TopHUI: Top-k high-utility itemset mining with negative utility

Wensheng Gan[1], Shicheng Wan[2], Jiahui Chen[2], Chien-Ming Chen[3], and Lina Qiu[4]*

[1]*Jinan University, Guangdong, China*
[2]*Guangdong University of Technology, Guangdong, China*
[3]*Shandong University of Science and Technology, Shandong, China*
[4]*South China Normal University, Guangzhou, China*

Email: wsgan001@gmail.com, scwan1998@gmail.com, csjhchen@gmail.com,
chienmingchen@ieee.org, lina.qiu@scnu.edu.cn

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378288

***Abstract*—In the field of data science, utility-driven data mining has become an emergent intelligent technique with wide applications. The existing utility mining algorithms usually discover all the patterns satisfying a given minimum utility threshold. However, a huge number of return results is not intuitive, not interpretable, and not easy for users to understand. Besides, it is often difficult and time-consuming for users to set a proper minimum utility threshold that is quite sensitive to the mining results. To address these issues, the problem of top-k high-utility itemset mining has been studied. In this paper, we present an efficient algorithm (named TopHUI) for finding top-k high-utility itemsets from transactional database that contains both positive and negative utility. This algorithm utilizes the positive-and-negative utility-list (PNU-list) to store the compress information, including positive, negative, and remaining utility. Besides, several threshold raising strategies and pruning strategies are proposed to prune the search space. Finally, some extensive experiments were conducted to evaluate the performance of the proposed TopHUI algorithm on both real-life and synthetic datasets, particularly in terms of effectiveness and efficiency.**

***Index Terms*—data science, utility mining, top-k, negative utility, threshold raising strategies.**

I. Introduction

In the past decades, data-driven intelligent technique has shown its powerful capability to extract meaning from data with wide applications. Data intelligence [1] is the interaction and analysis of diverse configurations of data. Since it is essential for humans to discover hidden and useful information from rich data, data mining has become a useful technique in the era of big data. Up to now, many data mining methods have been developed for data-driven analytic form different types of data. One of the most basic techniques in the field of data mining is pattern mining [2], [3], such as frequent pattern mining [2], [3], [4], sequential pattern mining [5], utility-oriented pattern mining [6], [7], and son on.

In recent decades, a multitude of investigators have hastened to improve data mining algorithms due to only finding limited interestingness measures, for example, frequency or support. However, these are insufficient because every object or item is actually unequal in the final analysis. Other preferences, like the profit, cost, risk, and weight [6], [8], have increasingly been studied. And they allow more valuable information to be discovered than the previous support-confidence mining algorithms. In general, measure the interestingness of a pattern is challenging. Utility-driven mining framework, such as high-utility itemset mining (HUIM) algorithm [9], [10], considers the unit utility (also called external utility) and quantity (also called internal utility) of objects or items. Based on these two factors, it is easy to calculate the utility of itemsets. If the derived utility value is greater than a defined minimum utility threshold in advance, then the itemsets can be called high-utility itemsets (HUIs). In the early level-wise algorithms, e.g., the well-known Two-Phase algorithm [11], they first find the upper bound on the utility of itemsets and then trim most of the unqualified itemsets without further calculating their supersets. Other tree-based or list-based utility mining algorithms, such as UP-growth [12], HUI-Miner [13], and CoUPM [14], have been proposed to deal with different mining tasks. Besides, the sequence-based utility mining also attracts extensive attentions, such as ProUM [15], HUSP-ULL [16], and USPT [17]. Up to now, a large number of algorithms of utility mining have been proposed and reviewed in prior work [6], [18].

When performing the pattern mining task in a new database, finding a suitable minimum frequency threshold is not easy for a user. To overcome this usability problem, some top-k mining/ranking algorithms [19], [20] are proposed to return how many patterns user wants w.r.t. a simple parameter k. These algorithms discover the k most frequent patterns in the database. In recent years, top-k pattern mining is a well studied research area. For example, the problem of top-k frequent itemset mining aims at identifying the k highest frequent itemsets. However, many of these models mainly apply the frequency to distinguish the number of occurrences of the pattern, according to a parameter k value. In the field of utility mining [6], the problem of mining top-k high-utility patterns (e.g., itemsets [21], sequences [5]) instead of

mining complete HUIs with a minimum utility threshold has been studied. This is very meaningful and has many real-life application scenarios. Firstly, a huge number of return results of HUIM is not easy for users to understand, not intuitive, and not interpretable. Besides, it is often difficult and time-consuming for users to set a proper minimum utility threshold since it is quite sensitive to the mining results. Up to now, the TKU [22], TKO [23], kHMC [24], THUI [25] and other algorithms have been developed to address this interesting and challenging problem. In general, these top-k mining algorithms can overcome the overwhelming amount of results typically returned by traditional HUIM algorithms.

Overall these methods are not relevant in the domain of utility mining with negative utility values. In most of the existing utility mining algorithms, only the positive utility is used to assess the utility contribution of patterns. However, the negative utility value is also commonly seen in real applications and useful to evaluate the utility contribution. How to design a general utility mining framework for mining top-k HUIs with both positive and negative utility is very necessary. To this end, we design a novel, efficient, and scalable exploration algorithm: TopHUI, a more general and flexible method for mining top-k HUIs with both positive and negative utility. To the best of our knowledge, an algorithm for mining top-k HUIs with both positive utility and negative utility has not yet been proposed, although negative utility is also commonly seen in real-world applications. The main contributions of this paper can be summarized as follows:

- This paper presents a flexible algorithm aiming at mining top-k high utility itemsets from transactional databases that may contain both positive and negative utility values. To the best of our knowledge, this is the first study to address the problem of utility-driven mining of top-k HUIs with both positive and negative utility.
- To reduce the amount of access to the database, a vertical structure named positive-and-negative utility-list (PNU-list) is adopted in TopHUI. Several minimum utility threshold raising strategies, such as RIU, RTU, and RUC, are utilized to quickly raise the minimum utility of the set of potential top-k HUIs.
- Moreover, several pruning strategies are used to reduce the search space efficiently. To calculate the actual utilities of itemsets and their upper-bounds on utility in linear time, TopHUI utilizes several efficient techniques.
- To evaluate the performance of TopHUI, some extensive experiments were conducted on both real-life and synthetic datasets. The experiments show that several threshold raising strategies and pruning strategies can effectively eliminate most of the unpromising itemsets and improve the performance of the TopHUI algorithm in terms of memory consumption and runtime.

The remainder of this paper is organized as follows: Related work consists of two areas is reviewed in Section II. Some basic preliminaries are given in Section III. In Section IV, the properties of negative utility, PNU-list, several pruning strategies, and the complete TopHUI algorithm are introduced in detail. The experimental results that are conducted on both real-life and synthetic datasets are described in Section V. Finally, conclusions are summarized in Section VI.

II. Related Work

The related work consists of two areas, top-k high-utility pattern mining and high-utility pattern mining with negative utility. Details of the current developments and advances are presented below.

