X*, then all its descendent nodes are low-utility itemsets and can be ignored.

Lemma 2. *In the proposed HUI-DMiner algorithm, the processing order of items will not affect the final set of HUIs found.*

Proof. From Lemma 1, it can be seen that the processing order does not affect the pruning strategy. For any order, the ***iu***, ***nu***, and ***rpu*** fields of PNU-list can be calculated to correctly find the complete set of HUIs. The processing order of items in the enumeration tree can, however, may affect the efficiency of the proposed HUI-DMiner algorithm. For the HUI-Miner algorithm, using the ascending order of TWU was shown to be more efficient than using other orders such as the descending order of TWU or the lexicographic order [23]. Thus, the ascending order of *ubtwu* values is adopted in the proposed HUI-DMiner algorithm.

Besides, to further improve the efficiency of HUI-DMiner, we have integrated the Estimated Utility Co-occurrence Pruning (EUCP) strategy proposed in FHM [12]. This strategy allows pruning extensions of itemsets without constructing the PNU-list structure of their supersets. Note that the strategy is applied differently than in FHM. Here, the *ubtwu* value is used to estimate the relationships of 2-itemsets instead of the *twu* value. We name HUI-DEMiner, the version of HUI-DMiner that integrates this strategy. □

Lemma 3 (*Estimated Utility Co-occurrence Pruning strategy, EUCP*). *Based on the proposed ubtwu measure, if a 2-itemsets X is not a HTWUI, any supersets of X will not be a HTWUI or HUI.*

Proof. Let X be a 2-itemset and X^k be any k-itemsets ($k \geqslant 3$) that is a superset of X. From property 1, $ubtwu(X^k) \leqslant ubtwu(X^{k-1})$. Thus, if X is not a HTWUI, any k-itemset ($k \geqslant 3$) that is a superset of X will not be a HTWUI or HUI. □

Pruning strategy 2. *Let Y be an itemset/node (k-itemset, $k \geqslant 3$) encountered during the depth-first search of the enumeration tree. If the ubtwu of a 2-itemset $X \subseteq Y$ according to the constructed EUCS is less than the minimum utility threshold, then Y is not a HUI, and none of its descendent nodes are HUIs. It is thus unnecessary to construct the PNU-list of Y and its descendents.*

Rationale. According to Lemma 3, this pruning strategy is correct.

Based on the EUCP strategy, a huge number of unpromising k-itemset ($k \geqslant 2$) can be pruned. The constructed EUCS for the running example is given in Table 10. Note that the *ubtwu* values of 2-itemsets stored in the EUCS can be calculated during the second database scan.

The EUCS can be represented as a triangular matrix as shown in Table 10 for the running example. Considering the 2-itemsets (AB) and (DE), based on Table 5, their *ubtwu* are calculated as $ubtwu(AB) = ubtu(T_3) + ubtu(T_6) + ubtu(T_{10}) = 69.65 + 64.9 + 23 = 157.55$, and $ubtwu(DE) = ubtu(T_7) + ubtu(T_{10}) = 58 + 23 = 81$. These values are stored in the designed EUCS.

5.3. Proposed HUI-DEMiner algorithm

Based on the designed PNU-list structure and the above Theorems, the developed HUI-DEMiner algorithm is described in Algorithm 3.

Algorithm 3. HUI-DEMiner algorithm.

```
INPUT: $D$, a transactional database; $ptable$, the price table;
  $stable$, the strategies table; $\delta$, the user-specified minimum
  utility threshold.
OUTPUT: The set of high-utility itemsets (HUIs).
/* $D.PNUL$, the set of PNU-list of $D$ */;
/* $X.PNUL$, the PNU-list of itemset $X$ */;
/* $EUCS$, the Estimated Utility Co-Occurrence Structure */
1.  $D.PNUL \leftarrow \emptyset$, $X.PNUL \leftarrow \emptyset$, $EUCS \leftarrow \emptyset$;
2.  scan $D$ to calculate the $ubtwu$ value of each item $i \in I$.
3.  find $I^* \leftarrow \{i \in I | ubtwu(i) \geqslant \delta\}$;
4.  sort $I^*$ in $ubtwu$ ascending order;
5.  scan $D$ to construct the $X.PNUL$ for each item $i \in I^*$ and
    build the $EUCS$;
6.  $D.PNUL \leftarrow \cup X.PNUL$;
7.  call HUI-DEMining($\emptyset$, $I^*$, $\delta$, $EUCS$);
8.  return $HUIs$
```

The HUI-DEMiner algorithm takes as input: (1) a quantitative transactional database D, (2) the user-specified price table *ptable*, (3) the strategies table *stable*, and (4) the minimum utility threshold δ. It first scans the database D to calculate the *ubtwu* value of each item (Line 2) and identify the set of HTWUI[1], such that $I^* \leftarrow \{i \in I | ubtwu(i) \geqslant \delta\}$ (Line 3, pruning strategy by Theorem 1). The discovered HTWUI[1] (I^*) are sorted in ascending order of their *ubtwu* value (Line 4). The database is then rescanned to construct the PNU-list structure of each item $i \in I^*$ and build the *EUCS* (Line 5). After that, the search procedure **HUI-DEMining** is called recursively to discover HUIs using a depth-first search mechanism (Line 7). The **HUI-DEMining** procedure is presented in Algorithm 4.

