TOPIC: Top-k High-Utility Itemset Discovering

Jiahui Chen, *Member, IEEE,* Shicheng Wan, Wensheng Gan, *Member, IEEE,*
Guoting Chen, and Hamido Fujita, *Senior Member, IEEE*

***Abstract*—Utility-driven itemset mining is widely applied in many real-world scenarios. However, most algorithms do not work for itemsets with negative utilities. Several efficient algorithms for high-utility itemset (HUI) mining with negative utilities have been proposed. These algorithms can find complete HUIs with or without negative utilities. However, the major problem with these algorithms is how to select an appropriate minimum utility (*minUtil*) threshold. To address this issue, some efficient algorithms for extracting top-k HUIs have been proposed, where parameter k is the quantity of HUIs to be discovered. However, all of these algorithms can solve only one part of the above problem. In this paper, we present a method for TOP-k high-utility Itemset disCovering (TOPIC) with positive and negative utility values, which utilizes the advantages of the above algorithms. TOPIC adopts transaction merging and database projection techniques to reduce the database scanning cost, and utilizes *minUtil* threshold raising strategies. It also uses an array-based utility technique, which calculates the utility of itemsets and upper bounds in linear time. We conducted extensive experiments on several real and synthetic datasets, and the results showed that TOPIC outperforms state-of-the-art algorithm in terms of runtime, memory costs, and scalability.**

***Index Terms*—high-utility itemset, utility mining, top-k mining, threshold raising strategies.**

I. Introduction

FREQUENT itemset mining (FIM) [1]–[3] has been widely applied in many numerous domains in real-world applications. A classical case of FIM is market basket analysis [4]. The main target of FIM algorithm is to discover itemsets which are frequently represented; the classic sample is "*Bear* and *Diapers* Theory". However, simply investigating the frequency of itemsets causes other important factors to be ignored, for example, the quantity of goods that customers purchase or the profit of goods on sale. In FIM [1], [2], all items/objects are assumed to be the same quantity. For example, if a customer buys a loaf or ten loaves of bread in a store, the two items are regarded as the same quantity in FIM. Moreover, if a customer buys a luxury diamond or a cheap loaf, both goods are also regarded as having the same profit. Obviously, this case does not occur in real life as retailers and managers always focus on finding itemsets that will yield more profits. To address these issues, utility-driven itemset mining, also called high-utility itemset mining (HUIM) [5], has attracted considerable attention.

Utility-driven itemset mining [5]–[13] proposes a new concept called utility (i.e., importance or interest). It considers the quantity of items and their weight value (e.g., unit profit or price). Therefore, it plays a pivotal role in data mining. An itemset is called a high-utility itemset (HUI) if it has a higher utility value than the user-specified minimum utility (*minUtil*) threshold. Utility-driven mining has been widely applied in many practical applications, including user behavior analysis [14], website click-stream analysis [15], and cross-marketing analysis [16]. Compared with FIM algorithms, HUI mining is widely recognized as being more complicated. This is because frequency has the anti-monotonic property, which means that the superset of infrequent itemsets must be infrequent [1]. In contrast, utility is neither monotonic nor anti-monotonic, and it cannot cut off all non-HUIs and reduce the search space during the mining procedure. To address this limitation, Liu *et al.* [6] proposed an overestimation method based on the concept of transaction-weighted utilization (*TWU*), which has the downward closure property (anti-monotonic). Subsequently, several HUIM algorithms have adopted the *TWU*-based technique; however, all of them suffer from numerous candidate generations, itemset joining operations, and multiple database scanning. To solve these limitations, scholars have designed tree-based algorithms [7]–[9], [17], [18] for HUIM. Although tree-based algorithms can mine HUIs without generating many candidates, they need to scan databases more than once and generate numerous conditional sub-trees. All the aforementioned algorithms are categorized as two-phase model algorithms, and their typical feature is that the calculation process of HUIs is divided into two phases: 1) Generate a large number of candidates (or utility-based pattern tree) and 2) Select real high-utility itemsets from a small candidate set.

The above algorithms obviously require a lot of running time and memory because they generate too many candidate itemsets and require multiple database scans. To overcome these limitations, utility-list based algorithms such as HUI-Miner [12], HUP-Miner [19], HMiner [20], and FHM [21] have been proposed. All these algorithms can discover HUIs by constructing an utility-list structure by scanning the database only once and recursively mining HUIs in the memory. Utility-list is a vertical representation of a database, and it stores the key information of itemsets. However, all the previously discussed algorithms may calculate itemsets that do not appear in the database. To solve this limitation, a typical

This work was partially supported by National Natural Science Foundation of China (Grant No. 61902079 and Grant No. 62002136), the Key Areas Research and Development Program of Guangdong Province (Grant No. 2019B010139002), and the project of Guangzhou Science and Technology (Grant No. 201902020006 and Grant No. 201902020007). (Corresponding author: Wensheng Gan)

Jiahui Chen and Shicheng Wan are with the Department of Computer Sciences, Guangdong University of Technology, Guangzhou 510006, China. (E-mail: csjhchen@gmail.com and scwan1998@gmail.com)

Wensheng Gan is with the College of Cyber Security, Jinan University, Guangzhou 510632, Guangdong, China; and with Guangdong Artificial Intelligence and Digital Economy Laboratory (Pazhou Lab), Guangzhou 510335, China. (E-mail: wsgan001@gmail.com)

Guoting Chen is with the School of Science, Harbin Institute of Technology (Shenzhen), Shenzhen, China (E-mail: chenguoting@hit.edu.cn)

Hamido Fujita is with the Faculty of Software and Information Science, Iwate Prefectural University, Iwate, Japan (E-mail: HFujita-799@acm.org)

horizontal database projection-based algorithm called EFIM [22] was proposed. EFIM absorbs some outstanding ideas from d2HUP [23] in that it first proposes a reverse set enumeration tree. Afterward, it modifies the utility upper bounds and uses several pruning strategies to improve the performance. Although much work has been done on HUI mining, some scenarios exist in the real world in which retailers attract customers with promotions or discount tickets, that may cause some items bring negative utility values (i.e., profit) [24]. In previous algorithms, the utility of HUIs absolutely decreases when considering negative utility items, and the reason is discussed in [25]. The FHN algorithm [26] addresses this problem from the perspective of the utility list [12]. EHIN [10] calculates HUIs by dividing the items into positive and negative utility items. After obtaining a positive HUI, EHIN tries to add negative utility items and checks whether the extended itemset is still a HUI.

One of the biggest limitations in HUIM domain is the suitability of *minUtil* threshold, while users generally do not know how much they should set. Thus, specifying the *minUtil* threshold is a key task, and it is very challenging because it directly affects the results and performance of a HUIM algorithm. In a study by Wu *et al.* [27], they demonstrated that a small change in the *minUtil* threshold will get completely different execution time results. On the one hand, if we set the *minUtil* too low, we will obtain numerous HUIs and excessive amounts of time and memory may be wasted [28]. On the other hand, if we set the *minUtil* threshold too high, we will discover few HUIs so that most interesting patterns will be lost. To find an appropriate threshold, users often have to test the algorithm repeatedly, which is a trial-and-error approach. To overcome this drawback, top-k HUI mining was proposed [27], [29]–[31]. In top-k HUIM, set k is used instead of threshold, where k indicates that the user deserves the quantity of HUIs. Although top-k pattern mining is practicable, it is more difficult to adopt than *minUtil* threshold for finding complete HUIs. The key point is that top-k pattern mining algorithm needs to store potential top-k patterns in the memory anytime, and it requires *minUtil* to be automatically raised when finding HUIs.

In our literature survey, a state-of-the-art algorithm called TopHUI [31] was used to mine top-k HUIs with or without negative utility values; however, it suffers from mining performance bottleneck such as long execution time and high memory cost. In this paper, we propose an efficient method called **TOP**-k high-utility **I**temset dis**C**overing (TOPIC) with positive and negative utility values to efficiently solve all the above challenges. The novel contributions of TOPIC are that it can effectively discover the exact top-k HUIs with negative utility values in a large database. The main contributions of this study are as follows:

- The novel algorithm adopts database-projection and transaction-merging techniques to reduce run time and memory consumption while processing.
- One of the key challenges is calculating the utility of HUIs without generating numerous candidates. Hence, we adopted an efficient array-based utility-counting technique to obtain the *TWU* and upper bounds of itemsets in linear time.
- To store the top-k HUIs, we used a priority queue, and employed the *minUtil* threshold-raising strategy for *minUtil* efficiently and automatically increase.
- Extensive experimental evaluations were conducted on both real and synthetic datasets to evaluate the proposed algorithm. We also compared the performance of TOPIC and TopHUI. The results show that our algorithm is efficient in terms of both run time and memory consumption. Additionally, it is superior to TopHUI for dense datasets.

The remainder of this paper is organized as follows: In Section II, related works on traditional utility mining and top-k domains are introduced. In Section III, some basic preliminaries and the problem statement of top-k HUI mining are introduced. Furthermore, a novel TOPIC algorithm is proposed in Section IV. The experimental results are presented in Section V, and conclusion and future work are presented in Section VI.

II. Related work

In this section, we briefly review some studies about high-utility (positive and negative) itemset mining and top-k high-utility itemset mining.

A. High-Utility Itemset Mining

Since the first HUIM algorithm called Two-Phase [6] was proposed, researchers have conducted many studies on HUIM algorithms, including UMining [32], IHUP [17], BAHUI [33], and HUI-Miner [12]. All HUIM algorithms can be classified into two-phase and single-phase model algorithms. The biggest difference between these two types of algorithms depends on whether they generate numerous candidate itemsets. The Two-Phase algorithm prunes the search space uses *TWU* concept. Similar to FIM, it has the downward closure property, in which the superset of an itemset cannot be a HUI if its *TWU* is less than the *minUtil* threshold. There are two shortcomings of the two-phase model algorithm: 1) it produces numerous candidates; 2) it calculates HUIs by scanning the dataset at least twice. Meanwhile, IHUP [17] solves these limitations by constructing a search tree only scan dataset once, and it reduces the number of candidates generated; thus, it performs better than the two-phase model algorithms. However, the upper bound of the IHUP is not sufficiently accurate, and many unpromising supersets are produced because of the overestimated HUIs. Tseng *et al.* [7] designed an influential tree-based algorithm called UP-Growth to maintain the information of items to mine HUIs.