A. High-utility pattern mining using top-k setting

Pattern mining is a well-known exploratory data mining technique used to discover interesting patterns in databases. For example, frequent pattern mining (FPM) aims at discovering frequently co-occurring patterns that beyond a minimum support/frequency threshold [2], [3]. However, there are some limitations of the support-confidence-based pattern mining framework [3], [6]. For example, how to judge the "*interesting*" of rule/pattern is difficult based on its support and confidence. Thus, a new data intelligent called utility-driven pattern mining [6] is proposed. There are many Apriori-like, tree-based, and list-based utility mining algorithms, such as Two-Phase [11], HUP-growth [26], UP-growth [12], HUI-Miner [13], FHM [27], and CoUPM [14]. In addition to transactional data, the topic of high-utility sequence mining from sequential data has also been studied with methods like USpan [28], ProUM [15], and HUSP-ULL [16]. Several studies of utility mining have been introduced to improve the mining effectiveness with constraints of various discount strategies [29] and discriminative patterns [30]. Developing effective and efficient algorithms for mining high-utility patterns is an active research area, and more recent studies can be referred to the review literature [6].

Overall these methods are not relevant in the top-k issue. Several variants of the top-k mining problem have also been studied in the recent literature. We briefly review each of these methods in the following. As reviewed in Ref. [31], a number of algorithms and techniques have been developed to efficiently discover top-k high-utility patterns (e.g., itemsets, sequences, episodes). In this paper, we mainly focus on the topic of high-utility itemset mining, thus the itemset-based utility mining algorithms that are related to top-k mining without setting a minimum utility threshold are described and summarized below. In 2012, Wu *et al.* [22] first introduced the top-k HUI mining framework and presented a two-phase TKU algorithm. In the first phase, it constructs an UP-Tree [12] and generates the potential top-k HUIs (PKHUIs). Subsequently, in the second phase, the unpromising PKHUIs are filtered to determine the actual top-k HUIs. Inspired the utility-list [13], the TKO algorithm [23] mines top-k HUIs in one-phase. By utilizing the proposed Raising threshold by Utilities of Candidates (RUC) and Exploring the most Promising Branches (EPB) first strategies, the one-phase TKO method outperforms the two-phase TKU method. By using six different threshold raising strategies (e.g., PUD, RIU, RSD, NU, MC, and SEP),

the REPT algorithm [32] can quickly raise the minimum utility threshold. Both REPT and TKU methods utilize a data structure named UP-Tree for storing utility information. After that, another one-phase algorithm namely kHMC [24] is developed to discover top-k HUIs. Note that kHMC is a utility-list-based approach, and it uses three pruning properties during the growth stage of mining, including utility prune (U-Prune), early abandonment, and transitive extension pruning (TEP) [24]. Recently, a more efficient algorithm namely THUI [25] is proposed. The empirical results show that the THUI algorithm achieves a good scalability and high efficiency than the state-of-the-art algorithms. However, a serious limitation is that only the positive utility but not the negative utility is considered in these algorithms.

B. High utility pattern mining with negative utility

As mentioned before, both positive utility and negative utility are commonly seen in real-world applications. Up to now, although many high-utility pattern mining algorithms have been extensively studied, a serious limitation of these existing studies is that they usually ignore the negative utility values embedding in the transactional data, e.g., market basket data. Note that the pioneer work on utility mining by considering of negative utility values is the HUINIV-Mine which was proposed by Chu *et al.* [33]. It first introduced the new problem of high-utility pattern mining with negative utility values, and proposed the concept of redefined transaction-weighted utilization (*RTWU*). As the first algorithm for mining HUIs with negative utility, the level-wise-based HUINIV-Mine has a serious disadvantage that it needs to generate a large number of candidate patterns and quite time-consuming. After that, the model [34] is designed to deal with data streams. To further reduce the memory consumption and improve the mining efficiency, the positive-and-negative list (PNU-list) based FHN algorithm [35] was proposed by using a depth-first search strategy and the EUCS structure based on the PNU-list to effectively prune the search space without generating candidate sets. Then, the more efficient algorithm is further proposed [36]. Experimental results show that the execution time of FHN is reduced by 500 times compared to HUINIV-Mine, and the memory consumption is reduced by 250 times compared to HUINIV-Mine. Besides, Lan et al. [37] firstly introduced the problem of mining on-shelf HUIs from transactional database which contains the positive and negative utility values, and proposed the TS-HOUN mining algorithm. As the first on-shelf utility mining algorithm with negative utility value, total three-phases are performed in TS-HOUN which is memory intensive and computationally demanding. A survey of high-utility pattern mining with negative utility is recently summarized in Ref. [38]. The k value instead of the minimum utility threshold was chosen to focus on a top-k ranking approach. The utility mining model should be able to discover the top-k patterns and terminate processes. The addressed problem in this paper is different from existing top-k HUIM algorithms due to the flexibility in which both positive and negative instances are considered. Besides, the existing utility mining algorithms cannot be directly applied.

III. Preliminary and Problem Statement

In this section, we first introduce the basic preliminaries and concepts of high-utility itemset mining. More details of these concepts can be referred to Ref. [6], [31]. And then we given the formula problem statement of top-k high-utility itemset mining with consideration of both positive and negative utility values.

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a collection of distinct items; and let $D = \{T_1, T_2, \ldots, T_n\}$ be a quantitative transactional database. Each transaction T_k consists of a set of items and their different quantitative values (also called internal utility, e.g., occurred quantity). The sum of the utility of each transaction is called transaction utility (*tu*). A running example database is shown in Table I, which has ten transactions and five items that are distinct to each other. Table II shows the unit utility (also called external utility, e.g., profit) of each item, which are manually defined. Table I and Table II will be taken as an illustrative example.

TABLE I: Example of an e-commerce database

tid	**Transaction {item: quantity}**	*tu*
T_1	{*a*: 1} {*d*: 2} {*e*: 1}	$27
T_2	{*b*: 10} {*c*: 2} {*d*: 6}	$2
T_3	{*a*: 3} {*d*: 5}	$45
T_4	{*a*: 1} {*e*: 1}	$15
T_5	{*b*: 1} {*c*: 2} {*d*: 6}	$29
T_6	{*b*: 1} {*c*: 1} {*e*: 2}	$15
T_7	{*a*: 2}	$10
T_8	{*a*: 3} {*d*: 1}	$21
T_9	{*b*: 10} {*c*: 1} {*d*: 4}	-$8
T_{10}	{*a*: 1} {*e*: 1}	$15

TABLE II: Unit utility of each item

Item	**Utility ($)**
a	5
b	−3
c	−2
d	6
e	10

Example 1: Table I is considered as the running example and can be described as follows: It has ten transactions $\{T_1, T_2, \ldots, T_{10}\}$ in which transaction T_1 shows that the items $\{a\}$, $\{d\}$, and $\{e\}$ are occurred together in T_1, and their quantities are 1, 2, and 1, respectively. We also assume that unit utilities of the distinct items in Table I are defined in *utable* as: *utable* = {eu(*a*): $5, eu(*b*): -$3, eu(*c*): -$2, eu(*d*): $6, eu(*e*): $10}. Thus, it is obvious to see that (b) and (c) is sold together at loss.