Algorithm 4. HUI-DEMining(X, *extenX*, δ, *EUCS*).

```
INPUT: $X$, an itemset; $extenX$, the set of PNU-lists for all 1-
  extensions of $X$; $\delta$, the minimum utility threshold; the $EUCS$.
OUTPUT: The set of high-utility itemsets (HUIs).
1.   for each item $X_a \in extenX$ do
2.     if $X_a.IU \geqslant \delta$ then
3.       $HUIs \leftarrow HUIs \cup X_a$;
4.     end if
5.     if ($X_a.IU - X_a.NU + X_a.RPU \geqslant \delta$) then
6.       $extenX_a \leftarrow \emptyset$;
7.       for each $X_b$ after $X_a$ in $extenX$ do
8.         if $\exists\, ubtwu\,(a,b) \in EUCS \wedge ubtwu(a,b) \geqslant \delta$ then
9.           $X_{ab} \leftarrow X_a \cup X_b$;
10.          $X_{ab}.PNUL \leftarrow$ Construct($X$, $X_a$, $X_b$);
11.          $extenX_a \leftarrow extenX_a \cup X_{ab}.PNUL$;
12.        end if
13.      end for
14.      call HUI-DEMining($X_a$, $extenX_a$, $\delta$, $EUCS$);
15.    end if
16.  end for
17.  return $HUIs$
```

The HUI-DEMining procedure takes as input: (1) an itemset X, (2) the set of PNU-lists for all 1-extensions of X (*extenX*), (3) the minimum utility threshold δ, and (4) the *EUCS*. It performs a loop over each 1-itemset X in *D.PNUL* to determine if X is a HUI (Lines 2–4). Then, the algorithm checks the pruning condition ($X.IU - X.NU + X.RPU \geqslant \delta$) to check whether extensions of X should be explored using the depth-first search (Line 5, pruning strategy 2). If yes, each extension X_{ab} of X_a according to *D.PNUL* is considered. If the *ubtwu* value in the constructed *EUCS* is no less than the minimum utility threshold δ for the itemset $\{X_a \cup X_b\}$ (Line 8), then the itemset $X_{ab} = \{X_a \cup X_b\}$ is considered; otherwise, it is not, thus reducing the search space for mining HUIs (Lines 8–12, pruning strategy 2). The construction procedure of PNU-list structure is performed to construct the set of PNU-list structures of all extensions ($extenX_a$) of X_a (Lines 10–11). Finally, the HUI-DEMiner algorithm is called to recursively mine HUIs until no itemsets are generated (Line 14).

The proposed HUI-DMiner and HUI-DEMiner algorithms are efficient algorithms. Their strategies allow pruning unpromising itemsets without constructing the PNU-list structure of several supersets. The execution time can thus be greatly reduced by avoiding the construction of many PNU-lists and also by pruning part of the search space in the enumeration tree.

Lemma 4. *The final set of HUIs derived by the proposed HUI-DMiner and HUI-DEMiner algorithms are correct and complete.*

Proof. By Lemmas 1–3, the HUI-DMiner and HUI-DEMiner algorithms will not discard any HUIs (**completeness**). Furthermore, by calculating the utility of itemsets using the PNU-list structure, only HUIs will be output (**correctness**). Therefore, it can be

Table 10
Constructed EUCS.

Item	*A*	*B*	*C*	*D*	*E*
B	157.55	–	–	–	–
C	262.3	240.55	–	–	–
D	157.55	181.15	192.55	–	–
E	150.75	39.0	185.75	81	–
F	134.55	158.15	134.55	166.95	0

concluded that the HUI-DMiner and HUI-DEMiner algorithms are correct and complete based on the PNU-list structure and the developed pruning strategies. □

5.4. An illustrated example of the HUI-DEMiner algorithm

We next presented an illustrated example of the HUI-DEMiner algorithm execution. The database and parameters used in this example are the same as in Section 4.3.

The HUI-DEMiner algorithm first scans the database to find the set of HTWUI^1. For any 1-item $i \in I$ satisfying the condition $ubtwu(i) \geqslant \delta$, put it into the set of $\text{HTWUI}^1(I^*)$. In this example, this set is equal to {A: 285.3; B: 303.15; C: 426.3; D: 247.95; E: 224.75; F: 166.95}. Then, the set I^* is sorted by *ubtwu* ascending order, which gives $\{F < E < D < A < B < C\}$. After that, the database is rescanned to extract all the necessary information (i.e., ***tid***, ***iu***, ***nu*** and ***rpu***) to construct the PNU-lists of each item $i \in I^*$ and build the *EUCS*, as shown in Algorithm 2.

In this example, the 1-itemset (F) is first processed. Since $F.IU = 8.8 + 4.4 + 13.2 + 4.4 + 4.4 = 35.2 < 100$, the itemset (F) is not output as a HUI. The sum of $(F.IU - F.NU + F.RPU)$ is calculated as: $35.2 - 0 + 126.75 = 160.95$, which is no less than the minimum utility threshold (= 100). Thus, child nodes of (F) may be HUIs. The depth-first search is thus performed to explore child nodes of (F). Based on the processing order of items in HTWUI^1, the first 2-itemset (FE) is processed, while (FE) does not exist in database because the set of tids of (FE) is empty. Then, (FD) is constructed, as shown in Fig. 2. Based on the PNU-list structure of (FD), the utility of (FD) in database is calculated as $FD.IU = 3.8 + (-10.6) + 8.2 + (-15.6) + (-0.6) = (-14.8)$; $FD.NU = (-5) + (-15) + (-5) + (-20) + (-5) = (-50)$; $FD.RPU = 0 + 65.25 + 2 + 60.5 + 4 = 131.75$. Thus, (FD) is not a HUI. The child nodes of (FD) are, however, explored by the depth-first search since $FD.IU - FD.NU + FD.RPU = (-14.8) - (-50) + 131.75 = 166.95 > 100$. The PNU-list structures of 3-itemset (FDA) having (FD) as prefix are constructed and shown in Fig. 3. The other nodes are processed in the same way. When no more nodes can be explored in the enumeration tree, the algorithm terminates and the complete set of HUIs is output by the HUI-DEMiner algorithm without an additional database scan. The final set of HUIs found by the algorithm is shown in Table 4.

6. Experimental evaluation

An extensive experimental study was performed to assess the performance of the proposed algorithms in terms of execution time, number of patterns generated and memory consumption. We evaluated the performance of the proposed algorithms (HUI-DTP, HUI-DMiner, and HUI-DEMiner) using four datasets including three real-life datasets [1] (foodmart dataset was acquired from Microsoft foodmart 2000) and a synthetic dataset [4]. Note that the three-phase algorithm [19] cannot discover the complete set of HUIs when considering items with various discount strategies are used, and that the FHN algorithm [11] was designed to solve the problem of mining HUIs with negative profit values, but that it cannot be used for mining HUIs with various discount strategies. Hence, only the three-phase algorithm is compared with the proposed algorithms. Note that we have also improved the efficiency of the three-phase algorithm by adopting the TID-list structure, thus speeding up that algorithm for mining HUIs.