In the HUIM domain, the HUI-Miner algorithm [12] is a breakthrough work, and it is a single-phase model algorithm. Compared with two-phase model algorithms, the single-phase model algorithm calculates HUIs while generating candidates. HUI-Miner through a new list structure called utility-list, a pair of utility-lists of length l-1 intersect to obtain a utility-list of length l. It successfully avoids the problem of numerous candidates because of the remaining utility [12]. The highlight of HUI-Miner is that it uses utility-list to calculate HUIs in the memory instead of scanning the dataset multiple times.

Meanwhile, Zida *et al.* [22] proposed the EFIM algorithm, which has a higher accuracy than the *TWU*-based pruning algorithm (two new upper bounds: *local utility* and *revised sub-tree utility*). They designed two new techniques *high-utility database projection* and *high-utility transaction merging* to improve the efficiency of dataset scanning. Additionally, several HUIM algorithms have been extensively studied to extend the effectiveness of HUIM, such as incremental HUIM [34], concise representation-based HUIM [35], [36], top-k HUIM [27], [30], [37], HUIM from uncertain data [38], [39], and so on [40]. More details on utility-oriented pattern mining can be obtained from Gan *et al.* [5]. Although much work has already been done in this data mining field, few studies considered negative utility items.

B. High-Utility Itemset Mining with Negative Utilities

HUINV-Mine [25] is the first level-wise algorithm to explore negative effects in the HUIM domain. It is a two-phase model algorithm, and it adopts a relatively rough method to solve the negative items problem by pruning itemsets that comprise only negative utility items. However, it does not propose an excellent method for dealing with negative utility items. Meanwhile, UP-GNIV [18] is based on set enumeration tree-based concept and does not generate candidates. It is a modification of UP-Growth algorithm [7], and it is used to find interesting patterns that include negative utilities of items. The performance of UP-GNIV is better than that of HUINIV-Mine. Furthermore, Lin *et al.* [26] proposed FHN (modified from FHM [21]), a utility-list-based mining algorithm, to solve the problems of tree-based algorithms that find long itemsets by recursively searching shorter itemsets. FHN discovers HUIs from a set of transactions in a vertical data format, whereas level-wise and tree-based algorithms use a horizontal data format. The most interesting highlight is the construction of the PNU-list, which is a tuple (*tid*, *putil*, *nutil*, *rputil*) [26]. It also utilizes the EUCS structure [21], EUCP strategy [21], and LA-Prune strategy [19] to discover HUIs more efficiently. Inspired by the idea of the PNU-list, the EHIN algorithm [10] separately lists negative utility items and tries to add negative utility items after calculating positive HUIs. Afterward, it checks whether they are still HUIs. Gan *et al.* [41] then proposed a novel algorithm that can discover HUIs with negative utility values from an uncertain dataset. The algorithm constructs a probability utility-list with a positive-and-negative utility (PU±-list) structure to maintain positive and negative utility items. However, a question that is the biggest limitation of utility-driven pattern mining algorithms arises: how do we appropriately set the minimum utility threshold?

C. Top-k High-Utility Itemset Mining

As previously mentioned, Wu *et al.* [42] demonstrated that a slight difference in the *minUtil* threshold can lead to a significance difference in the number of candidates generated. In top-k HUIM, the parameter k replaces the *minUtil* threshold. Until now, most top-k HUIM algorithms have been based on modifications of previous positive HUIM algorithms [27], [29]. TKU [27] is an extension of the UP-Growth algorithm with some efficient threshold-raising strategies, and it is also the first top-k HUI mining algorithm. Moreover, REPT [30] stores item information through a pre-evaluation matrix in descending order.

Meanwhile, some researchers on top-k HUIM have sought more efficient methods by adopting single-phase model. The TKO [27] algorithm utilizes the idea of HUI-Miner, and it uses a structure called PE-matrix to increase *minUtil*. Moreover, it adopts the DGU pruning strategy to remove inefficient items during dataset scanning. KHMC [29] is an extension of FHM. By employing three effectively threshold-raising strategies (RIU, CUD, COV), the COV strategy not only prunes the search space in HUI mining but also raises the *minUtil* threshold and optimizes the EUCS structure by using Hash-Map to construct a new EUCST structure. Additionally, KHMC designs a new co-occurrence pruning technique called EUCPT to address the problem of joining operation costs when calculating the utilities of itemsets HUI-Miner does. KHMC performs better than the TKO and REPT algorithms for top-k HUI mining in terms of memory consumption and execution time. Recently, Gan *et al.* [31] proposed a top-k HUIM algorithm called TopHUI, which is the first can work in a transaction database comprising various types of itemsets with positive and negative utilities. It is an extension of THUI [28]. However, TopHUI adopts the PNU-list and thus carries on limitations of FHN [26], as discussed above.

Although there are many studies on HUI mining or top-k HUI mining, few of them focus on how to discover top-k high-utility (positive or negative) itemsets. This study addresses this gap by mining top-k high-utility itemsets with negative utilities. The proposed algorithm adopts novel search-space pruning strategies to effectively find the correct top-k HUIs.

III. Preliminaries and Problem Definition

In this section, we present some properties and definitions adopted in the TOPIC algorithm. Most of them were introduced by Liu and Qu [12], Zida *et al.* [22], Singh *et al.* [10], and Li *et al.* [43].

A. Basic Definitions

Definition 3.1: (**Transaction database**) Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of distinct items which may be positive or negative. An itemset is defined as a set $X \subseteq I$. $D = \{T_1, T_2, \ldots; T_n\}$ is a transaction database, where each transaction $T_j \in D$ has a unique identifier j called its *TID* (transaction ID) and n is the number of transactions in D. Table I lists the sample database.

TABLE I
A TRANSACTION DATABASE

Tid	Transaction (item, quantity)
T_1	$(A,1)\,(D,2)\,(E,1)$
T_2	$(B,1)\,(C,2)\,(D,6)$
T_3	$(A,3)\,(D,5)$
T_4	$(A,1)\,(E,1)$
T_5	$(B,1)\,(C,2)\,(D,6)$
T_6	$(B,1)\,(C,1)\,(E,2)$

Definition 3.2: (**Internal utility** and **external utility**) Let x be an item. $IU(x, T_j)$ is specified as an internal utility (e.g., purchase quantity) of x, and $EU(x)$ is specified as an external utility (e.g., unit profit) of x. $EU(x)$ represents the relative importance of x to users. Table II lists the external utility of each item.

TABLE II
EXTERNAL UTILITY VALUES

Item	A	B	C	D	E
External utility	\$5	\$-3	\$-2	\$6	\$10

Definition 3.3: (**Utility of an item**) The utility of item x in transaction T_j is defined as $U(x, T_j) = IU(x, T_j) \times EU(x)$. This indicates how much profit can be generated according to the sale of item x in transaction T_j. The utility of item x in D is denoted by $U(x) = \sum_{x \in T_j \subseteq D} U(X, T_j)$, and it is used to identify the item x that users need. This is a significant HUIM evaluation standard.

From Table I, the database contains six transactions (T_1, T_2, ..., and T_6). Transaction T_2 consists of items B, C, and D, which have quantities 1, 2, and 6, respectively. Choosing an item E and computing its utility in D, $U(E) = U(E, T_1) + U(E, T_4) + U(E, T_6) = \$10 + \$10 + \$20 = \$40$.

Definition 3.4: (**Utility of an itemset**) Let an itemset X consist of $\{x_1, x_2, \ldots, x_m\}$. The utility of X in transaction T_j is defined as $U(X, T_j) = \sum_{x_i \in X \wedge X \subseteq T_j} U(x_i, T_j)$ ($1 \leq i \leq m$). The utility of an itemset X in a database is defined as $U(X) = \sum_{X \subseteq T_j \in D} U(X, T_j)$ ($1 \leq j \leq n$).

For example, if an itemset $X = \{A, D\}$, we obtain $U(\{AD\}) = U(\{AD\}, T_1) + U(\{AD\}, T_3) = (\$5 \times 1 + \$6 \times 2) + (\$5 \times 3 + \$6 \times 5) = \$62$.

Definition 3.5: (**High-utility itemset**) An itemset X is referred as HUI if $U(X) \geq$ *minUtil* is true. This means that we suppose these itemsets are interesting.

Definition 3.6: (**Remaining utility**) Given a transaction T_j, all items x that upper than itemset X are defined as T_j/X, where $x \in T_j/X$ (the comparison rule here can be replaced by $\succ_T$ and this symbol will be explained further in the next section). The remaining utility of X in transaction T_j is defined as $RU(X, T_j) = \sum_{x_i \in (T_j/X)} U(x_i, T_j)$ [12].

As mentioned above, many items compose transactions. All these items have utility properties and transactions also have utility properties. Why do we need to calculate transaction utility? In FIM, if an itemset is frequent, all its supersets are also frequent. For example, if we calculate the utility of item A and its supersets $\{A, D\}$, $\{A, E\}$, and $\{A, D, E\}$ in the database as \$25, \$62, \$30 and \$27, we can clearly see that the utility of $\{A, D, E\}$ is less than that of $\{A, D\}$ and $\{A, E\}$ but greater than $\{A\}$. The utility values of itemsets are neither monotonic nor anti-monotonic. Thus, Liu and Qu [12] proposed a new concept called *TWU*. Herein, we will introduce the following concepts.

Definition 3.7: (**Utility of transaction**) The utility of a transaction T_j is defined as $TU(T_j) = \sum_{x_i \in T_j} U(x_i, T_j)$.

Definition 3.8: (**Transaction-weighted utilization**) The transaction-weighted utilization (*TWU*) of itemset X in D is a utility upper bound, which refers to the sum of transaction utilities that contain X. It is denoted as $TWU(X)$ and defined as $TWU(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} TU(T_j)$.

TABLE III
UTILITY OF TRANSACTION

Tid	T_1	T_2	T_3	T_4	T_5	T_6
Utility	\$27	\$29	\$45	\$15	\$29	\$15

We can compute the real utility of a transaction T_1 as $TU(T_1) = U(A, T_1) + U(D, T_1) + U(E, T_1) = \$27$. The computation of the utility of other transactions are given in Table III. By calculating the *TWU* of item A, transactions T_1, T_3, and T_4 contain A, as given in Table I. Hence, $TWU(A) = TU(T_1) + TU(T_3) + TU(T_4) = \$27 + \$45 + \$15 = \$87$. Table IV lists the items and their *TWU* values for the sample transaction database of Table I.