Definition 1 (utility computation): The $u(i, T_c)$ indicates the utility of an item i in the transaction T_c, which can be calculated as: $u(i, T_c) = q(i, T_c) \times eu(i)$. $u(X, T_c)$ shows the utility of an itemset X in a transaction T_c, which can be calculated as: $u(X, T_c) = \sum_{i \in X} u(i, T_c)$. Note that $X \subseteq I$, and the total

utility of X in a database D can be denoted as $u(X)$, which can be calculated as: $u(X) = \sum_{X \subseteq T_c \wedge T_c \in D} u(X, T_c)$.

Example 2: For example in Table I, the utility of $\{a\}$ in T_1 is calculated as: $u(a, T_1)$ = 1 × \$5 = \$5. The utility of $\{a, e\}$ in T_1 is calculated as: $u(\{a, e\}, T_1) = u(a, T_1) + u(e, T_1)$ = 1 × \$5 + 1 × \$10 = \$15. Therefore, the utility of $\{a, e\}$ in Table I can be summed up as: $u(\{a, e\}) = (u(a, T_1) + u(e, T_1)) + (u(a, T_4) + u(e, T_4)) + (u(a, T_{10}) + u(e, T_{10}))$ = (\$5 + \$10) + (\$5 + \$10) + (\$5 + \$10) = \$45. For the $\{b, c, d\}$, the total utility of $\{b, c, d\}$ can be calculated as: $u(\{b, c, d\}) = (u(b, T_2) + u(c, T_2) + u(d, T_2)) + (u(b, T_5) + u(c, T_5) + u(d, T_5)) + (u(b, T_9) + u(c, T_9) + u(d, T_9))$ = ((-\$30) + (-\$4) + \$36) + ((-\$3) + (-\$4) + \$36) + ((-\$30) + (-\$2) + \$24) = \$23.

Definition 2 (high-utility itemset): Given a transactional data-base D and a minimum utility threshold *minutil*, an itemset X is called high-utility itemset (HUI) in D if the $u(X) \geq$ *minutil*. Otherwise, this itemset is said to be a low utility itemset (LUI).

Definition 3 (top-k high-utility itemset): Given a transactional database D and a minimum utility threshold *minutil*, an itemset X is one of the top-k high-utility itemsets (THUIs) in D if there are less than k items which have the utility larger than the utility of itemset X.

Example 3: For example, if the *k* and *minutil* are respectively set as *k* = 8 and *minutil* = \$27, the derived top-8 highest utility itemsets from Table I are {{$\{a, d, e\}$: \$27}; {$\{b, d\}$: \$33}; {$\{a, e\}$: \$45}; {$\{e\}$: \$50}; {$\{a\}$: \$55}; {$\{a, d\}$: \$83}; {$\{c, d\}$: \$86}; {$\{d\}$: \$144}}. Based on the definitions given above, we formulate the problem of top-k HUIM as follows.

Problem Statement. Given a transactional database D (that may have both positive and negative utility values) and the user's desired number of HUIs (aka k), then the problem of top-k high-utility itemset mining (top-k HUIM for short) involves determining exactly the rank k highest utility itemsets in D. Obviously, the problem of top-k HUIM only requires an exact k value as the input parameter, without setting a minimum utility threshold.

It is obvious that the task of mining top-k HUIs depends upon one parameter k, and our addressed problem is more flexible and general than current developed top-k HUIM algorithms. The reason is that different situations of processed database (e.g., only contains positive utility values, only contains negative utility values, or contains both positive and negative utility values) are allowed in our proposed model. When the processed database only contains positive utility values, the mining task equals to the previous studies of top-k utility mining.

IV. Proposed TopHUI Algorithm

In this section, we first introduce several properties of positive and negative utilities in Subsection IV-A, and also illustrate their key benefits. We then introduce a list-based structure called PNU-list in Subsection IV-B. And several strategies for raising the minimum utility threshold are proposed in Subsection IV-C. Subsequently, we describe the proposed TopHUI algorithm in Subsection IV-D. Note that an illustrative example is provided throughout this paper.

A. Positive and negative utilities

Property 1: We first assume that $pu(X)$ and $nu(X)$ are respectively the sum of positive utility and negative utility of an itemset X in a database. Then we can obtain that the utility of *X* is calculated as: $u(X) = pu(X) + nu(X)$, where $nu(X) \leq u(X) \leq pu(X)$ holds.

From the above-stated property, we can conclude that $u(X)$ and $nu(X)$ cannot be straightforwardly used as the over-estimated utility of a pattern (*X*). Moreover, even $pu(X)$ is the upper-bound utility value of *X*, the downward closure property for the superset of *X* cannot be held since both the positive and negative utilities of the items are considered in the addressed problem. We then re-utilize the traditional *TWU* property [11] to establish a new over-estimated value of the discovered pattern.

Definition 4 (redefined transaction-weighted utility, RTWU): In HUI mining, the transaction utility (*TU*) is defined as: $TU(T_c) = \sum_{i \in T_c} u(i, T_c)$. By considering both positive and negative unit utility of items, the transaction utility can be re-defined as: $RTU(T_c) = \sum_{i \in T_c \wedge eu(i) > 0} u(i, T_c)$ [33], [35]. The transaction weighted utilization of an itemset *X* is then redefined as *RTWU*: $RTWU(X) = \sum_{X \subseteq T_c \wedge T_c \in D} RTU(T_c)$ [33], [35]. The above-stated definitions can be used to hold the *downward closure* property for mining the required HUIs that may contain negative utility values. Note that $RTWU(X) \geq u(X)$.

Example 4: Consider the second transaction, the $RTU(T_2)$ is \$36. And the ten transactions in Table I have their re-defined transaction utilities as: $RTU(T_1)$ = \$27, $RTU(T_2)$ = \$36, $RTU(T_3)$ = \$45, $RTU(T_4)$ = \$15, $RTU(T_5)$ = \$36, $RTU(T_6)$ = \$20, $RTU(T_7)$ = \$10, $RTU(T_8)$ = \$21, $RTU(T_9)$ = \$24, and $RTU(T_{10})$ = \$15. Clearly, in some transactions, RTU is not equal to the *tu* value. Consider two patterns $\{a, e\}$ and $\{b, c, d\}$, the $RTWU(\{a, e\}) = RTU(T_1) + RTU(T_4) + RTU(T_{10})$ = \$57, and $RTWU(\{b, c, d\}) = RTU(T_2) + RTU(T_5) + RTU(T_9)$ = \$96. Thus, both of them are the over-estimated values of $u(\{a, e\})$ = \$45 and $u(\{b, c, d\})$ = \$23.

B. Search space and PNU-List structure

The search space of the top-k HUI mining problem can be represented as a set-enumeration tree [13]. The all potential itemsets can be explored using depth-first-search in the set-enumeration tree starting from the root which is an empty set. We sort the items in an increasing order of *RTWU* values that may reduce the search space [11], [13]. Some definitions and concepts related to exploration of itemsets in the TopHUI algorithm with this conceptual tree are given below.

Definition 5 (Extension of an itemset): An itemset α can be extended into itemset Y (Y appears in the redefined subtree of the set-enumeration tree), if $Y = \alpha \cup \{X\}$, where X/α, X cannot be empty. In addition, the itemset α extends a single itemset Y (Y is a child of α in the set-enumeration tree).