All algorithms were implemented in Java and experiments were carried done on a computer equipped with an Intel Core2 Duo 2.8G Hz processor and 4 GB of RAM, running the 64 bit Windows 7 operating system. The quantities and price values of items were generated for all datasets except foodmart. The parameters for dataset generation are the same as in previous studies, such as [24]. Quantities were generated in [1,5] interval. The cost price was randomly assigned to each item in [1, 1000] interval. Note that the distribution of the cost prices of all items satisfies the Pareto distribution. Thus, a large amount of cheap products and fewer expensive products have been generated for each of the three datasets. The tag price of each item is initially assigned as 1.5 times its cost price. The discount strategies are set following a normal distribution. For example, consider the first strategy. Discounts of 0% and 100% are quite rare, while most items were sold at discount of around (0%,100%). Parameters and characteristics of the four datasets are respectively presented in Tables 11 and 12.

6.1. Runtime

We first run all the algorithms on the four datasets to compare the runtime of the proposed algorithm with the three-phase algorithm, when the minimum utility threshold (*MU*) is varied. Results are shown in Fig. 4.

It can be observed that the proposed HUI-DMiner and HUI-DEMiner algorithms are up to two or three orders of magnitude faster than the three-phase and HUI-DTP algorithms for all four datasets. Furthermore, we can see that the runtime decreases as the *MU* is increased. This is reasonable since fewer candidates HUIs are generated when the *MU* is set higher. When the *MU* is set lower, the gap between the proposed HUI-DMiner and HUI-DEMiner algorithms and the three-phase algorithm becomes larger, which indicates that the three-phase algorithm requires more computations than the proposed HUI-DMiner and HUI-DEMiner algorithms. Since these latter algorithms both use the PNU-list structure and similar pruning strategies, there is no a very large performance difference between them. The three-phase and HUI-DTP algorithms extend the traditional TWU model, which requires to repeatedly perform time-consuming database scans to calculate the utility of the candidate itemsets. Although the TID-list structure was applied to speed up the three-phase algorithm, it still suffers from the well-known combinatorial explosion problem when the *MU* threshold is decreased or when dense datasets are used. This can be observed in Fig. 4(c) for the mushroom dataset, a dataset containing long transactions. The runtime of the three-phase and HUI-DTP algorithms on this dataset is more than 10^4 seconds. The reason is that a huge amount of HTWUIs is generated and that costly database scans need to be performed to calculate their utilities.

In Fig. 4, it can also be observed that the three-phase algorithm is faster than the proposed HUI-DTP algorithm. The reason is that the three-phase algorithm uses the traditional TWU model to find HUIs based on its underestimated transaction-weighted utility measure while the HUI-DTP algorithm uses the redefined TWU model with the higher upper-bound transaction-weighted utility measure to be able to mine the complete set of HUIs. The higher upper-bound transaction-weighted utility measure is more expensive to compute. However, this latter can be used to find the complete set of HUIs since it always overestimates the utility, while the former one may miss some HUIs. Besides, an interesting observation is that HUI-DEMiner has similar performance for the mushroom dataset as shown in Fig. 4(c) when *MUs* are respectively set from 3000 K to 3800 K, using 200 K increments. For the foodmart dataset, HUI-DMiner performs even better than HUI-DEMiner. The reason is that for very sparse dataset such as foodmart, which have short transactions (average length of 4.4 items), many unpromising candidates can be pruned using the pruning strategies. But the EUCP strategy does not allows pruning many candidates and constructing the EUCS is costly. For the dense mushroom dataset, all candidate itemsets have high transaction-weighted utilities. Thus, the EUCP strategy cannot prune many

Table 11
Parameter Description.

#|D|	Total number of transactions
#|I|	Number of distinct items
AvgLen	Average length of transactions
MaxLen	Maximal length of transactions

Table 12
Characteristics of used datasets.

Dataset	#|D|	#|I|	AvgLen	MaxLen	Type
foodmart	21,556	1559	4	11	Sparse
retail	88,162	16,470	10.3	76	Sparse
mushroom	8124	119	23	23	Dense
T10I4D100K	100,000	870	10.1	29	Sparse

itemsets efficiently to reduce the search space when datasets are too sparse or too dense.

6.2. Number of candidates and HUIs

The numbers of generated candidates (HTWUIs) and HUIs have also been measured to compare the performance of the proposed algorithms. The three-phase and HUI-DTP algorithms generate HTWUIs (denoted as HTWUIs* and HTWUIs, respectively) and HUIs (denoted as HUIs* and HUIs, respectively). The proposed HUI-DMiner and HUI-DEMiner algorithms directly mine the HUIs (denoted as HUIs) without candidate generation. Experiments have been conducted to compare the number of patterns under various *MUs* with fixed discount strategies. The results are shown in Fig. 5.

From Fig. 5, it can be observed that the number of HTWUIs* and HUIs* for the three-phase algorithm are respectively less than the number of HTWUIs and HUIs for the proposed algorithms. The gap becomes larger when the *MU* decreases. As mentioned before, the three-phase algorithm uses the traditional TWU model to mine HTWUIs* and HUIs*. The discovered HUIs* are, however, not complete since the three-phase algorithm underestimates the transaction-weighted utilities of itemsets to reveal HTWUIs*. From Fig. 5(a), it can be observed that the proposed algorithms generate more HUIs than the number of HUIs* found by the three-phase algorithm. Thus, the three-phase algorithm cannot perform the mining task of HUIM when considering items with various discount strategies.