TABLE IV
TRANSACTION WEIGHTED UTILIZATION

Item	A	B	C	D	E
TWU	\$87	\$73	\$73	\$130	\$57

Property 3.1: (**Transaction-weighted downward closure property**) If the *TWU* of an itemset X is less than the *minUtil* threshold, then X and all its supersets are low-utility itemsets. This property is usually exploited as a key pruning strategy for mining HUIs. Its proof process is given in [6].

Property 3.2: (***TWU*-based pruning strategy** [6]) From the above properties and definitions, it can be inferred that if the *TWU* value of X is less than the user-specified (*minUtil*) threshold ($TWU(X) <$ *minUtil*), then this itemset and its supersets are low-utility itemsets. Subsequently, we can remove them from the search space.

Definition 3.9: (**High transaction-weighted utilization itemset** [6]) After pruning the itemset by *TWU*, the remaining itemsets are a set called high transaction-weighted utilization itemsets (HTWUIs), which are potential HUIs. We need to further inspect the HTWUIs to find the true HUIs. If we set *minUtil* to \$70, we can obtain the HTWUIs listed in Table V.

TABLE V
HIGH TRANSACTION-WEIGHTED UTILIZATION ITEMSETS

HTWUI	***Utility***	***HTWUI***	***Utility***
$\{A\}$	\$70	$\{C\}$	\$73
$\{A, D\}$	\$72	$\{D\}$	\$130
$\{B\}$	\$73	$\{B, C\}$	\$73

B. Dealing with Negative Utilities

Property 3.3: (**Relationship between positive and negative utility itemsets**) Given any itemset X, the positive utility of X in a transaction or database is defined as *pUtil(X)*, whereas its negative utility is defined as *nUtil(X)*. Therefore, the real utility of an itemset X in a transaction or database is given as $U(X) = pUtil(X) + nUtil(X)$. It can be inferred that $pUtil(X) \geq U(X) \geq nUtil(X)$ [26], [44].

Most HUIM algorithms [6]–[10], [12], [21] adopt a *TWU*-based pruning strategy. *TWU* not only supports overestimation

for mining HUIs but is also used to prune the search space. However, *TWU* cannot be directly applied to items with negative utilities because $\{B\}, \{C\}$ and $\{B, C\}$ are mistaken for *HTWUI* in Table V. To address this error, Chu *et al.* [25] first redefined the utility value of a transaction and *TWU*. We will introduce them in the following.

Definition 3.10: (**Redefined transaction-weighted utilization** [25]) To avoid the challenges that negative utility items bring, the redefined transaction utility is given as $RTU(T_j)$ for a transaction T_j, considering only the positive external utility. Thus, $RTU(T_j) = \sum_{x \in T_j \wedge EU(x) > 0} U(x, T_j)$. The redefined transaction-weighted utilization (*RTWU*) of an itemset X is given as $RTWU(X) = \sum_{X \subseteq T_j \in D} RTU(T_j)$.

If $j = 2$, $RTU(T_2) = TU(D) = \$6 \times 6 = \$36$. In particular, the redefined transaction utility of items with negative utility is set as \$0; hence, we can easily deduce $RTU(T_j) \geq TU(T_j)$. Assume itemset $X = \{A\}$, transactions T_1, T_3, and T_4 should be considered. Thus, $RTWU(A) = RTU(T_1) + RTU(T_3) + RTU\ (T_4)$ = \$27 + \$45+ \$15 = \$87. Table VII lists the corresponding *RTWU* of the items.

TABLE VI
REDEFINED TRANSACTION WEIGHTED UTILITY

Item	A	B	C	D	E
RTWU	\$87	\$92	\$92	\$144	\$62

Property 3.4: (***RTWU*-based pruning strategy**) For an itemset X, if $RTWU(X) < minUtil$, then X is not a HUI and all supersets of X are low-utility itemsets. The details of the proof can be found in [26].

Definition 3.11: (**Potential top-k high-utility itemset**) An itemset is regarded as a potential top-k high-utility itemset (PKHUI) if its estimated utility value (i.e., *TWU*) is higher than the current *minUtil* threshold. In other words, if the *TWU* of this item is higher than the utility of the k-th itemset, it may be referred to as a top-k HUI, and the contents of the top-k HUIs will be adjusted.

Definition 3.12: (**Top-k high-utility itemset**) An itemset X is called a top-k HUI if there is a list only k-1 itemsets which utility values are higher than $U(X)$, and X is the k-th highest utility itemset in this list. In particular, k is a user-specified parameter.

Given an itemset α, some items that can be added to α are defined as $E(\alpha) = \{z \mid z \in I \wedge z \succ x, \forall x \in \alpha\}$ [22] (the symbol "$\succ$" will be explained in the next section). If $k = 5$, the top five highest utility itemsets containing negative utility items in the sample database are displayed in Table VII, and the final *minUtil* threshold is \$58.

TABLE VII
TOP-5 HIGH-UTILITY ITEMSETS

Itemset	**Utility**
$\{D\}$	\$144
$\{B, D\}$	\$66
$\{C, D\}$	\$64
$\{A, D\}$	\$62
$\{B, C, D\}$	\$58

IV. THE TOPIC ALGORITHM

In this section, we present the TOPIC algorithm for mining top-k HUIs with negative utility values. In Subsection IV-B, two efficient database scanning techniques are utilized, namely: database projection and transaction merging. In Subsection IV-C, we explain how to calculate the upper bounds (redefined sub-tree utility and redefined local utility) using utility array (UA). In Subsection IV-D, we propose an efficient and automatic *minUtil* threshold-raising strategy. In Subsection IV-E, we present the pseudo-code of the TOPIC algorithm and describe it in detail.

A. Upper Bounds on Utilities for Pruning Search Space

Definition 4.1: (**Extension of an itemset** [22]) If an itemset α can be extended into itemset $Y = \alpha \cup \{X\}$, where $X \in 2^{E(\alpha)}$, and X should not be empty. Similarly, if α can be extended with a single itemset $\{z\}$ that contains only one item, $Y = \alpha \cup \{z\}$, where $z \in E(\alpha)$.

Definition 4.2: (**Extension of a negative itemset** [10]) Itemset α can be extended to itemset Y, $Y = \alpha \cup \{X\}$, where X is a set of items with negative utility.

The quantity of transactions contain itemset $\alpha \cup \{X\}$ is less or equal than the number of transactions contains itemset α. α that extends with positive utility items may be higher or equal to or lower than $U(\alpha)$. However, when α is extended with a negative utility item $\{X\}$, it must be lower than $U(\alpha)$. Furthermore, if $U(\alpha) \geq minUtil$, then we can try to add $\{X\}$ to α. If $U(\alpha \cup \{X\})$ is still higher than or equal to *minUtil*, then $\alpha \cup \{X\}$ is a HUI. We can know that if $\alpha = \{A\}$, then in transaction T_1, $E(\alpha) = \{D, E\}$ from Table I. And extensions of α in lexicographical order are $\{A, D\}$, $\{A, E\}$ and $\{A, D, E\}$. [10] introduces this rationale and proof.

We set $\succ$ as the total order of items. Our novel algorithm is updated based on the EFIM algorithm, and it explores the search space by using a depth-first search starting from the root (which is an empty set). To make any itemset α become larger, TOPIC recursively appends item x_i to α individually through the $\succ$ order. If we only consider the positive items, the $\succ$ order is sorted by increasing *TWU* [12], [27]. However, in order to efficiently use the projection technique during the database scanning, each item and original transaction are sorted according to the $\succ$ total order. Moreover, items are sorted by the *RTWU*-ascending order. If the *RTWU* of the items are equal, then the $\succ$ total order follows the lexicographical order. Particularly, negative items always follow positive items in the sorting rule. Afterward, pseudo-projection is performed in each projection; in other words, each projected transaction is represented by an offset pointer on the corresponding original transaction [10], [22].

Note that $pUtil(X) \geq U(X) \geq nUtil(X)$. Inspired by previous studies [10], [26], [44], we only take $pUtil(X)$ into account and ignore all items with negative external utility. With this overstatement, then we adopt the following upper-bound concepts in our novel top-k utility mining algorithm.

Definition 4.3: (**Redefined local utility and redefined sub-tree utility**) The redefined local utility (*RLU*) of item x with respect to an itemset α that may contain both

positive and negative utilities is defined as $RLU(\alpha, x) = \sum_{(\alpha \cup \{x\}) \subseteq T_j \wedge T_j \subseteq D} [U(\alpha, T_j) + RU(\alpha, T_j)]$, subject to $EU(x) > 0$. The redefined sub-tree utility (*RSU*) of item x with respect to itemset α (the addition of x to α follows the depth-first search of the sub-tree) is defined as follows: $RSU(\alpha, x) = \sum_{(\alpha \cup x) \subseteq T_j \wedge T_j \subseteq D} [U(\alpha, T_j) + U(x, T_j) + \sum_{i \in T_j \wedge i \in E(\alpha \cup \{x\})} U(i, T_j)]$, subject to $EU(x) > 0$.

Note that the original concepts of local utility and sub-tree utility are defined in EIFM [22]. For example, if $\alpha = \{A\}$, then $RLU(A, D) = (U(\{A\}, T_1) + RU(\{D\}, T_1)) + (U(\{A\}, T_3) + RU(\{D\}, T_3)) = \$15 + \$15 = \$30$. If $\alpha = \{A\}$, then $RSU(A, D) = (U(A, T_1) + U(D, T_1) + \$0) + (U(A, T_3) + U(D, T_3) + \$0) = \$17 + \$45 = \$62$. Obviously, the negative utility items are not computed here.

Property 4.1: (**Redefined local utility-based overestimation**) Given an item x and an itemset α, where $x \in E(\alpha)$, and x is an extension of α, then $x \in X$ (X is a sub-itemset in $E(\alpha)$). Therefore, $RLU(\alpha, x) \geq U(X)$ always holds. Furthermore, if $RLU(\alpha, x) < minUtil$, then the item x and all extensions of α containing item x have low utility in a sub-tree. Thus, x and its supersets can be pruned to explore all sub-trees of α.

Property 4.2: (**Redefined sub-tree utility-based overestimation**) Given an item x and an itemset α, where $\forall x \in E(\alpha)$, and x can be an extension of α, then $x \in X$ (X is a sub-itemset belongs to $E(\alpha)$). Therefore, $RSU(\alpha, x) \geq U(X)$ always holds, when dealing with the database which may contain both positive and negative utilities. Furthermore, if $RSU(\alpha, x) < minUtil$, then x and all extensions of α that contain x have low utility in the sub-tree. Thus, x and its supersets can be pruned while exploring all sub-trees of α.