Definition 6 (processing order): Supposing that the items in the database are arranged in a certain order, such as alphabetical order or *RTWU*-ascending order, then there is no harm in reordering the items in the database in support of ascending order and expressing this order with the symbol $\prec$.

In the FHN algorithm [35], a new vertical data structure named PNU-list is used to store necessary information, such as the positive utility, the negative utility, and the remaining positive utility. Note that the processing order of the items in the database is defined as $\succ$, which holds the properties as: (1) the items are sorted as the *RTWU*-ascending order; and (2) negative items always succeed positive ones. The PNU-list structure used in the TopHUI algorithm is stated as follows:

Definition 7 (PNU-list [35]): Let X be an itemset in the database. The PNU-list of X is denoted as *X.PNUL*, and it consists of four elements: (1) *tid* represents the transaction ID in the database; (2) pu shows the positive utility of X in T_{tid}, and $u(X, T_{tid}) \geq 0$; (3) nu represents the negative utility of X in T_{tid}, and $u(X, T_{tid}) < 0$; (4) rpu represents $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$ and $u(i, T_{tid}) \geq 0$, which keeps only the positive utility values for the remaining items.

(c)

tid	*pu*	*nu*	*rpu*
T_2	0	-$4	0
T_5	0	-$4	0
T_6	0	-$2	0
T_9	0	-$2	0

(d)

tid	*pu*	*nu*	*rpu*
T_1	$12	0	0
T_2	$36	0	0
T_3	$30	0	0
T_5	$36	0	0
T_8	$6	0	0
T_9	$24	0	0

(a)

tid	*pu*	*nu*	*rpu*
T_1	$5	0	$12
T_3	$15	0	$30
T_4	$5	0	0
T_7	$10	0	0
T_8	$15	0	$6
T_{10}	$10	0	0

(e)

tid	*pu*	*nu*	*rpu*
T_1	$10	0	$17
T_4	$10	0	$5
T_6	$20	0	0
T_{10}	$10	0	$5

(b)

tid	*pu*	*nu*	*rpu*
T_2	0	-$30	0
T_5	0	-$3	0
T_6	0	-$3	0
T_9	0	-$30	0

Fig. 1: PNU-lists of five items

Example 5: The search space of the developed TopHUI approach can be shown as the utility-based set-enumeration tree [30], called a PNU-tree, based on the developed PNU-list. Since $\{RTWU(a)$: \$133; $RTWU(b)$: \$116; $RTWU(c)$: \$116; $RTWU(d)$: \$189; and $RTWU(e)$: \$77$\}$, the designed processing order $\succ$ of the running example can be represented as: $\{e \succ a \succ d \succ b \succ c\}$. Thus, the constructed PNU-list for all items in Table I is shown in Fig. 1. We have $\{a\}$*.PNUL* = $\{(T_1$, \$5, \$0, \$12), $(T_3$, \$15, \$0, \$30), $(T_4$, \$5, \$0, \$0), $(T_7$, \$10, \$0, \$0), $(T_8$, \$15, \$0, \$6), $(T_{10}$, \$5, \$0, \$0)$\}$; $\{e\}$*.PNUL* = $\{(T_1$, \$10, \$0, \$17), $(T_4$, \$10, \$0, \$5), $(T_6$, \$20, \$0, \$0), $(T_{10}$, \$10, \$0, \$5)$\}$; $\{a, e\}$*.PNUL* = $\{(T_1$, \$15, \$0, \$12), $(T_4$, \$15, \$0, \$0), $(T_{10}$, \$15, \$0, \$0)$\}$.

As shown in the adopted PNU-list, it is easy to obtain the total utility, the total positive utility, the total negative utility, and the remaining utility of a target itemset in the entire processed database. For easier calculation, the information that is extracted from PNU-list is defined below.

Definition 8: The sums of the total utility, *pu* values, *nu* values, and *rpu* values in the PNU-list of an itemset X are respectively denoted as: *SUM(X.u)*, *SUM(X.pu)*, *SUM(X.nu)*, and *SUM(X.rpu)*, which can be respectively defined as: $SUM(X.pu) = \sum_{X \in T_c \wedge T_c \subseteq D} X.pu(T_c)$; $SUM(X.nu) = \sum_{X \in T_c \wedge T_c \subseteq D} X.nu(T_c)$; $SUM(X.rpu) = \sum_{X \in T_c \wedge T_c \subseteq D} X.rpu(T_c)$; and $SUM(X.u) = SUM(X.pu) + SUM(X.nu)$.

Therefore, if we set itemset $X = \{a\}$ that $SUM(X.pu)$ = \$60, $SUM(X.rpu)$ = \$48, $SUM(X.nu)$ = \$0, and $SUM(X.u)$ = \$0 + \$60 = \$60. For a detailed analysis of the exact strategy, the reader can refer to Ref. [35].

C. Minimum utility threshold raising strategies

One of the main challenging problems in top-k HUIM is that only the k value instead of the minimum utility threshold (*minutil*) is provided in advance. Besides, the minimum utility threshold when the algorithm runs starts from 0 or other value. Therefore, how to automatically increase the minimum utility threshold during the mining process, without losing any true top-k HUIs, is a key challenge. To solve this problem, we used several effective raising strategies.

Raising strategy 1. *RIU: Raising threshold based on real item utilities [32].* If there are at least k items and the k-th highest *RIU* value (i.e., RIU_k) is greater than *minutil*, then *minutil* value can be raised to RIU_k.

Raising strategy 2: *RTU: Raising threshold based on transaction utilities.* If there are at least k transactions and the k-th highest *TU* value (i.e., the top-3 TU values in Table I are \$45, \$29, and \$27) is greater than *minutil*, then *minutil* value can be raised to the k-th TU value.

The REPT algorithm [32] applies the RIU strategy to further increase the value of *minutil*. The strategy involves increasing the value of *minutil* based on some real utilities of distinct items. Firstly, we sort a set of *RIU* values for all items, and then set *minutil* to the k-th largest *RIU* value. If the number of 1-items is less then the k parameter, *minutil* is set to zero. After that, we use the *RTU* strategy to raise *minutil* in a similar way. For the sample database with k = 8, the eighth highest TU = \$10 which is larger than the current *minutil* value of 0 (obtained after applying the RIU strategy). Hence, the *minutil* value will be raised by applying the RTU strategy for the running example.

Raising strategy 3: *RTWU: Raising threshold based on RTWU*. Next, we continue to introduce the basic content of *RTWU* strategy in detail. First, scan the database once, and calculate the *RTWU*(x) of all items in this process. Let *RTWU* = $\{RTWU_1, RTWU_2, \ldots, RTWU_n\}$ be the set of *RTWU* of items in I. Then, we sort the *RTWU* list. Then, we use *RTWU* strategy to increase *minutil* in this set according to the value of the k-th *RTWU*. Then, an updated *minutil* is just found. For example, suppose the user sets k to 8, and the eighth largest value in the *RTWU* set is set to *minutil*. Finally, the new *minutil* is increased by the TopHUI algorithm.