From Fig. 5(b), it can also be observed that the number of HTWUIs* and HTWUIs increase dramatically when the *MU* threshold is decreased. Although the TWDC property of the TWU model is adopted to prune unpromising candidate itemsets, the three-phase and HUI-DTP algorithms still perform a level-wise search to generate candidates to then respectively derive HUIs* and HUIs. On the other hand, the HUI-DMiner and HUI-DEMiner algorithms adopt the proposed PNU-list structure to avoid performing database scans and significantly prune the number of itemsets to be considered to discover HUIs. In general, if more itemsets are considered than the more time will be required to mine HUIs and the more memory will be used to maintain the discovered patterns. The memory consumption of the mining algorithms are analyzed next.

6.3. Number of potential candidates

We have compared the number of candidates HTWUIs generated by the HUI-DTP algorithm (denoted as N_1), the number nodes

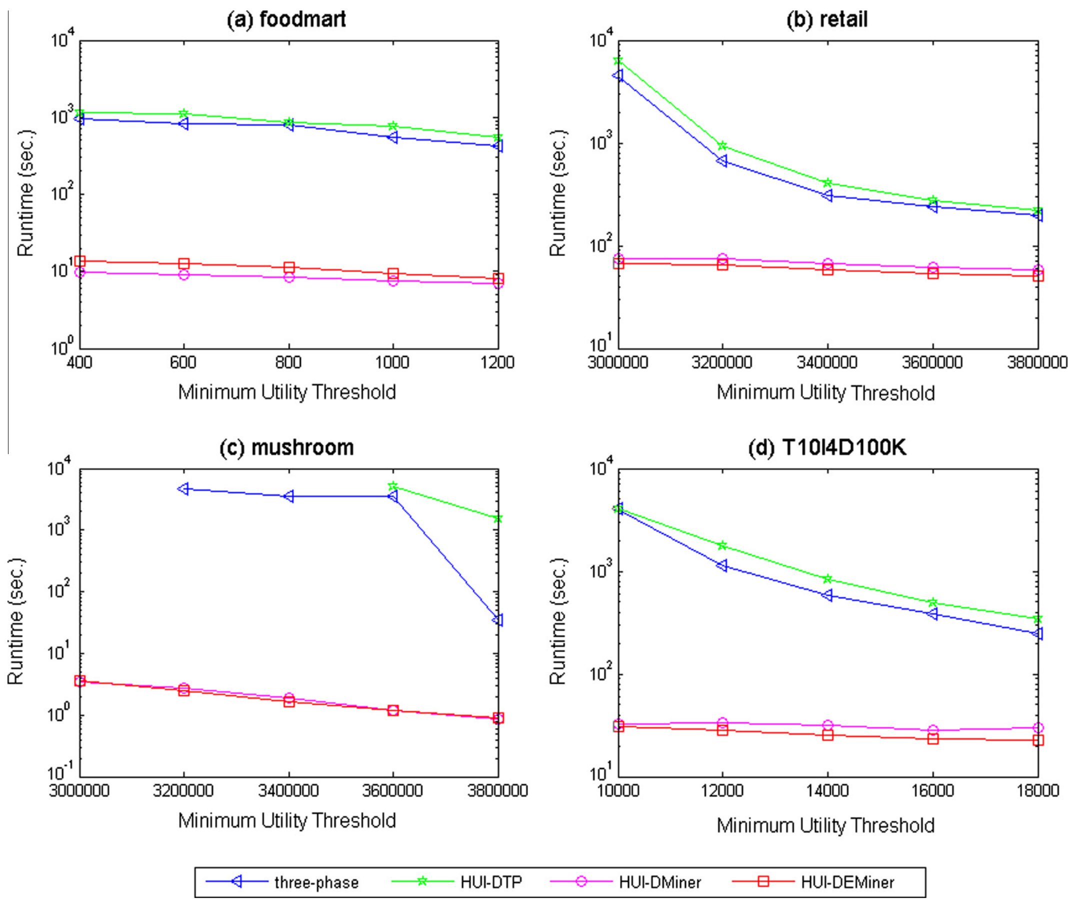


Fig. 4. Runtime under various minimum utility thresholds.

of the enumeration tree traversed by the HUI-DMiner algorithm (denoted as N_2), and the number of nodes traversed by the HUI-DEMiner (denoted as N_3), to compare the effectiveness of the pruning strategies of the three algorithms. Experiments have been conducted under various *MUs* with fixed discount strategies. Results are shown in Fig. 6.

From Fig. 6, it can be observed that $N_1 \geqslant N_2 \geqslant N_3$ for the three datasets except foodmart dataset. It is reasonable since each item in the foodmart dataset does not satisfy the Pareto distribution for the cost price and tag price. In addition, the average transaction length for foodmart is 4. For this dataset, the designed downward closure property of HUI-DTP can efficientlty prune unpromising candidates because the dataset is sparse. The developed HUI-DTP algorithm uses a higher upper-bound transaction-weighted utility measure to ensure that the complete set of HUIs is discovered. The tighter *twu* measure is adopted in the HUI-DMiner and HUI-DEMiner algorithms. This is the reason why $N_1 \geqslant N_2$. Furthermore, the EUCP strategy is adopted in the HUI-DEMiner algorithm to prune unpromising 2-itemsets early, thus avoiding the construction of many PNU-lists in the later stages of the mining process. The number of nodes traversed by the HUI-DEMiner algorithm is much smaller than that of the HUI-DMiner algorithm. Thus, the number of candidates for discovering HUIs is $N_1 \geqslant N_2 \geqslant N_3$, which illustrates the effectiveness of the designed pruning strategies in the three proposed algorithms.

6.4. Memory consumption

The maximum memory consumption of the compared algorithms was measured using the Java API. Several experiments were conducted to compare memory usage under various *MUs* and fixed discount strategies. Results are shown in Fig. 7.

It can be observed that the proposed HUI-DMiner and HUI-DEMiner algorithms require less memory and are more stable when the *MU* threshold is increased compared to the other algorithms. The reason is that the HUI-DMiner and HUI-DEMiner algorithms utilize the PNU-list structure to maintain the necessary information about each itemset and that no time-consuming dataset scans need to be performed. Moreover, the proposed pruning strategies can significantly reduce the number of itemsets that need to be considered. When the *MU* threshold is decreased, the proposed HUI-DMiner and HUI-DEMiner algorithms require less memory than the other algorithms, which can be observed in Fig. 7(d). In Fig. 7, it can also be observed that the proposed HUI-DEMiner algorithm always requires slightly more memory than the HUI-DMiner algorithm under various *MUs*. The reason is that the HUI-DEMiner algorithm requires additional memory usage to keep the constructed *EUCS* of the EUCP strategy. Based on this evaluation of the memory consumption, the designed HUI-DMiner and HUI-DEMiner algorithms can be seen as acceptable for running in limited memory environments and for real-world applications.