The indirectly proof of the above two properties are demonstrated in EFIM [22]. It explains why the upper bound *RLU* are tighter than *TWU*. It shows that *RSU* and *RU* are mathematical equivalents. The major difference is their calculation methods are depth-first searching and child itemsets, respectively. Hence, *RSU* cuts off the whole sub-tree of α, including nodes x, and *RU* prunes only the descendants of α. Therefore, we utilized the *RSU* upper bound rather than the *RU* upper bound to prune the search space. Subsequently, we categorized itemset α into $primary(\alpha)$ and $secondary(\alpha)$.

Definition 4.4: (**Primary and secondary sets** [22]) For an itemset α in a given database, the *primary* items of α are given as $Primary(\alpha) = \{x \mid x \in E(\alpha) \wedge RSU(\alpha, x) \geq minUtil\}$, and the *secondary* items of α are given as $Secondary(\alpha) = \{x \mid x \in E(\alpha) \wedge RLU(\alpha, x) \geq minUtil\}$. Because $RLU(\alpha, x) \geq RSU(\alpha, x)$, $primary(\alpha) \subseteq secondary(\alpha)$. $Secondary(\alpha)$ indicates items that are extensible, as all items can combine with another distinct item to form an itemset. This means that extendable items and all items in α can be extended by other elements of $E(\alpha)$. In addition, $primary(\alpha)$ indicates items that are searchable, and each item of this set can be an extension element to expand $secondary(\alpha)$ items.

In particular, the *RSU* upper bound cannot be directly applied in vertical algorithms such as HUI-Miner, FHM, and FHN because once the utility list is established, these algorithms do not need to perform database scanning again

B. Scanning Using Projection and Merging

Database scanning using projection technique. This novel algorithm utilizes a database projection technique to reduce the memory consumption and speed up the run time. When an itemset α is considered when depth-first searching and scanning the transactions of database D to calculate the utility of itemsets within the sub-tree of itemset α, those items that do not belong to the α extension are pruned. Database without these items (which is pruned) is called projected database [21], [22], [26].

Definition 4.5: (**Projected transaction and projected database** [22]) For an itemset α, the projected transaction T_j is defined as α-$T_j = \{x | x \in T_j \wedge x \in E(\alpha)\}$. The projected database D is defined as α-$D = \{\alpha\text{-}T_j | T_j \in D \wedge \alpha\text{-}T_j \neq \emptyset\}$. As given in Table I, if an itemset $\alpha = \{A\}$, then the projected database α-$T_1 = \{D, E\}$, α-$T_3 = \{D\}$, and α-$T_4 = \{E\}$. α-D contains these transactions.

Database scanning using merging technique. Our novel algorithm also utilizes the transaction-merging technique to reduce the database scanning cost. After the database is projected, some identical transactions (which may contain the same items but do not have the same internal utility values) or empty transactions may exist. Merging technique is used to replace these identical transactions with a single transaction [10], [22]. If T_i is identical to T_j, it represents two transactions containing the same items. However, they may not have the same internal utility (purchase quantity) for each item.

Definition 4.6: (**Transaction merging** [19]) In a database D, several identical transactions such as $\{T_{j_1}, T_{j_2}, \ldots, T_{j_n}\}$ are replaced by a new transaction $T_M = T_{j_1} = T_{j_2} = \ldots = T_{j_n}$. The quantity of each item x in these identical transactions is $IU(x, T_M) = \sum_{1 \leq i \leq n} IU(x, T_{j_i})$.

For instance, we can observe from Table I that transactions T_2 and T_5 are identical. After merging the transactions, a new transaction T_{25} is obtained, where $IU(B, T_{25}) = 2$, $IU(C, T_{25}) = 4$, and $IU(D, T_{25}) = 12$.

Definition 4.7: (**Projected transaction merging** [22]) If there are several identical projected transactions such as $\{T_{j_1}, T_{j_2}, \ldots, T_{j_n}\}$, they are replaced by a new transaction $T_M = T_{j_1} = T_{j_2} = \ldots = T_{j_n}$ in database α-D. The internal utility of each item $x \in T_M$ is defined as $IU(x, T_M) = \sum_{1 \leq i \leq n} IU(x, T_{j_i})$.

For example, if an itemset $\alpha = \{A\}$, then the projected database α-D contains transactions α-$T_1 = \{D, E\}$, α-$T_2 = \emptyset$, α-$T_3 = \{D\}$, α-$T_4 = \{E\}$, α-$T_5 = \emptyset$, and α-$T_6 = \emptyset$. Thus, transactions α-T_2, α-T_5, and α-T_6 can be replaced by a new transaction $T_{256} = \emptyset$.

When identifying identical transactions, a naive method is used to compare each transaction, which is inefficient. To make the transaction merging technique more efficient, we adopt a new total order $\succ_T$ on the transactions in the database before merging [10], [22].

Definition 4.8: (**Total order on transactions** [22]) The $\succ_T$ order is defined as the lexicographical order when reading all transactions from back to front. Further details about $\succ_T$ follow the EFIM algorithm [22].

If there are three transactions $T_x = \{a, b, c\}$, $T_y = \{a, b, e\}$, and $T_z = \{a, b\}$, then $T_y \succ_T T_x \succ_T T_z$.

Property 4.3: (**Transaction order in $\succ_T$-sorted database** [22]) If there is an itemset α and $\succ_T$-sorted database D, identical transactions appear consecutively in the projected database α-D.

Proof: First, while reading the transactions backward, all of them are sorted in lexicographical order. Second, projections always prune the lowest items of a transaction in lexicographical order. For more details and analysis, refer to Ref. [22]. ■

C. Calculation of Upper Bounds using Utility Array

Novel upper bounds are vital for pruning the search space. After searching the utility itemset mining literature, we utilize an array-based structure called *UA*.

Definition 4.9: (**Utility array** [22]) In a database D, there is a set of items I. The array element for an item x in the array is given as $UA[x]$. Each element stores the utility value of the item x, and *UA* has a length of $|I|$.

Calculate *RLU*(α) using UA. First, *UA* is initialized by filling all the elements with 0. Second, $UA[x] = UA[x] + U(\alpha, T_j) + RU(\alpha, T_j)$, where $x \in T_j \cap E(\alpha) \wedge \forall T_j \subseteq D$. After database scanning, $\forall x \in E(\alpha)$, $UA[x] = RLU(\alpha, x)$, which gives the local utility of all positive itemsets.

Calculate *RSU*(α) using UA. First, *UA* is initialized by filling all the elements with 0. Second, $UA[x] = UA[x] + \sum_{I \in T_j \wedge I \in E(\alpha \cup x)} U(I, T_j) + U(\alpha, T_j) + U(x, T_j)$, where item $x \in T_j \cap E(\alpha) \wedge \forall T_j \subseteq D$. After database scanning, $\forall x \in E(\alpha)$, $UA[x] = RSU(\alpha, x)$.

According to the UA technique, we can obtain the upper bounds of utility in linear time. For more details and comparisons, refer to [10].

D. Threshold Raising Strategy

A key method is to automatically increase the (*minUtil*) threshold, and our new algorithm sets the *minUtil* threshold value to 1 at the beginning. The TopHUI algorithm [31] proposes that the threshold should be raised based on the RTU (raising threshold based on transaction utilities) strategy. The REPT [30] introduces a real item utilities (RIU) threshold raising strategy. TOPIC also utilizes it to increase the *minUtil*. Other *minUtil* raising strategies and their detailed discussion are given in [31].

Algorithm 1 RIU strategy

Input: *top-k list*: a list of utility values for all items, k: the desired number of HUIs.
Output: *minUtil*.

1: sort top-k list by descending order;
2: **if** | top-k list | $\geq k$ **then**
3: set the k-th highest value as a new current *minUtil*;
4: **end if**
5: **return** *minUtil*

Here, we provide a brief introduction of **Algorithm 1**. After calculating $\sum_{T_j \in D} U(x, T_j)$ for all the items, it is added to the top-k list as an input parameter. The subscript k indicates that the user specifies the number of HUIs they need. Afterward, all elements in the top-k list are sorted in descending order. This operation will help us obtain the k highest utility for convenience (Line 1). If the length of the top-k list is higher than k, then the current *minUtil* is raised to the k-th highest value (Lines 2–4). Finally, we obtain a new *minUtil* as the output (Line 5).

E. The TOPIC Algorithm

The proposed algorithm TOPIC (**Algorithm 2**) adopts some new techniques mentioned in the previous sections. It mainly takes a transaction database and a user-specific parameter k as input parameters and returns the top-k HUIs. In Lines 1–4 of the algorithm, the empty itemsets are separately initialized as α. ρ stores a set of positive and negative utility items in the database as η, and the minimum utility threshold value is 1. In Line 5, a k priority queue is created to maintain a "candidate" *minUtil* to raise the *minUtil*. In Line 6, the real utility values of all items $z \in I$ is computed and a list *RIU* is used to store these values. Subsequently, the threshold-raising utility function is called to increase the current *minUtil* threshold (Line 7). Afterward, the RLU of each item is calculated using an array (Line 8), and it prepares to select items that can be expanded. Items whose RLUs are higher than the current *minUtil* are then selected to form the *secondary* set (Line 9), and the *secondary* items are sorted in ascending order of *RTWU* (Line 10). Negative utility items are always followed by positive utility items in the algorithm. In Line 11, all low-utility items are removed based on database scanning (*RTWU*-based pruning strategy). Afterward, empty transactions are deleted (Line 12) because there may be some transactions that have only items that are already removed in Line 11. Thereafter, the remaining transactions are sorted by $\succ_T$ using lexicographical order in Line 13. Transaction merging is performed in Line 14, and in Line 15, the remaining transactions are scanned again and a UA is used to calculate $RSU(\alpha, z)$, where items $z \in$ *secondary*(α). A new set of *primary*(α) items is then obtained (Line 16), which will help to prune the search sub-tree. In Line 17, the negative utility items are stored in the global variable because it needs to try to add these items in the HUI to consider whether it would still be a HUI. The **search_P** procedure is called in Line 18 starting with itemset α in the depth-first search. Finally, the top-k high-utility itemsets are returned.

The **Algorithm 3** has seven input parameters: α is the current itemset prepared to be extended (it is initialized as an empty set), η denotes a set of negative utility items, α-D is the current projected database (it is initially an original database), the *primary* set contains primary items of itemset α, the *secondary* set contains secondary items of itemset α, *minUtil* represents the raised minimum utility threshold, and k *patterns* is a priority queue of k items. This algorithm recursively calls itself to extend each positive item of α to constantly find extensions of α. Line 2 starts traversing each item $z \in$ *primary*(α), and these are regarded as extensible items. In Line 3, each item z is combined with α to form a new itemset β. Based on the scanned database α-D, the utility of itemset β is calculated and a new merging and projection

Algorithm 2 Proposed TOPIC algorithm

Input: D: a database, k: the desired number of HUIs.
Output: Top-k HUIs with negative utility items.