Raising strategy 4: *RUC: Raising the threshold by the Utilities of Candidates [23]*. If there are at least k high-utility itemsets in PNU-list structure and the k-th highest utility value of an itemset is greater than *minutil*, then the *minutil* value can also be raised to this k-th highest utility value.

RUC strategy is similar, in principle, to the strategies used in previous studies [22], [32]. The candidate top-k HUIs are maintained in a priority queue structure. The entries in the queue are updated when a new candidate with higher utility value is observed. The RUC strategy helps in raising the threshold value and improving the performance of top-k HUI mining. These raising strategies are incorporated as part of the *Raise-Threshold* function (refer to step 10 of Algorithm 2). This function updates the queue dynamically, and revises the *minutil* value by applying the above strategies.

D. Proposed algorithm with pruning strategies

This section describes the proposed algorithm with several effective pruning strategies in detail. The complete pseudo-code of the TopHUI algorithm can be referred to Algorithm 1. To summary, the TopHUI algorithm makes use of the above *minutil* raising strategies and the following pruning properties: *RTWU*-prune [11], RU-prune [13], [35], EUCS-prune [27], and LA-prune [35], [39]. These pruning properties are applied at different stages: the *RTWU*-Prune is applied during early stages to remove unpromising 1-itemsets; the RU-Prune and EUCS-prune strategies are applied during depth-first search and exploration in the search space; and the LA-Prune is applied as part of the PNU-list construction. The adopted strategies are presented below.

Strategy 1 (RTWU-prune): When depth-first traversing the search space of the designed TopHUI algorithm, if the *RTWU* of a node X is less than *minutil*, then this node and its descendants can be directly pruned.

Proof: This strategy is based on the concept of *RTWU*, and the property can be extracted as $RTWU(X_k) \leq RTWU(X_{k-1})$. There is no doubt that if $RTWU(X_{k-1}) <$ *minutil*, then $RTWU(X_k) <$ *minutil* and X_k can be directly pruned. They will not be the top-k HUIs. ■

Strategy 2 (RU-prune): In a set of PNU-lists constructed in a consistent $\prec$ order, if the upper-bound on utility of their offspring ($SUM(X.pu) + SUM(X.rpu)$) which is calculated based on a node X is less than *minutil*, then all the nodes rooted at X as descendant nodes can be quickly pruned [13], [35].

Algorithm 1 TopHUI(D, *utable*, k)

Input: D: a transactional database, $utable$: the utility-table with positive and negative unit utilities, k.
Output: *THUIs*: a set of top-k HUIs.

1: scan D to calculate the tu value of each transaction, and *RTWU* of each item $i \in I$;
2: apply *RIU* and *RTU* raising strategies to raise *minutil*;
3: **for** each $T_c \in D$ **do**
4: obtain new T'_c by removing the items $RTWU(i) <$ *minutil*;
5: sort new T'_c as per the ordering heuristic ($\prec$);
6: **for** each $i \in T'_c$ in order $\prec$ **do**
7: call ***UpdateLIU***(i, T'_c) [25];
8: construct/update PNU-list for each i in T'_c;
9: call ***Raise-Threshold*** by applying several threshold raising strategies;
10: call ***THUI-Search***($\emptyset$, *PNULs*, *minutil*);
11: **return** *THUIs*

Algorithm 2 Construction(X, X_a, X_b)

Input: X, an itemset with its corresponding PNU-list and *PNU-table*; X_a, the extension of X with an item a; X_b, the extension of X with an item b.
Output: X_{ab}.

1: initialize *sumUti* = 0, $X_{ab}.UL \leftarrow \emptyset$;
2: **for** each tuple $E_a \in X_a.UL$ **do**
3: **if** $\exists E_a \in X_b.UL \wedge E_a.tid == E_b.tid$ **then**
4: **if** $X.UL \neq \emptyset$ **then**
5: search for $E \in X.UL$, $E.tid$ = $E_a.tid$;
6: $E_{ab} \leftarrow <E_a.tid,\ E_a.pu + E_b.pu - E.pu,\ E_a.nu + E_b.nu - E.nu,\ E_b.rpu>$;
7: **else**
8: $E_{ab} \leftarrow <E_a.tid,\ E_a.pu + E_b.pu,\ E_a.nu + E_b.nu,\ E_b.rpu>$;
9: $X_{ab}.pu$ += $E_{ab}.pu$;
10: $X_{ab}.nu$ += $E_{ab}.nu$;
11: $X_{ab}.rpu$ += $E_{ab}.rpu$;
12: $X_{ab}.UL \leftarrow X_{ab}.UL \cup E_{ab}$;
13: **else**
14: *sumUti* -= ($X_{ab}.pu$ + $X_{ab}.nu$);
15: **if** *sumUti* $<$ *minutil* **then**
16: **return** *null*;
17: **return** X_{ab}

Strategy 3 (LA-prune): In the designed TopHUI, if the sum of overall utility and remaining utility of a pattern X is less than the minimum utility value, then this pattern is not a top-k high-utility pattern. Thus, this itemset X and all nodes with it as the root in the search space will be pruned [35], [39].

As shown in Algorithm 2, when the PNU-lists of the 1-itemsets have been constructed, there is no need to follow these processes to build them for k-itemsets ($k > 1$) by rescanning the database. Instead of traversing the database

multiple times, the following construction based on the PNU-list of the 1-itemsets is used, since the required information is already contained in the built PNU-list. Algorithm 2 shows how to construct the PNU-list of k-itemsets ($k \geq 2$) based on the PNU-lists of 1-itemsets, and more details can be found in [35]. At the beginning, an itemset X and two extensions of it, namely, X_a and X_b, are given, and the order of a precedes b. A new itemset X_{ab} can be obtained by combining these two extensions. The algorithm involves LA-prune, which is explained in Lines 14-16.

Strategy 4 (Leaf-prune): This strategy was used in the THUI Algorithm [25] to raise the *minutil* value. If the utility of leaf node that is less than or equal to the minimum utility threshold, then it is not necessary to measure its extended itemsets.

Feasible strategies for trimming the search space and reducing the runtime have been proposed above. The core processes of the TopHUI algorithm are explained according to these proposed strategies and shown in Algorithm 1. Due to the space limit, the detailed pseudo-code of the *THUI-Search* procedure is skipped here. Note that the *THUI-Search* utilizes the PNU-list structure to explore the search space w.r.t. a set-enumeration tree. It is similar to the operations in the THUI [25] and FHN [35] algorithms.

V. Experimental Evaluation

In this section, we describe the experiments that were conducted on several real datasets. Note that the evaluations were compared by the proposed TopHUI algorithm and the sate-of-the-art THUI [25] algorithm. As mentioned before, TopHUI is the first algorithm to address the problem of top-k utility mining with or without negative utility. All existing top-k utility mining algorithms (e.g., kHMC [24], TKO [23], and THUI [25]) can not solve this task successfully since they lack the flexibility to deal with negative utility. Therefore, all these top-k HUIM algorithms are not suitable for the comparison to evaluate the effectiveness and efficiency of the proposed TopHUI algorithm. In order to access the efficiency of TopHUI, two variants adopting different pruning strategies are further compared. Note that TopHUI$_{base}$ only utilizes the threshold raising strategies, while TopHUI adopts both the threshold raising strategies and pruning strategies.