6.5. Effect of using different discount strategies

In another experiment, we evaluate the influence of alternative discount strategies on the behavior of the algorithms in terms of number of patterns found. The goal of this experiment is to evaluate if the proposed algorithms remain applicable for alternative discount strategies. The results of mining HUIs from the retail dataset under alternative discount strategies are shown in Fig. 8.

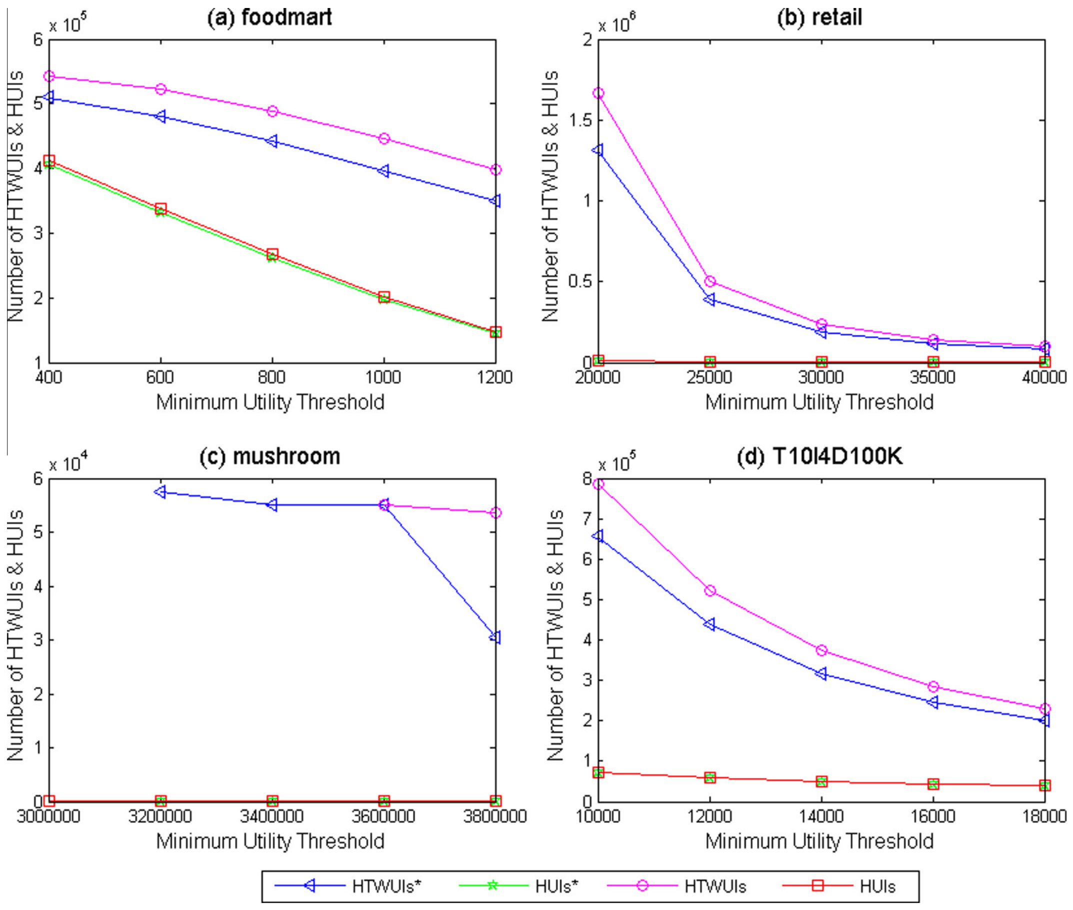


Fig. 5. Number of candidates and HUIs under various minimum utility thresholds.