1: initialize $\alpha \leftarrow \emptyset$;
2: initialize $\rho \leftarrow$ a set of positive utility items;
3: initialize $\eta \leftarrow$ a set of negative utility items;
4: initialize *minUtil* $\leftarrow 1$
5: create a priority queue of size k
6: compute real utility of all items $z \in I$, and store values into list *RIU*;
7: call **RIU**(*RIU, k*) to raise the *minUtil*;
8: scan all transactions, using utility-array to calculate *RLU*(α, z) of all items $z \in \rho$;
9: *Secondary*$(\alpha) = \{z | z \in \rho \wedge RLU(\alpha, z) \geq minUtil\}$;
10: sorted *Secondary*(α) by using the total order $\succ$ of *RTWU* increasing values;
11: scan D, remove low utility items $x \notin$ *Secondary*(α) from transactions;
12: remove all empty transactions;
13: sort all remaining transactions according to the $\succ_T$ using lexicographical order;
14: assign offset to each transaction in D;
15: scan all remaining transactions in D, using utility-array to calculate *RSU*(α, z) for all items $z \in$ *Secondary*(α);
16: calculate *Primary*$(\alpha) = \{z | z \in$ *Secondary*$(\alpha) \wedge RSU(\alpha, z) \geq minUtil\}$;
17: store the negative items in global variate;
18: call **search_P**(η, α, D, *Primary*(α), *Secondary*(α), *minUtil*, *k-patterns*);
19: **return** top-k HUIs

Algorithm 3 The *search_P* procedure

Input: α: the current itemset, η: a set of negative items, α-D: the current projected database, *Primary*(α): the *Primary* items of α, *Secondary*(α): the *Secondary* items of α, *minUtil*: a raised minimum utility threshold, and *top-k list*: a priority queue of k items.
Output: a set of top-k HUIs that are extensions of α with positive utility items.

1: **for** each item $z \in$ *Primary*(α) **do**
2: $\beta = \alpha \cup \{z\}$;
3: scan α-D, calculate $U(\beta)$, and create β-D;
4: **if** $U(\beta) \geq$ *minUtil* **then**
5: add β into *top-k list*;
6: **if** $|$*top-k list*$| > k$ **then**
7: pop the k-th values in *top-k list*;
8: raise current *minUtil* with the k-th value;
9: **end if**
10: **end if**
11: **if** $U(\beta) >$ *minUtil* **then**
12: call **search_N**(η, β, $\beta - D$, *minUtil*).
13: **end if**
14: scan β-D, calculate *RSU*(β, z), and *RLU*(β, z) where items $z \in$*Secondary*(α), using two *UAs*;
15: obtain *Primary*$(\beta) = \{z \in$ *Secondary*$(\alpha) \mid RSU(\beta, z) \geq minUtil\}$;
16: obtain *Secondary*$(\beta) = \{z \in$ *Secondary*$(\alpha) \mid RLU(\beta, z) \geq minUtil\}$;
17: call **search_P**(η, β, β-D, *Primary*(β), *Secondary*(β), *minUtil*);
18: **end for**

Algorithm 4 The *search_N* procedure

Input: η: a set of negative items, α: the current itemset, α-D: the current projected database, *Primary*(α): the *Primary* items of α, *Secondary*(α): the *Secondary* items of α, *minUtil*: a raised minimum utility threshold, and *top-k list*: a priority queue of k items.
Output: The set of top-k HUIs that are extensions of α with negative utility items.

1: **for** each item $z \in \eta$ **do**
2: $\beta = \alpha \cup \{z\}$;
3: scan α-D, calculate $U(\beta)$, and create β-D;
4: **if** $U(\beta) \geq$ *minUtil* **then**
5: add β into *top-k list*;
6: **if** $|$*top-k list*$| > k$ **then**
7: pop the k-th values in *top-k list*;
8: raise current *minUtil* with the k-th value;
9: **end if**
10: **end if**
11: calculate *RSU*(β, z) for all items $z \in \eta$ by scanning itemset β-D once, using the negative utility-array;
12: *Primary*$(\beta) = \{z \in \eta \mid RSU(\beta, z) \geq minUtil\}$;
13: call **search_N**(*Primary*(β), β, β-D, *minUtil*);
14: **end for**

database β-D is created. Lines 4-10 show that if the utility value of β is higher than or equal to the current *minUtil*, β will be recognized as a HUI and added to the *top-k list*. Moreover, if the size of the *top-k list* is larger than k, it indicates that top-k HUIs already exist. In this case, the k-th HUI will be removed and the current *minUtil* will be changed. Lines 11-13 show that if the utility of β is also higher (not equal) than the changed *minUtil*, we will try to add negative utility items to verify whether it will still be HUIs, because after itemset β extended some negative utility items, its utility may be still higher than the current *minUtil*. Moreover, similar to **Algorithm 2**, *RLU* and *RSU* of itemset β are computed, where items $z \in$ *secondary*(α) (Line 14). In Lines 15 and 16, the *primary* and *secondary* sets of β are separately calculated. Finally, the algorithm is repeatedly executed with an extension of β using a depth-first search (Line 17) until it satisfies the threshold.

The **Algorithm 4** is called when the utility of items/itemsets is greater than *minUtil* (not equal). Many of the steps are the same as in **Algorithm 3**. The main difference is that positive or negative utility items are extended to single items. Each item z combines with α to form a new itemset β, where each item $z \in \eta$ (Line 2). In Line 3, the database α-D is scanned, the utility of extended itemset β is computed, and a new projected database β-D is constructed. Moreover, transaction merging technique is adopted in the database β-D construction process. Lines 4-10 consider whether the threshold is raised.

Subsequently, the *RLU* and *RSU* are calculated again for all negative utility items and a new *primary* set is obtained in Lines 11 and 12. Thereafter, the algorithm recursively calls itself until it does not discover all extensions with negative utility items that satisfy the threshold of *minUtil* (Line 13).

V. Performance Evaluation

In this section, we conducted several experiments to demonstrate the effectiveness and efficiency of the proposed TOPIC algorithm. We conducted the experiment on a computer with a 3.0 GHz Intel Core Processor with 16 GB main memory running on Windows 10 Home Edition (64-bit operating system). We used Java language to implement all the algorithms and compared the performance of TOPIC with TopHUI [31]. Most of the existing top-k HUIM algorithms do not consider the common real case with negative utility values except TopHUI. To the best of our knowledge, TopHUI is the most efficient algorithm for mining top-k HUIs with negative utilities.

A. Data Description and Experimental Setup

To analyze the proposed algorithm in different situations, we evaluated its performance on several benchmark datasets. All datasets were downloaded from the SPMF data mining library [45]. Table VIII summarizes the detailed characteristics of all the datasets. The *Mushroom* and *Chess* datasets are highly dense in nature. *Chess* is a dense dataset with long transactions and few items. Although *Mushroom* is also a dense dataset, it has moderately long transactions. Additionally, *Retail* is a sparse dataset with large items in each transaction. *Accidents* is a dense dataset and has the highest number of transactions, with each transaction having many items. *T10I4D100K* and *T40I10D100K* are both sparse datasets. *BMSPOS* is a dense dataset that was used to test the scalability of the proposed algorithm. All the experimental results of these benchmark datasets are separately presented in the following sections. The runtime consumption, memory cost, number of visited candidate itemsets, and scalability are described in subsections V-B, V-C, V-D, and V-E, respectively.

TABLE VIII
Dataset characteristics

Dataset	#Trans	#Items	#AvgLen	#Type
Mushroom	8,142	119	23.0	Dense
Chess	3,196	75	37.0	Dense
Accidents	340,183	468	33.8	Dense
T40I10D100K	100,000	942	39.6	Dense
T10I4D100K	100,000	870	10.1	Sparse
Retail	88,162	16,470	10.3	Sparse
BMSPOS	515,366	1,656	6.51	Sparse

We tested both TopHUI and TOPIC on all datasets by increasing k. The *minUtil* was initialized as 1, and we implemented the TopHUI according to the descriptions provided in this paper. We implemented four versions of TOPIC: one with a transaction merging strategy, one with a sub-tree pruning strategy, one implemented with both merging and sub-tree pruning strategy, and the last one is the base version without these two strategies. These versions were referred to as TOPIC$_{merge}$, TOPIC$_{sub-tree}$, TOPIC, and TOPIC$_{none}$, respectively. All the algorithms were used for the experimental evaluation of the proposed top-k HUI mining method.

B. Experiments on Runtime

First, we evaluated the execution time of the proposed algorithm. Fig. 1 shows the comparison of the runtime of all the algorithms on different datasets with varied k values. In these figures, the TOPIC algorithm performs better than the TopHUI algorithm in many cases. For example, when k is 2000 in the *Mushroom* dataset, TOPIC only requires approximately 2 seconds to complete the mining process, whereas TopHUI requires approximately 6 seconds. On the *Retail* dataset, TopHUI takes too long time to return the result when k is set to 10,000. In all the tested datasets (*Mushroom, Chess, Retail, T10I4D100K, Accidents, T40I10D100K*), the runtime trend between TOPIC and TopHUI becomes increasingly different as the k values increase. In most datasets, TOPIC usually has a narrower fluctuation margin of the exchange rate.

From Table IX, TOPIC performs particularly well on dense and moderately dense datasets. As the parameter K increases, the runtime cost of all the algorithms becomes increasingly higher. Their results in Figure 1 shows TopHUI raises faster, but TOPIC is raising smooth except TOPIC$_{none}$. Generally, TOPIC is approximately one to three orders of magnitude faster than TopHUI. For the *Retail, Chess, Mushroom* and *T10I4D100K* datasets, TOPIC is up to 3, 20, 4, and 2 times faster than the TopHUI algorithm.

The most important reason why TOPIC has an excellent performance in all the datasets is that it utilizes the *RSU* and *RLU* upper bounds, depending on the projected database. It can prune a larger part of the search space compared to the TopHUI algorithm, which uses different strategies. Therefore, the proposed algorithm uses only a few itemsets to find high utility itemsets. It also utilizes a transaction merging technique to replace some transactions (which have identical items) with one transaction, which significantly reduces the cost of dataset scanning.