A. Experimental setup

We implemented all the compared algorithms in Java language. All the experiments were performed on a Dell workstation with a Intel Xeon 3.7GHz processor, 8GB Java heap size, and running a Linux OS with 64GB of RAM.

Totally four datasets were conducted in the experiments, and they can be downloaded from the open-source data mining library[1]. The details of the characteristics of the used datasets (retail, chess, mushroom, and T40I1D100K) are shown in Table III. Note that the external utilities of items in each dataset were generated between 0.01 and 1000 using a log-normal distribution. The internal utilities of items were randomly generated in the range of 1 to 10. This is the same as past experimental studies in the literature [6], [11], [13], [40]. In each dataset, the parameter k value is set to 100, 500, 1000, 2000, 3000, 5000, 7000, 8000, and 10000, respectively.

[1] http://www.philippe-fournier-viger.com/spmf/index.php

TABLE III: Features of the datasets

Dataset	$\|D\|$	$\|I\|$	Type
Retail	88,162	16,470	sparse
Chess	2,085	591	dense
Mushroom	8,124	120	dense
T40I1D100K	100,000	1000	sparse

B. Influence of the negative values

To access the effectiveness and the influence of the negative utility values, the final least utility value of two kinds of final derived patterns, top-k@HUIs* (discovered by THUI which does not deal with negative utility values) and top-k@HUIs (discovered by TopHUI), were compared. By varying the k threshold in each dataset, we recorded the execution time, the number of candidates, and the final top-k patterns. Finally, we calculated the final least utility value, and the detailed results are as shown in Table IV and Fig. 3, respectively. Note that the *least utility** is related to the set of top-k@HUIs*, and *least utility* is related to the set of top-k@HUIs. When the experiment exceeds 10,000 seconds or runs out-of-memory, we marked the results as "-".

According to the *least utility** in Table IV, we can known that the two kinds of top-k patterns are different. This is reasonable, because THUI only measures positive utility values, while TopHUI has more flexibility to deal with the complicate datasets which may contain both positive utility values and negative ones. In other words, TopHUI can process not only the datasets that tested in those top-k HUIM algorithms, but also the datasets having negative utility. In fact, the traditional top-k HUIM task can be regarded as a special case of flexible task of top-k HUIM with or without negative utility. It is clear that both THUI and TopHUI could successfully discover the same number of target top-k patterns. However, the final least utilities of the top-k@HUIs are not the same in many cases. For example, in retail dataset, when k is set to 1000, the 1000-th highest utility value in THUI is 5121, while that of TopHUI is 5594. In the dense mushroom dataset, the least utility values of two compared algorithms are the same. However, in another dense dataset, the results are not the same, as shown in chess. Therefore, the TopHUI algorithm can successfully address the top-k HUI mining task, while the state-of-the-art THUI algorithm cannot achieve this goal when dealing with negative utility values.

C. Candidate analysis

To access the efficiency, the candidates generated by two variants of the designed TopHUI algorithm are first evaluated below. For having a fair comparison, the same top-k value is assumed to each test, and then the total number of intermediate candidates are recorded. The results under various parameter

TABLE IV: Final results under various k values

Dataset	Result	Results under various parameter settings								
		$test_1$	$test_2$	$test_3$	$test_4$	$test_5$	$test_6$	$test_7$	$test_8$	$test_9$
(a) retail	**top@k**	100	500	1000	2000	3000	5000	7000	8000	10000
	least utility*	22638	8205	5121	3194	2429	1735	1380	1263	1092
	least utility	25191	8671	5594	3575	2762	2019	1634	-	-
(b) chess	**top@k**	100	500	1000	2000	3000	5000	7000	8000	10000
	least utility*	55176	8583	71299	208621	192175	171818	159814	151700	140465
	least utility	136944	124100	118279	112375	108704	-	-	-	-
(c) mushroom	**top@k**	100	500	1000	2000	3000	5000	7000	8000	10000
	least utility*	312237	268190	248771	225384	211026	193280	181638	178244	155043
	least utility	312237	268190	248771	225384	211026	193280	181638	178244	-
(d) T10I4D100K	**top@k**	100	500	1000	2000	3000	5000	7000	8000	10000
	least utility*	33123	17058	12548	8248	6005	3896	2903	2523	1990
	least utility	34284	18408	13855	9912	7904	5669	4462	4013	3364