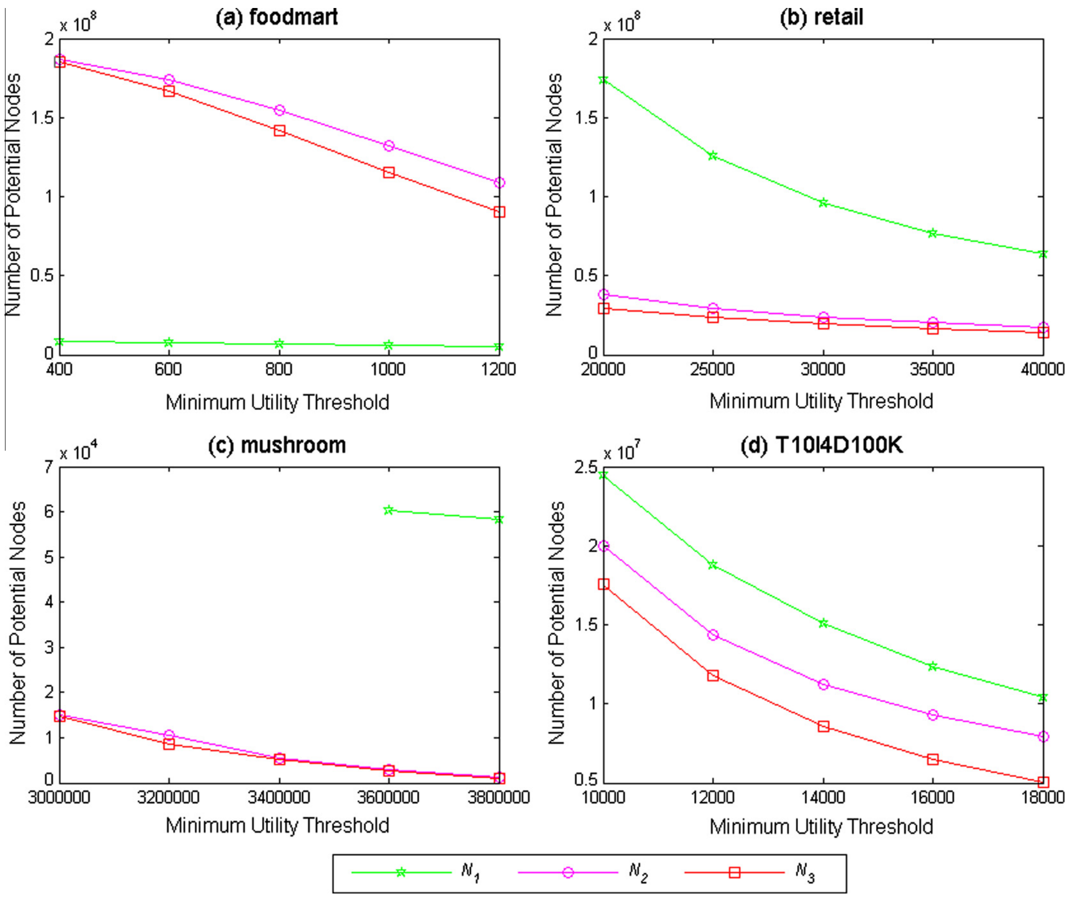


Fig. 6. N_1, N_2 and N_3 under various minimum utility thresholds.

As expected in Fig. 8, it can be observed that the set of HUIs discovered using the proposed algorithms changes depending the discount strategies that are used. In this experiment, our proposed algorithms efficiently find HUIs for the alternative discount strategies. From the above results, it can be concluded that the proposed algorithms can be applied in real-world applications, and the effectiveness and efficiency of the designed algorithms for mining HUIs with discount strategies is good. Moreover, this designed algorithms are useful to analyze the effect of cross-promotion by the derived HUIs under different discount strategies since different discount strategies may affect users' shopping behavior. Hence, the proposed algorithms can help managers or retailers to find the optimal discount strategies and decisions for selling goods. An interesting possibility for future work is to design effective approaches to suggest an appropriate set of discount strategies that would optimize the profit generated by a retail store. This is however a different research problem, that is outside the scope of this paper.

6.6. Scalability

In this section, the scalability of the proposed algorithms is evaluated on the synthetic T10I4N4KD| X| K dataset. For this experiment, the *MU* threshold and the predefined discount strategies are fixed and the number of transactions is varied. Results in terms of runtime, memory consumption and number of patterns are shown in Fig. 9.

From Fig. 9, it can be observed that the proposed HUI-DMiner and HUI-DEMiner algorithms have much better scalability than the three-phase and HUI-DTP algorithms for various dataset sizes in terms of runtime, memory consumption, and number of patterns. From the results of Fig. 9(a) to (c), it can be observed that all compared algorithms spend more time and memory to find HUIs as the size of the dataset increases. The proposed HUI-DEMiner algorithm generally has better performance than HUI-DMiner. An interesting result is that the runtime and memory consumption of the proposed HUI-DMiner and HUI-DEMiner algorithms steadily increase but those of the three-phase and HUI-DTP algorithms sharply increase when the size of the dataset is increased. For example, as dataset size is increased from 100 K to 500 K transactions, both HUI-DMiner and HUI-DEMiner are about two to three orders of magnitude faster than the HUI-DTP algorithm as shown in Fig. 9(a). This is because both HUI-DMiner and HUI-DEMiner algorithms used the PNU-list structure to compress the information to directly mine HUIs without candidate generation while the three-phase and HUI-DTP algorithms rely on the TWU model to mine HUIs based on the generate-candidate-and-test paradigm, thus requiring more computations and producing many unpromising candidates. Moreover, fewer HUIs are produced compared to the number of HTWUIs found by the HUI-DTP algorithm as shown in Fig. 9 (c). Based on the observation made for the scalability experiment, it can be concluded that the three proposed algorithms have excellent scalability and the proposed HUI-DMiner and HUI-DEMiner algorithms have better performance in terms of runtime, memory consumption, number of patterns, and scalability compared to the three-phase algorithm and the proposed HUI-DTP algorithm.