C. Experiments on Memory Evaluation

In this subsection, the memory usage of all the tested algorithms is recorded and compared with the varying K parameter. Fig. 2 shows the detailed result, in which TOPIC clearly outperforms TopHUI on all the datasets. For example, in Fig. 2(b), *Chess* dataset reveals that no matter the strategies that TOPIC adopts, TOPHUI uses almost eight times more memory than TOPIC to complete data mining. Moreover, in *Chess* datasets, when K is more than 5000, TopHUI cannot obtain the correct result in regular time (approximately 3 h).

Table X shows more details. In *T10I4D100K*, TOPIC uses 3.0, 2.6, 2.9, 3.8, and 3.2 times less memory than TopHUI when the k parameter is 100, 500, 1000, 5000, and 10,000 respectively. The worst case is that TopHUI cannot obtain the correct results when K is 10,000 in all the tested datasets. It is also interesting that TOPIC has the same performance as TOPIC$_{subtree}$, and TOPIC$_{merge}$ has the same performance as TOPIC$_{none}$. This s because sub-tree pruning strategy plays

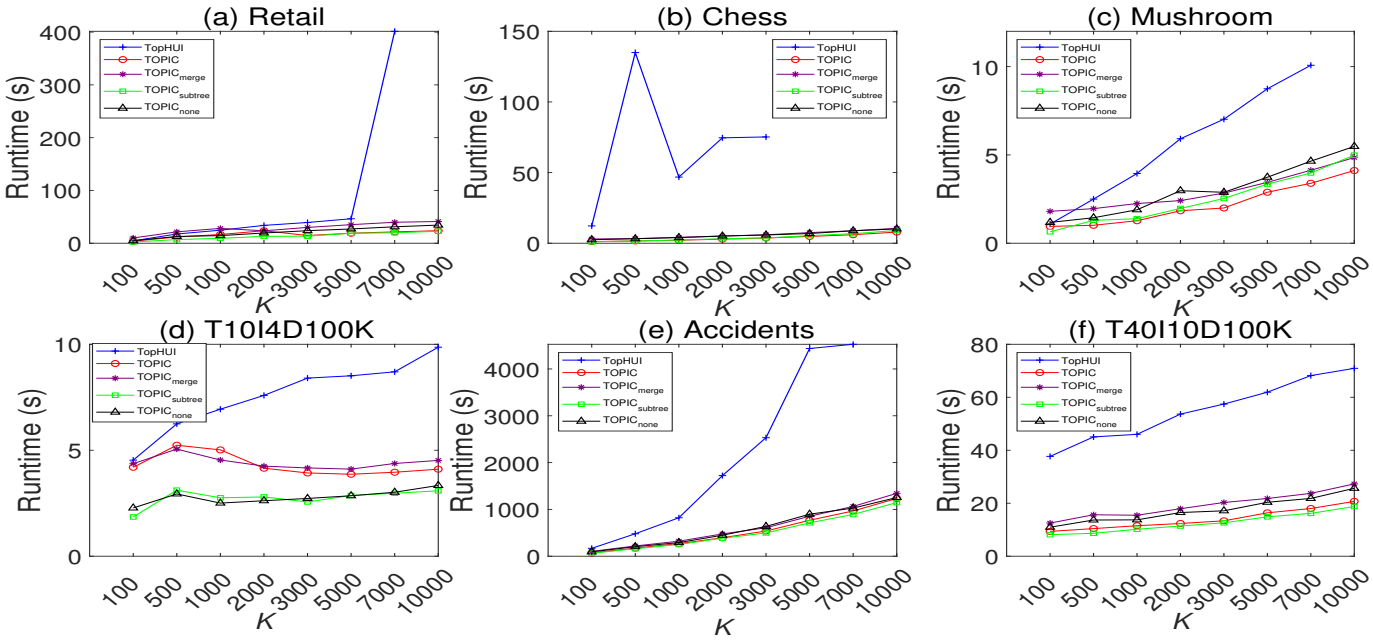


Fig. 1. Runtime cost under parameter (K).

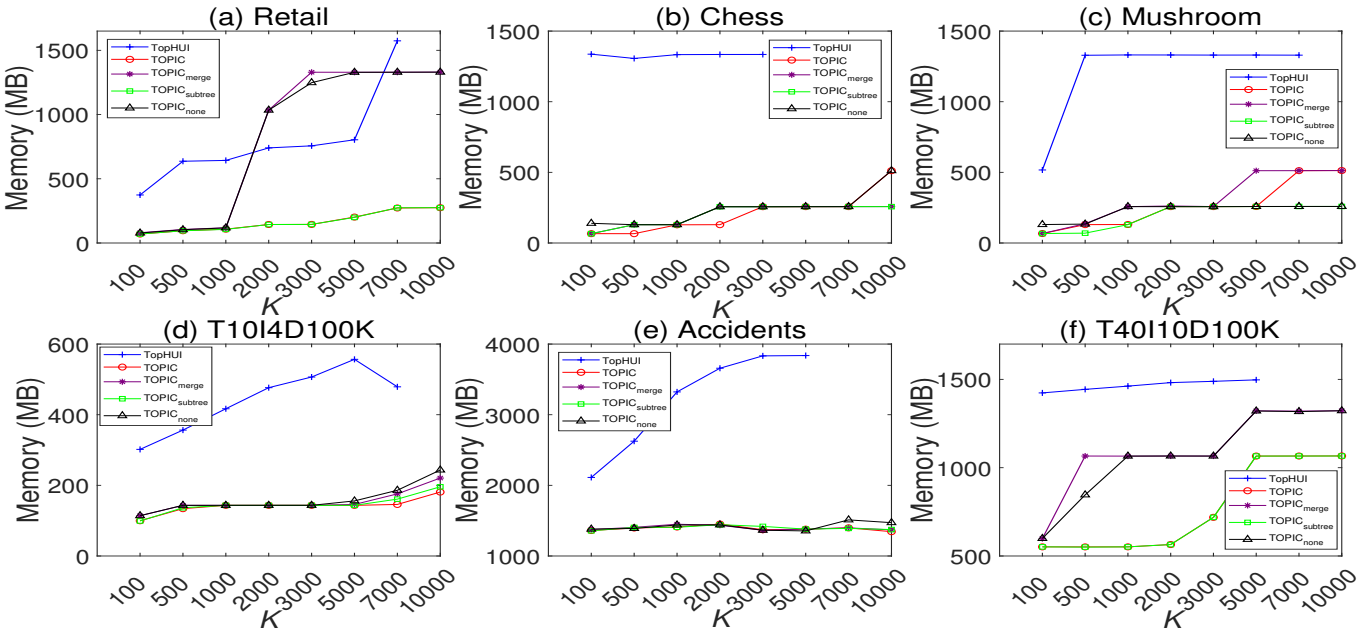


Fig. 2. Memory cost under parameter (K).

a vital role. While it iterates the searching space tree, if the *RTWU* value of an itemset is less than the current *minUtil*, subtree pruning strategy will remove it and its supersets. Because of the basis of the *RTWU*-based pruning, if an itemset X is not a HUI, all supersets of X would be low-utility itemsets.

Another reason why TOPIC performs quite efficiently is that it adopts a completely different data structure, which does not need to maintain a large amount of information in the memory. It only requires pointers for pseudo-projections to catch the pre-HUIs. However, TopHUI relies on a list structure to store additional information, which is more complex. It also calculates two utility upper bounds in linear time by arrays, which can be repeatedly used to count the upper bounds of each itemset while processing the depth-first search. These new upper bounds help to select extensible itemsets and potential HUIs to ignore unpromising itemsets.

D. Experiments on Candidates Analysis

We also compared the ability of the TOPIC and TopHUI algorithms to prune the search space. Table XI summarizes the

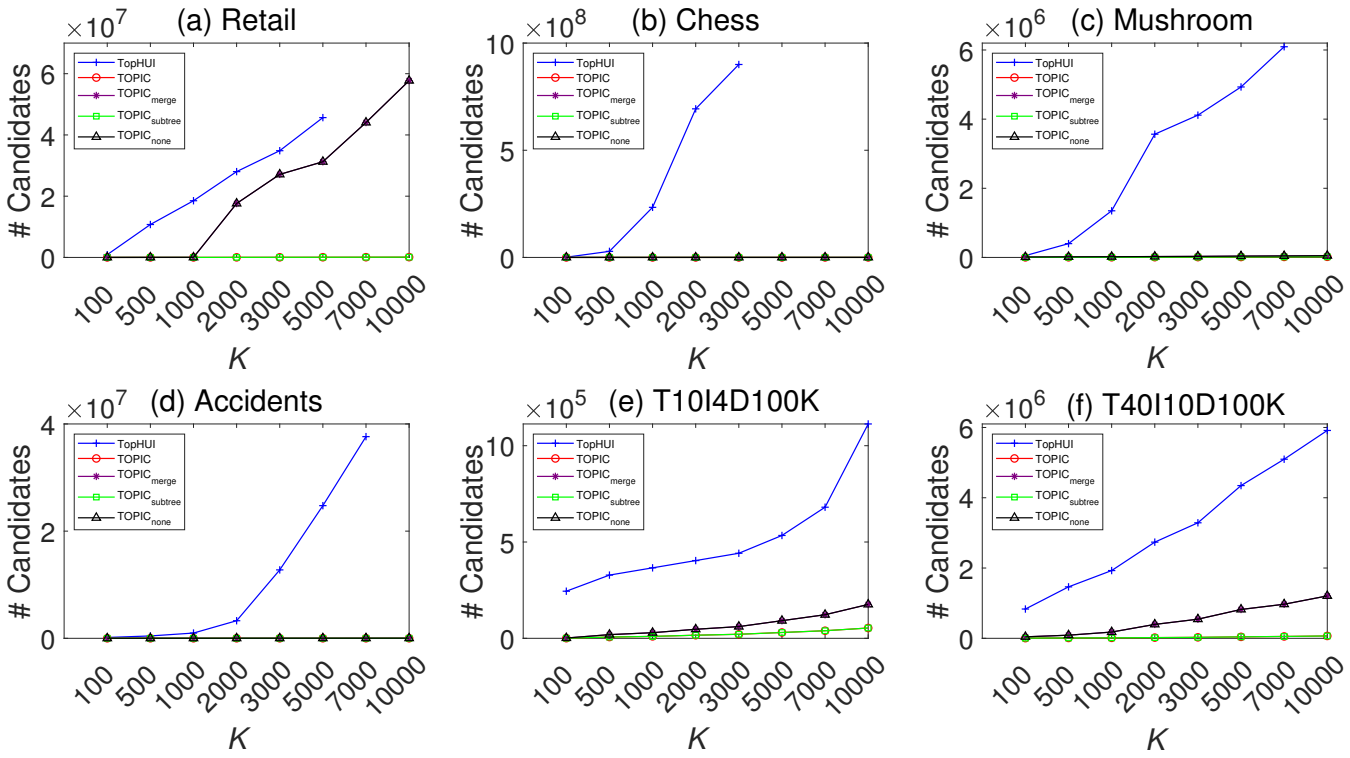


Fig. 3. Candidates under parameter (K).