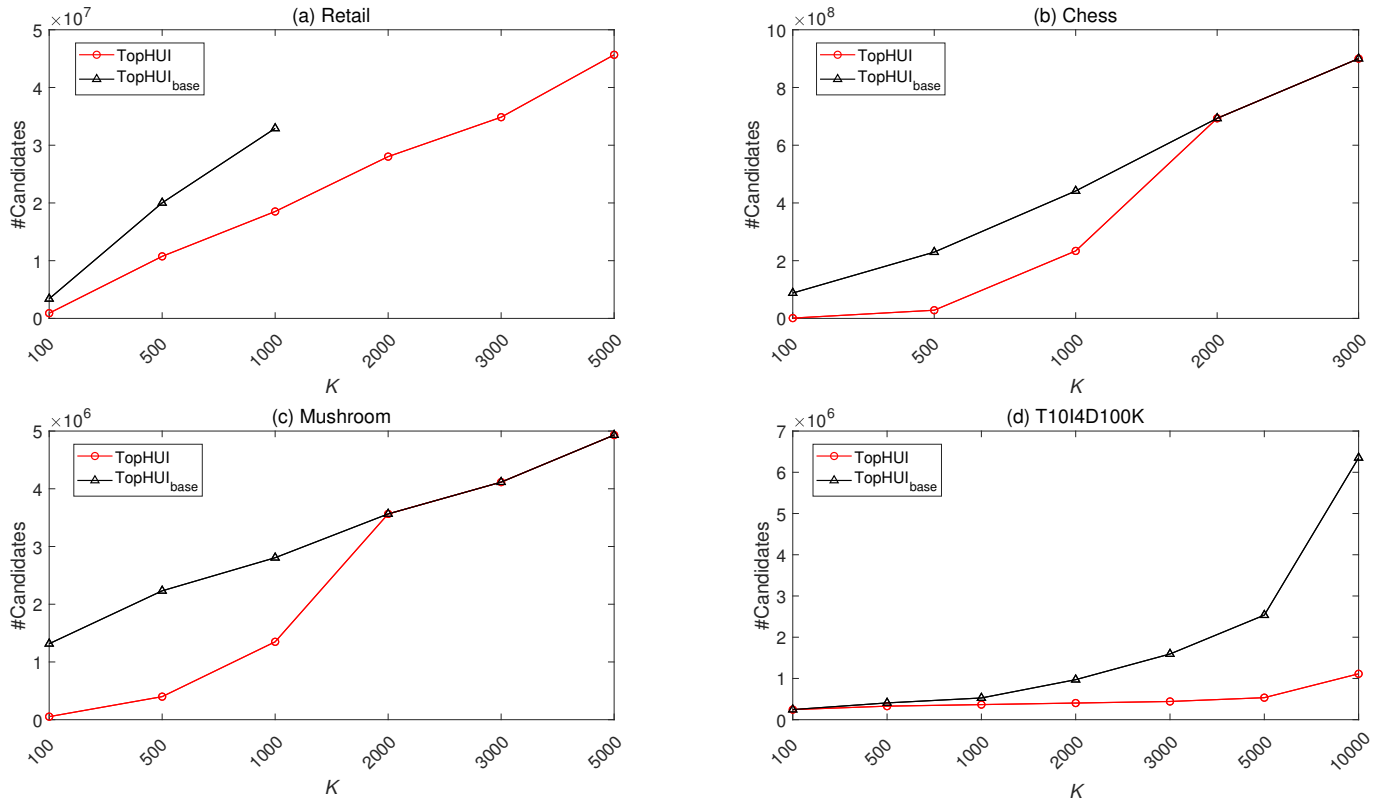

Fig. 2: Candidates under changed top-k value

settings w.r.t. the k value are presented in Fig. 2, from (a) to (d), respectively. In all tested datasets, the number of candidates of TopHUI$_{base}$ is quite more than that of TopHUI. As an enhanced variant, TopHUI adopts both the threshold raising strategies and pruning strategies, while TopHUI$_{base}$ only utilizes the threshold raising strategies. Therefore, the adopted pruning strategies in our designed algorithm make a positive impact to prune the search space and reduce the unpromising patterns. Surprisingly, in two dense datasets, as shown in chess and mushroom, the number of candidates of the baseline is more closer to that of the improved version, when the k value is set larger. For example, when k = 2000, they are the same in mushroom and chess. In summary, we can conclude that the adopted pruning strategies can limit the scope of scanning and reduce the number of candidates.

D. Runtime analysis

In this subsection, the execution time of TopHUI$_{base}$ and TopHUI is further compared. Notice that the TopHUI algorithm involves in only one parameter, and this top-k value in experiments may result in different efficiency performance. In some cases, the top-k value can also be regard as the optimal threshold in mining task for achieving best performance. In general, it is not easily to find a suitable k threshold for the task of top-k HUIM. It should be noted that the k parameter in top-k mining task depends on users' requirement.

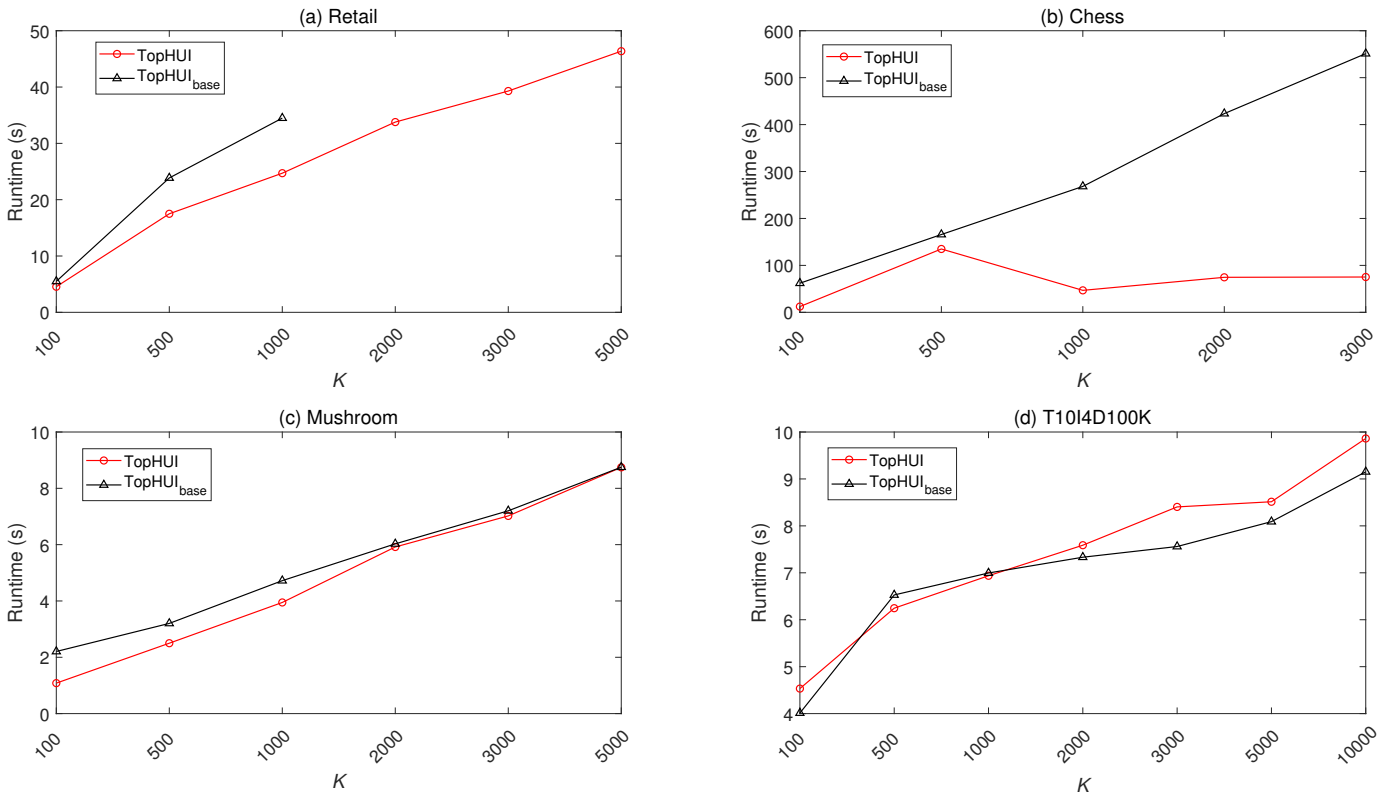

Fig. 3: Runtime under changed top-k value

In this set of experiments, we examined the performance of each algorithm on both sparse and dense datasets. The results of our experiments are presented in Fig. 3. The results of runtime performance for the proposed TopHUI algorithm were found to be significantly better than the TopHUI$_{base}$ algorithm. On chess, retail, and mushroom datasets, TopHUI showed remarkable performance compared to the version which does not adopt the pruning strategies. Surprisingly, on T10I4D100K, our designed TopHUI performed worse than TopHUI$_{base}$. One possible reason is the use of threshold raising strategies makes contribution to quickly achieve a high minimum utility threshold.

To summary, TopHUI performs well than the baseline version in many cases. The reason is that TopHUI can prune and reduce the search space during the mining process. Using the proposed pruning strategies with upper-bound on utility, TopHUI can successfully discover the top-k HUIs from transactional database with or without negative utility values. While in another side, the THUI algorithm cannot be used to address this mining task although sometimes its least utility is similar to TopHUI. The reason is that the adopted techniques in THUI can reduce the account of unpromising patterns, and then reduce the cost of database scanning, but it can not handle the negative utility values. Therefore, the designed TopHUI algorithm is more flexible than the existing top-k algorithms.