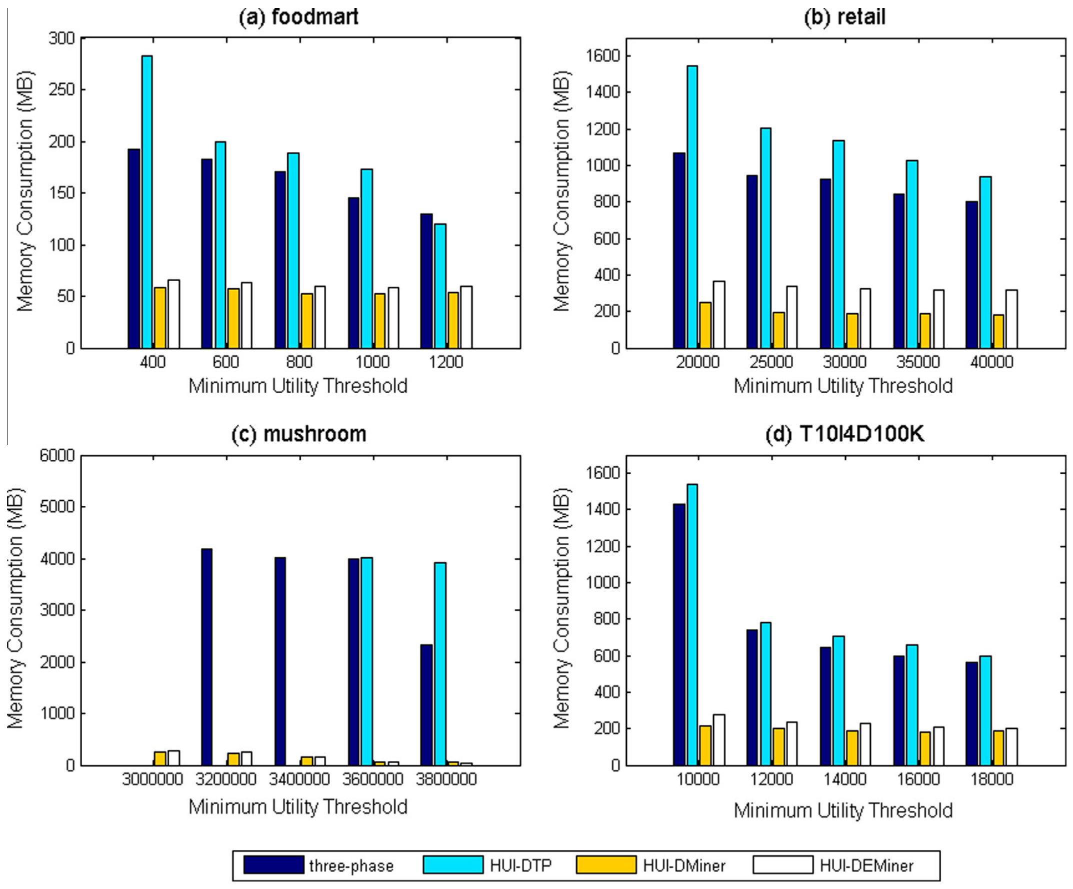


Fig. 7. Memory consumption under various minimum utility thresholds.

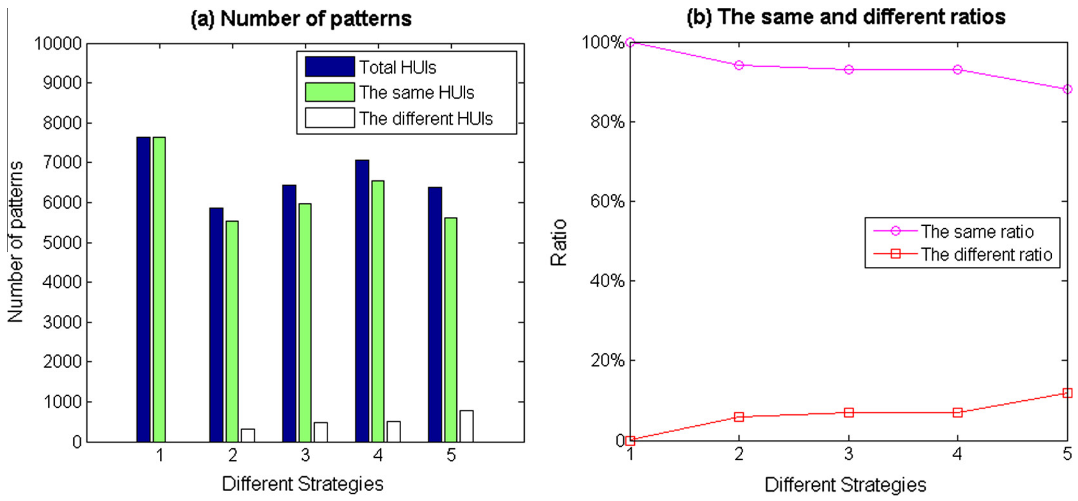


Fig. 8. Number of HUIs when varying the discount strategies.

7. Conclusion and discussion

Traditional HUIM algorithms have been proposed to efficiently mine HUIs from transactional databases. However, most of them consider fixed sale prices for items, which is inadequate for real-life applications. A three-phase algorithm was proposed to discover HUIs when considering items with various discount strategies but it fails to find the complete set of HUIs since it adopts the traditional TWU model. Besides, the three-phase algorithm suffers from the problem of combinatorial explosion since it simply finds HUIs using a level-wise search.

In this paper, several algorithms of mining high-utility itemsets with items having various discount strategies were proposed to find the complete set of HUIs. The HUI-DTP algorithm was proposed as a baseline approach. It performs a level-wise search to mine the complete set of HUIs based on the proposed downward closure property. Furthermore, a new PNU-list structure was proposed to compress the information required to mine HUIs and to

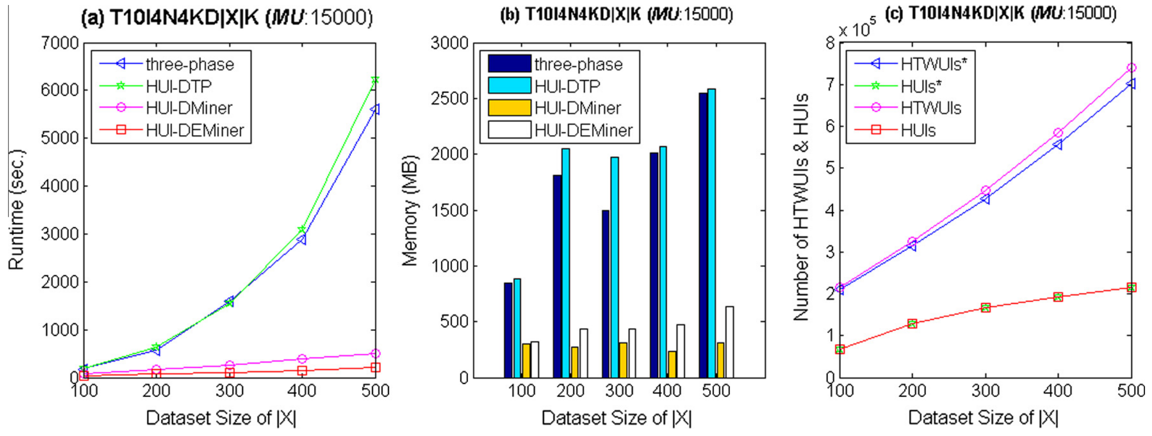


Fig. 9. Scalability of compared algorithms.

facilitate the mining process. The HUI-DMiner algorithm was then proposed to directly mine HUIs based on the designed PNU-list structure and pruning strategies. A HUI-DEMiner algorithm was further designed by adapting the Estimated Utility Co-occurrence Structure (EUCS) to keep the information between 2-itemsets and speed up the mining process. Using a depth-first search, the HUI-DMiner and HUI-DEMiner algorithms can discover HUIs from transactional databases easily and directly, without generating candidates. Substantial experiments have shown that the developed algorithms can discover the complete set of HUIs when items have various discount strategies, and that the two HUI-DMiner and HUI-DEMiner algorithms outperform the three-phase algorithm and the developed HUI-DTP algorithm.