TABLE IX
RUNTIME RESULT (SECONDS)

K	Algorithm	Retail	Chess	Mushroom	T10I4D100K	Accidents	T40I10D100K
100	TopHUI	4.545	12.269	1.082	4.533	168.629	37.684
	TOPIC	3.894	1.115	0.953	4.197	71.484	9.371
	TOPIC$_{merge}$	9.985	3.01	1.807	4.367	103.594	12.515
	TOPIC$_{subtree}$	2.411	0.991	0.647	1.852	63.977	8.14
	TOPIC$_{none}$	4.804	2.491	1.178	2.276	93.59	10.949
500	TopHUI	17.497	134.975	2.5	6.246	479.663	45.039
	TOPIC	11.818	1.614	1.016	5.237	173.129	10.429
	TOPIC$_{merge}$	21.583	3.261	1.959	5.054	218.826	15.653
	TOPIC$_{subtree}$	6.879	1.589	1.293	3.116	158.809	8.658
	TOPIC$_{none}$	12.83	3.192	1.443	2.938	203.958	13.666
1000	TopHUI	24.705	46.818	3.947	6.937	821.578	46.007
	TOPIC	16.557	2.117	1.274	5.017	268.598	11.508
	TOPIC$_{merge}$	28.104	4.148	2.247	4.54	321.059	15.467
	TOPIC$_{subtree}$	9.334	2.23	1.391	2.757	250.732	10.182
	TOPIC$_{none}$	14.416	3.995	1.894	2.508	290.812	13.727
5000	TopHUI	46.365	-	8.745	8.514	4,437.073	61.913
	TOPIC	18.967	4.818	2.888	3.868	767.916	16.38
	TOPIC$_{merge}$	35.193	7.506	3.447	4.111	853.639	21.797
	TOPIC$_{subtree}$	18.925	5.398	3.349	2.867	713.551	14.858
	TOPIC$_{none}$	27.036	6.888	3.737	2.853	897.455	20.361
10000	TopHUI	-	-	-	9.862	-	70.921
	TOPIC	23.744	7.892	4.117	4.107	1,233.659	20.703
	TOPIC$_{merge}$	41.208	10.512	4.85	4.521	1,343.106	27.303
	TOPIC$_{subtree}$	22.614	9.042	4.987	3.086	1,146.068	18.76
	TOPIC$_{none}$	34.354	10.156	5.482	3.339	1,253.623	25.641

results of TopHUI, TOPIC$_{merge}$, TOPIC$_{subtree}$, TOPIC$_{none}$, and TOPIC when K is 100, 500, 1000, 5000, and 10,000, respectively. It can be observed that TOPIC$_{merge}$, TOPIC$_{subtree}$, TOPIC$_{none}$, and TOPIC are more effective than TopHUI when pruning the search space. This is because the TOPIC algorithm adopts two special tight upper bounds (*RSU* and *RLU*). Upper bounds help to remove these low utility itemsets because they are irrelevant. In Table XI, each column shows that TopHUI generates beyond one to three orders of magnitude compared to TOPIC. Fig. 3 shows the rough trend of all outputs of the

TABLE X
MEMORY COST (MB)

K	Algorithm	Retail	Chess	Mushroom	T10I4D100K	Accidents	T40I10D100K
	TopHUI	373.95	1,336.48	516.9	301.8	2112	1,423.43
	TOPIC	69.63	66.09	66.91	99.8	1,360.61	551.7
100	TOPIC$_{merge}$	80.04	65.97	67.28	114.07	1,373.68	599.09
	TOPIC$_{subtree}$	69.64	65.95	66.97	99.12	1,356.82	551.73
	TOPIC$_{none}$	77.63	139.4	130.54	114.06	1,380.66	599.33
	TopHUI	636.49	1,306.25	1,328.7	356.42	2,624.92	1,443.47
	TOPIC	95.45	66.02	130.7	134.01	1,396.46	551.45
500	TOPIC$_{merge}$	105.38	129.57	136.55	143.39	1,405.88	1,065.99
	TOPIC$_{subtree}$	95.33	129.55	69.68	136.48	1,404.48	550.33
	TOPIC$_{none}$	103.39	129.51	133.79	143.39	1,390.12	845.97
	TopHUI	642.96	1,333.29	1,331.07	416.84	3,322.79	1,461.81
	TOPIC	107.7	128.72	130.81	143.38	1,410.31	551.74
1000	TOPIC$_{merge}$	118.71	129.51	257.3	143.39	1,449.51	1,065.49
	TOPIC$_{subtree}$	106.1	129.59	129.77	143.38	1,406.9	551.69
	TOPIC$_{none}$	117.69	129.59	257.38	143.38	1,439.16	1,065.58
	TopHUI	803.66	-	1,330.19	556.73	3,837.37	1,497.28
	TOPIC	201.84	257.42	258.2	143.38	1,379.98	1,065.81
5000	TOPIC$_{merge}$	1,329.31	256.93	511.73	145.87	1,381.22	1,321.86
	TOPIC$_{subtree}$	199.38	257.35	257.81	143.39	1,380.56	1,065.74
	TOPIC$_{none}$	1,328.95	257.3	258.33	155.92	1,354.89	1,321
	TopHUI	-	-	-	572.66	-	1,502.87
	TOPIC	275.03	511.8	512.94	181.01	1,344.19	1,066.54
10000	TOPIC$_{merge}$	1,330.7	257.98	512.7	220.63	1,369.76	1,322.49
	TOPIC$_{subtree}$	275.03	257.21	258.64	195.88	1,373.93	1,066.19
	TOPIC$_{none}$	1,330.12	511.83	258.61	243.4	1,471.99	1,322.48

TABLE XI
CANDIDATES GENERATION

K	Algorithm	Retail	Chess	Mushroom	T10I4D100K	Accidents	T40I10D100K
	TopHUI	902,981	1,095,506	52,751	244,575	158,950	83,360
	TOPIC	1,105	7,578	1,822	551	1,438	4,759
100	TOPIC$_{merge}$	2,949	63,098	10,686	1,438	4,137	42,021
	TOPIC$_{subtree}$	1,105	7,578	1,822	551	1,438	4,759
	TOPIC$_{none}$	2,949	63,098	10,686	1,438	4,137	42,021
	TopHUI	10,747,653	28,358,203	399,129	328,210	441,707	1,463,994
	TOPIC	5,044	11,132	3,533	6,543	2,385	9,774
500	TOPIC$_{merge}$	11,311	83,155	16,479	19,111	6,614	88,898
	TOPIC$_{subtree}$	5,044	11,132	3,533	6,543	2,385	9,774
	TOPIC$_{none}$	11,311	83,155	16,479	19,111	6,614	88,898
	TopHUI	18,518,220	3,522,998	1,348,989	365,864	981,799	1,928,124
	TOPIC	9,661	13,521	4,832	9,981	2,923	13,118
1000	TOPIC$_{merge}$	22,454	96,391	20,236	29,096	8,229	177,968
	TOPIC$_{subtree}$	9,661	13,521	4,832	9,981	2,923	13,118
	TOPIC$_{none}$	22,454	96,391	20,236	29,096	8,299	177,968
	TopHUI	45,661,581	-	493,057	533,965	24,758,053	4,345,977
	TOPIC	60,025	22,067	12,313	30,181	5,101	41,525
5000	TOPIC$_{merge}$	31,247,162	135,472	37,989	91,556	14,428	820,906
	TOPIC$_{subtree}$	60,025	22,067	12,313	30,181	5,101	41,525
	TOPIC$_{none}$	31,247,162	135,472	37,989	91,556	14,428	820,906
	TopHUI	-	-	-	1,112,932	-	5,914,611
	TOPIC	93,323	27,598	18,146	53,488	6,865	66,951
10000	TOPIC$_{merge}$	57,730,102	158,524	52,090	175,800	18,736	1,206,662
	TOPIC$_{subtree}$	93,323	27,598	18,146	53,488	6,865	66,951
	TOPIC$_{none}$	57,730,102	158,524	52,090	175,800	18,736	1,206,662

compared algorithms.

E. Experiments on Scalability Test

Finally, we tested the scalability of TOPIC. We varied the size of the *BMSPOS* dataset from 20% (= 103,073 transactions) to 100% (= 515,366 transactions), and compared the execution time and memory consumption. We set the value of K to 10,000 to check the scalability performance of four variants, such as TOPIC, TOPIC$_{merge}$, TOPIC$_{subtree}$, and TOPIC$_{none}$. Figs. 4 and 5 separately show that the runtime and memory cost increase linearly with increased dataset size. In particular, both the runtime and memory consumption of TOPIC performed better than others. Thus, TOPIC has suitable scalability for large-scale datasets.

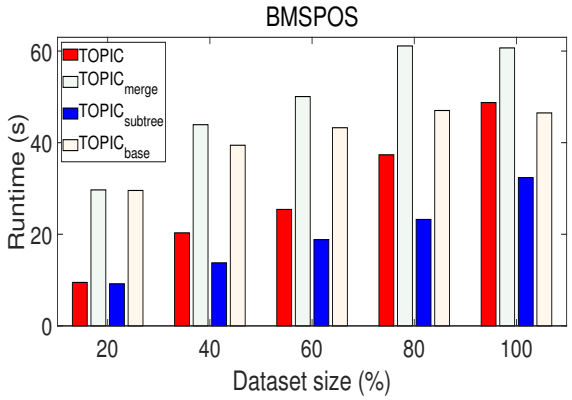


Fig. 4. Runtime scalability of algorithms on *BMSPOS*.