VI. Conclusions

So far, several algorithms have been developed to solve the problem of mining top-k high utility itemsets with only positive utility values. In addition to positive utility, however, negative utility value is also commonly seen in real-world databases. To solve this problem, a novel algorithm namely TopHUI is proposed in this paper for efficiently mining top-k high-utility itemsets with both positive and negative utility values. As a flexible algorithm, TopHUI adopts a vertical structure called PNU-list to store the necessary information from databases. By utilizing some properties of positive and negative utility, PNU-list can help to avoid scanning database many times. Several pruning strategies with upper bound on utility are utilized to improve the efficiency of searching and reduce the running time. The experimental results showed that the TopHUI algorithm has a good performance on effectiveness and efficiency since it not only discovers the complete set of top-k HUIs with consideration of negative utility values, but also outperforms the state-of-the-art algorithms.

Acknowledgment

This research was partially supported by the Key Areas Research and Development Program of Guangdong Province (Grant No. 2019B010139002), National Natural Science Foundation of China (Grant No. 61902079 and Grant No. 62002136).

REFERENCES

[1] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 1–12, 2000.

[2] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487–499.

[3] J. Pei, J. Han, and L. V. Lakshmanan, "Mining frequent itemsets with convertible constraints," in *Proceedings of 17th International Conference on Data Engineering*, 2001, pp. 433–442.

[4] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and J. Zhan, "Mining of frequent patterns with multiple minimum supports," *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 83–96, 2017.

[5] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and P. S. Yu, "A survey of parallel sequential pattern mining," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 3, pp. 1–34, 2019.

[6] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, V. S. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *IEEE Transactions on Knowledge and Data Engineering. DOI: 10.1109/TKDE.2019.2942594*, 2019.

[7] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and P. S. Yu, "HUOPM: High utility occupancy pattern mining," *IEEE Transactions on Cybernetics*, vol. 50, no. 3, pp. 1195–1208, 2020.

[8] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, "Weighted frequent itemset mining over uncertain databases," *Applied Intelligence*, vol. 44, no. 1, pp. 232–250, 2016.

[9] R. Chan, Q. Yang, and Y. D. Shen, "Mining high utility itemsets," in *Proceedings of the 3rd IEEE International Conference On Data Mining*. IEEE, 2003, pp. 19–26.

[10] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data and Knowledge Engineering*, vol. 59, no. 3, pp. 603–626, 2006.

[11] Y. Liu, W. K. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2005, pp. 689–695.

[12] V. S. Tseng, C. W. Wu, B. E. Shie, and P. S. Yu, "UP-Growth: an efficient algorithm for high utility itemset mining," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2010, pp. 253–262.

[13] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. ACM, 2012, pp. 55–64.

[14] W. Gan, J. C. W. Lin, H. C. Chao, T. P. Hong, and P. S. Yu, "CoUPM: Correlated utility-based pattern mining," in *Proceeding of the IEEE International Conference on Big Data*. IEEE, 2018, pp. 2607–2616.

[15] W. Gan, J. C. W. Lin, J. Zhang, H. C. Chao, H. Fujita, and P. S. Yu, "ProUM: High utility sequential pattern mining," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2019, pp. 767–773.

[16] W. Gan, J. C. W. Lin, J. Zhang, P. Fournier-Viger, H. C. Chao, and P. S. Yu, "Fast utility mining on sequence data," *IEEE Transactions on Cybernetics. DOI: 10.1109/TCYB.2020.2970176*, 2020.

[17] W. Gan, J. C. W. Lin, H. C. Chao, A. V. Vasilakos, and S. Y. Philip, "Utility-driven data analytics on uncertain data," *IEEE Systems Journal*, 2020.

[18] W. Gan, J. C. W. Lin, H. C. Chao, S. L. Wang, and P. S. Yu, "Privacy preserving utility mining: a survey," in *IEEE International Conference on Big Data*. IEEE, 2018, pp. 2617–2626.

[19] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, "Mining top-k frequent closed patterns without minimum support," in *Proceedings of IEEE International Conference on Data Mining*. IEEE, 2002, pp. 211–218.

[20] R. C. W. Wong and A. W. C. Fu, "Mining top-k frequent itemsets from data streams," *Data Mining and Knowledge Discovery*, vol. 13, no. 2, pp. 193–217, 2006.

[21] C. M. Chen, L. Chen, W. Gan, L. Qiu, and W. Ding, "Discovering high utility-occupancy patterns from uncertain data," *Information Sciences*, vol. 546, pp. 1208–1229, 2020.

[22] C. W. Wu, B. E. Shie, V. S. Tseng, and P. S. Yu, "Mining top-k high utility itemsets," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 78–86.

[23] V. S. Tseng, C. W. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining top-k high utility itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 54–67, 2015.

[24] Q. H. Duong, B. Liao, P. Fournier-Viger, and T. L. Dam, "An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies," *Knowledge-Based Systems*, vol. 104, pp. 106–122, 2016.

[25] S. Krishnamoorthy, "Mining top-k high utility itemsets with effective threshold raising strategies," *Expert Systems with Applications*, vol. 117, pp. 148–165, 2019.

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

[27] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 2014, pp. 83–92.

[28] J. Yin, Z. Zheng, and L. Cao, "USpan: an efficient algorithm for mining high utility sequential patterns," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2012, pp. 660–668.

[29] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, "Fast algorithms for mining high-utility itemsets with various discount strategies," *Advanced Engineering Informatics*, vol. 30, no. 2, pp. 109–126, 2016.

[30] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and H. C. Chao, "FDHUP: Fast algorithm for mining discriminative high utility patterns," *Knowledge and Information Systems*, vol. 51, no. 3, pp. 873–909, 2017.

[31] S. Krishnamoorthy, "A comparative study of top-k high utility itemset mining methods," in *High-Utility Pattern Mining*. Springer, 2019, pp. 47–74.

[32] H. Ryang and U. Yun, "Top-k high utility pattern mining with effective threshold raising strategies," *Knowledge-Based Systems*, vol. 76, pp. 109–126, 2015.

[33] C. J. Chu, V. S. Tseng, and T. Liang, "An efficient algorithm for mining high utility itemsets with negative item values in large databases," *Applied Mathematics and Computation*, vol. 215, no. 2, pp. 767–778, 2009.

[34] H. F. Li, H. Y. Huang, and S. Y. Lee, "Fast and memory efficient mining of high-utility itemsets from data streams: with and without negative item profits," *Knowledge and Information Systems*, vol. 28, no. 3, pp. 495–522, 2011.

[35] J. C. W. Lin, P. Fournier-Viger, and W. Gan, "FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits," *Knowledge-Based Systems*, vol. 111, pp. 283–298, 2016.

[36] S. Krishnamoorthy, "Efficiently mining high utility itemsets with negative unit profits," *Knowledge-Based Systems*, vol. 145, pp. 1–14, 2018.

[37] G. C. Lan, T. P. Hong, J. P. Huang, and V. S. Tseng, "On-shelf utility mining with negative item values," *Expert Systems with Applications*, vol. 41, no. 7, pp. 3450–3459, 2014.

[38] K. Singh, S. S. Singh, A. Kumar, and B. Biswas, "High utility itemsets mining with negative utility value: A survey," *Journal of Intelligent & Fuzzy Systems*, vol. 35, no. 6, pp. 6551–6562, 2018.

[39] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[40] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 12, pp. 1708–1721, 2009.