Based on the proposed mining algorithms, the positive cross-promoted and more useful HUIs can be discovered, which can be used to aid managers or can be applied into expert and intelligent systems for making more efficient decisions. With the evaluation of the maximal total profit which bring from derived HUIs, the managers or retailers can find the optimal discount strategies and decisions get the maximal revenue.

Acknowledgement

This research was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61503092, by the Tencent Project under Grant CCF-Tencent RAGR20140114, by the Shenzhen Peacock Project, China, under Grant KQC201109020055A, by the Natural Scientific Research Innovation Foundation in Harbin Institute of Technology under Grant HIT. NSRIF.2014100, and by the Shenzhen Strategic Emerging Industries Program under Grant ZDSY20120613125016389.

References

[1] Frequent itemset mining dataset repository, 2012. <http://fimi.ua.ac.be/data/>.
[2] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large database, in: The ACM SIGMOD International Conference on Management of Data, 1993, pp. 207–216.
[3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: International Conference on Very Large Data Bases, 1994, pp. 487–499.
[4] R. Agrawal, R. Srikant, Quest synthetic data generator, 1994. <http://www.Almaden.ibm.com/cs/quest/syndata.html>.
[5] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong, Y.K. Le, Efficient tree structures for high utility pattern mining in incremental databases, IEEE Trans. Knowl. Data Eng. 21 (12) (2009) 1708–1721.
[6] A.J.C. Trappey, T.A. Chiang, A DEA benchmarking methodology for project planning and management of new product development under decentralized profit-center business model, Adv. Eng. Inform. 22 (4) (2008) 438–444.
[7] R. Chan, Q. Yang, Y.D. Shen, Mining high utility itemsets, in: IEEE International Conference on Data Mining, 2003, in: 19–26.
[8] M.S. Chen, J. Han, P.S. Yu, Data mining: an overview from a database perspective, IEEE Trans. Knowl. Data Eng. 8 (6) (1996) 866–883.
[9] S. Chi, S.J. Suk, Y. Kang, S.P. Mulva, Development of a data mining-based analysis framework for multi-attribute construction project information, Adv. Eng. Inform. 26 (3) (2012) 574–581.
[10] C.J. Chu, V.S. Tseng, T. Liang, An efficient algorithm for mining high utility itemsets with negative item values in large databases, Appl. Math. Comput. 215 (2) (2009) 767–778.
[11] P. Fournier-Viger, FHN: efficient mining of high-utility itemsets with negative unit profits, Adv. Data Mining Appl. (2014) 16–29.
[12] P. Fournier-Viger, C.W. Wu, S. Zida, V.S. Tseng, FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning, Found. Intell. Syst. 8502 (2014) 83–92.
[13] G.C. Lan, T.P. Hong, V.S. Tseng, Discovery of high utility itemsets from on-shelf time periods of products, Expert Syst. Appl. 38 (5) (2011) 5851–5857.
[14] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: a frequent-pattern tree approach, Data Mining Knowl. Discovery 8 (1) (2004) 53–87.
[15] J.C.W. Lin, W. Gan, T.P. Hong, J.S. Pan, Incrementally updating high-utility itemsets with transaction insertion, in: The 10th International Conference Advanced Data Mining and Applications, 2014, pp. 44–56.
[16] J.C.W. Lin, W. Gan, P. Fournier-Viger, T.P. Hong, Mining high-utility itemsets with multiple minimum utility thresholds, in: International C* Conference on Computer Science & Software Engineering, 2015, pp. 9–17.
[17] J.C.W. Lin, W. Gan, T.P. Hong, Vincent S. Tseng, Efficient algorithms for mining up-to-date high-utility patterns, Adv. Eng. Inform. 29 (3) (2015) 648–661.
[18] G.C. Lan, T.P. Hong, J.P. Huang, V.S. Tseng, On-shelf utility mining with negative item values, Expert Syst. Appl. 41 (7) (2014) 3450–3459.
[19] Y. Li, Z. Zhang, W. Chen, F. Min, Mining high utility itemsets with discount strategies, J. Inform. Comput. Sci. 11 (17) (2014) 6297–6307.
[20] C.W. Lin, T.P. Hong, W.H. Lu, An effective tree structure for mining high utility itemsets, Expert Syst. Appl. 38 (6) (2011) 7419–7424.
[21] C.W. Lin, T.P. Hong, G.C. Lan, J.W. Wong, W.Y. Lin, Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, Adv. Eng. Inform. 29 (1) (2015) 16–27.
[22] J. Liu, K. Wang, B.C.M. Fung, Direct discovery of high utility itemsets without candidate generation, 2012, pp. 984–989.
[23] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, in: ACM International Conference on Information and Knowledge Management, 2012, pp. 55–64.
[24] Y. Liu, W.K. Liao, A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, in: Lecture Notes in Computer Science, 2005, pp. 689–695.
[25] V.S. Tseng, C.W. Wu, B.E. Shie, P.S. Yu, UP-growth: an efficient algorithm for high utility itemset mining, in: The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 253–262.
[26] C.W. Wu, B.E. Shie, V.S. Tseng, P.S. Yu, Mining top-*k* high utility itemsets, in: The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 78–86.
[27] H. Yao, H.J. Hamilton, C.J. Butz, A foundational approach to mining itemset utilities from databases, in: The SIAM International Conference on Data Mining, 2004, pp. 211–225.
[28] H. Yao, H.J. Hamilton, Mining itemset utilities from transaction databases, Data Knowl. Eng. 59 (3) (2006) 603–626.
[29] M.J. Zaki, Scalable algorithms for association mining, IEEE Trans. Knowl. Data Eng. 12 (3) (2000) 372–390.
[30] M. Zihayat, A. An, Mining top-*k* high utility patterns over data streams, Inform. Sci. 285 (2014) 138–161.