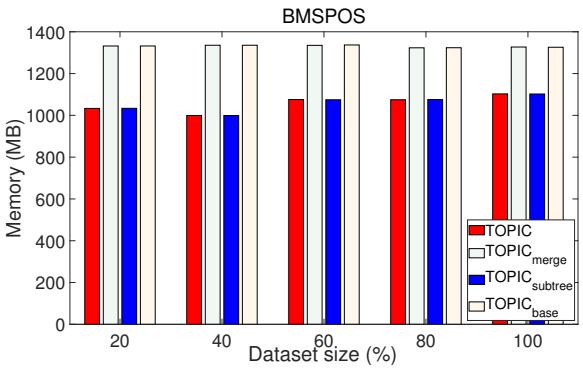


Fig. 5. Memory scalability of algorithms on *BMSPOS*.

VI. Conclusion and Future Work

In this work, top-k HUI mining with negative utility was proposed. Our proposed algorithm, TOPIC, adopts two new upper bounds called redefined local utility and redefined subtree utility to quickly prune the search space. In addition, we utilized novel utility arrays to efficiently calculate these upper bounds. To reduce the costs of dataset scanning and memory, we adopted dataset projection and transaction merging techniques. Without setting threshold, *minUtil* threshold auto-raising strategy was utilized. Compared with state-of-the-art algorithms, the results show that TOPIC has a significantly improved runtime performance on real and synthetic datasets. Moreover, the memory consumption of TOPIC on all datasets was excellent.

In the future, we will improve the threshold auto-raising strategy and design more compressed data structures. The proposed idea can also be used in the field of on-shelf utility mining, incremental mining of HUIs, and mining of top-k HUIs from data streams or sequential datasets.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th ACM International Conference on Very Large Data Bases*, vol. 1215. Citeseer, 1994, pp. 487–499.

[2] P. Fournier Viger, J. C. W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 4, p. e1207, 2017.

[3] W. Gan, J. C. W. Lin, H. C. Chao, and J. Zhan, "Data mining in distributed environment: a survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 6, p. e1216, 2017.

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

[5] W. Gan, C. W. Lin, P. Fournier Viger, H. C. Chao, V. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1306–1327, 2021.

[6] Y. Liu, W. 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.

[7] V. S. Tseng, C. 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*, 2010, pp. 253–262.

[8] V. S. Tseng, B. E. Shie, C. W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772–1786, 2012.

[9] U. Yun, H. Ryang, and K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3861–3878, 2014.

[10] K. Singh, H. K. Shakya, A. Singh, and B. Biswas, "Mining of high-utility itemsets with negative utility," *Expert Systems*, vol. 35, no. 6, p. e12296, 2018.

[11] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. k. Lee, "HUC-Prune: An efficient candidate pruning technique to mine high utility patterns," *Applied Intelligence*, vol. 34, no. 2, pp. 181–198, 2011.

[12] 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*, 2012, pp. 55–64.

[13] G. Lan, T. P. Hong, and V. S. Tseng, "An efficient projection-based indexing approach for mining high utility itemsets," *Knowledge and Information Systems*, vol. 38, no. 1, pp. 85–107, 2014.

[14] B. E. Shie, P. S. Yu, and V. S. Tseng, "Mining interesting user behavior patterns in mobile commerce environments," *Applied Intelligence*, vol. 38, no. 3, pp. 418–435, 2013.

[15] C. Chu, V. S. Tseng, and T. Liang, "An efficient algorithm for mining temporal high utility itemsets from data streams," *Journal of Systems and Software*, vol. 81, no. 7, pp. 1105–1117, 2008.

[16] S. J. Yen and Y. Lee, "Mining high utility quantitative association rules," in *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2007, pp. 283–292.

[17] 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.

[18] K. Subramanian and P. Kandhasamy, "UP-GNIV: An expeditious high utility pattern mining algorithm for itemsets with negative utility values," *International Journal of Information Technology and Management*, vol. 14, no. 1, pp. 26–42, 2015.

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

[20] ——, "HMiner: Efficiently mining high utility itemsets," *Expert Systems With Applications*, vol. 90, pp. 168–183, 2017.

[21] P. Fournier Viger, C. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," *Foundations of Intelligent Systems*, pp. 83–92, 2014.

[22] S. Zida, P. Fournier Viger, J. C. W. Lin, C. Wu, and V. S. Tseng, "EFIM: a fast and memory efficient algorithm for high-utility itemset mining," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 595–625, 2017.

[23] J. Liu, K. Wang, and B. C. M. Fung, "d2HUP: Mining high utility patterns in one phase without generating candidates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1245–1257, 2015.

[24] 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.

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

[26] 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.

[27] V. S. Tseng, C. 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.

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

[29] 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.

[30] 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.

[31] W. Gan, S. Wan, J. Chen, C. M. Chen, and L. Qiu, "TopHUI: Top-k high-utility itemset mining with negative utility," in *IEEE International Conference on Big Data*. IEEE, 2020, pp. 5350–5359.

[32] H. Yao, H. J. Hamilton, and L. Q. Geng, "A unified framework for utility-based measures for mining itemsets," in *Proceedings of ACM SIGKDD 2nd Workshop on Utility Based Data Mining*. Citeseer, 2006, pp. 28–37.

[33] W. Song, Y. Liu, and J. Li, "BAHUI: Fast and memory efficient mining of high utility itemsets based on bitmap," *International Journal of Data Warehousing and Mining*, vol. 10, no. 1, pp. 1–15, 2014.

[34] W. Gan, J. C. W. Lin, P. Fournier Viger, H. C. Chao, T. P. Hong, and H. Fujita, "A survey of incremental high-utility itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1242, 2018.

[35] V. S. Tseng, C. Wu, P. Fournier Viger, and P. S. Yu, "Efficient algorithms for mining the concise and lossless representation of high utility itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 726–739, 2014.

[36] L. T. Nguyen, V. V. Vu, M. T. Lam, T. T. Duong, L. T. Manh, T. T. Nguyen, B. Vo, and H. Fujita, "An efficient method for mining high utility closed itemsets," *Information Sciences*, vol. 495, pp. 78–99, 2019.

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

[38] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, "Efficient algorithms for mining high-utility itemsets in uncertain databases," *Knowledge-Based Systems*, vol. 96, pp. 171–187, 2016.

[39] ——, "Efficiently mining uncertain high-utility itemsets," *Soft Computing*, vol. 21, no. 11, pp. 2801–2820, 2017.

[40] T. Mai, B. Vo, and L. T. Nguyen, "A lattice-based approach for mining high utility association rules," *Information Sciences*, vol. 399, pp. 81–97, 2017.

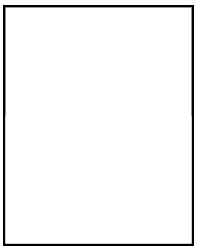
[41] W. Gan, J. C. W. Lin, P. Fournier Viger, H. C. Chao, and V. S. Tseng, "Mining high-utility itemsets with both positive and negative unit profits from uncertain databases," in *Advances in Knowledge Discovery and Data Mining*. Springer International Publishing, 2017, pp. 434–446.

[42] C. 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.

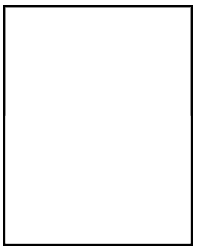
[43] Y. C. Li, J. S. Yeh, and C. C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data & Knowledge Engineering*, vol. 64, no. 1, pp. 198–217, 2008.

[44] W. Gan, J. C. W. Lin, H. C. Chao, A. V. Vasilakos, and P. S. Yu, "Utility-driven data analytics on uncertain data," *IEEE Systems Journal*, vol. 14, no. 3, pp. 4442–4453, 2020.

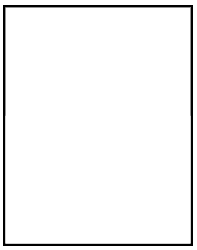
[45] P. Fournier Viger, J. C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The spmf open source data mining library version 2," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2016, pp. 36–40.



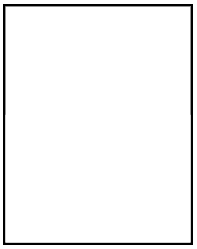
Jiahui Chen (Member, IEEE) received the BS degree from South China Normal University, China in 2009, and MS and PhD degrees from South China University of Technology, China, in 2012 and 2016, respectively. He joined National University of Singapore as a research scientist between form 2017 to 2018. He is currently an associate professor in the School of Computer Sciences, Guangdong University of Technology, China. His research interests mainly focus on public key cryptography, post-quantum cryptography, and information security.



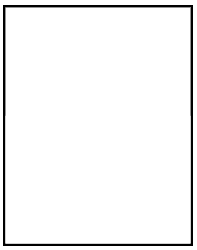
Shicheng Wan received the B.S. degree in Gannan Normal University, Jiangxi, China in 2020. He is currently a postgraduate in the School of Computer Sciences, Guangdong University of Technology, China. His research interests include data mining, utility mining, and big data.



Wensheng Gan (Member, IEEE) received the B.S. degree in Computer Science from South China Normal University, China in 2013. He received the Ph.D. in Computer Science and Technology, Harbin Institute of Technology (Shenzhen), China in 2019. He was a joint Ph.D. student with the University of Illinois at Chicago, Chicago, USA, from 2017 to 2019. He is currently an Association Professor with the College of Cyber Security, Jinan University, Guangzhou, China. His research interests include data mining, utility computing, and big data analytics. He has published more than 80 research papers in peer-reviewed journals (i.e., IEEE TKDE, IEEE TCYB, ACM TKDD, ACM TOIT, ACM TMIS) and international conferences. He is an Associate Editor of *Journal of Internet Technology*.



Guoting Chen is currently a full professor with School of Science, Harbin Institute of Technology, Shenzhen. He received B.S., M.S. and Ph.D. degrees in Mathematics from Wuhan University, China in 1982, from Wuhan University, China in 1985, and from University de Grenoble 1, France in 1990, respectively. His research interests include Mathematics, differential equations, and data science. He has published 30 peer-reviewed research papers.



Hamido Fujita (Senior Member, IEEE) is currently a Professor with Iwate Prefectural University, Takizawa, Japan, as Director of Intelligent Software Systems. He received Doctor Honoris Causa from Óbuda University, Budapest, Hungary, in 2013 and received Doctor Honoris Causa from Timisoara Technical University, Timisoara, Romania, in 2018, and a title of Honorary Professor from Óbuda University, in 2011. He is the Emeritus Editor-in-Chief for Knowledge-Based Systems, and currently Editor-in-Chief of Applied Intelligence (Springer), He is the Vice President of International Society of Applied Intelligence. He headed a number of projects including intelligent HCI, a project related to mental cloning for healthcare systems as an intelligent user interface between human-users and computers, and SCOPE project on virtual doctor systems for medical applications. He has published more 400 highly cited Papers.