Targeted High-Utility Itemset Querying

Jinbao Miao, Shicheng Wan, Wensheng Gan, *Member, IEEE*, Jiayi Sun, and Jiahui Chen, *Member, IEEE*

***Abstract*—Traditional high-utility itemset mining (HUIM) aims to determine all high-utility itemsets (HUIs) that satisfy the minimum utility threshold in transaction databases. However, in most applications, not all HUIs are interesting because only specific parts are required. Thus, targeted mining based on user preferences is more important than traditional mining tasks. This article is the first to propose a target-based HUIM problem and to provide a clear formulation of the targeted utility mining task in a quantitative transaction database. A tree-based algorithm known as Target-based high-Utility iteMset querying (TargetUM) is proposed. The algorithm uses a lexicographic querying tree and three effective pruning strategies to improve the mining efficiency. We implemented experimental validation on several real and synthetic databases, and the results demonstrate that the performance of TargetUM is satisfactory, complete, and correct. Finally, owing to the lexicographic querying tree, the database no longer needs to be scanned repeatedly for multiple queries.**

***Impact Statement*—This article contributes to a utility-based targeted pattern discovery model for artificial intelligence and data science. To the best of our knowledge, it is the first article that proposes a realistic utility-based solution for the targeted pattern discovery instead of all pattern discovery from a real-world dataset. The designed TargetUM method can be a benchmark of target-based utility mining. The proposed method addresses several challenges and achieves state-of-the-art performance on massive datasets. TargetUM can provide acceptable querying performance. This targeted utility mining problem formulation and efficient algorithm contribute to the artificial intelligence systems in many applications, such as market basket analysis, risk prediction, smart retail, intrusion detection, and so on.**

***Index Terms*—Data mining, target high-utility itemset (THUI), target pattern, utility mining.**

I. Introduction

With the rapid development of information technology and database management systems, extremely large amounts of raw data are produced daily. Massive useful messages are hidden in these Big Data. The discovery of valuable information and its utilization has emerged as an important topic. For example, customers usually buy keyboards and mouses after purchasing computers. Following this, other potential requirements exist, such as a pair of headphones and a suitable chair. However, customers may not gather these items into baskets simultaneously. Thus, we cannot intuitively observe the relationship among these goods. If the purchase records are analyzed according to data mining technologies [1], the results will aid in understanding the knowledge that is hidden in rich data. For example, clerks can place headphones or chairs between computers and keyboards to improve the sale volume. In the past several decades, researchers have proposed hundreds of data mining algorithms that rely on different data formats in many applications and various domains. Among these, one traditional data mining technology is known as frequent pattern mining (FPM) [2], [3], and the Apriori algorithm [2] is the most famous and earliest algorithm. Apriori discovers interesting patterns based on frequency and confidence metrics. Since then, many other algorithms [4]–[6] have been proposed for mining interesting patterns.

However, FPM algorithms exhibit a fatal flaw in that their results may be frequent but result in low profit. The frequency of a pattern may not be a sufficient indicator of interest, as it only reflects the number of transactions that contain the pattern in a database. In fact, other important factors (e.g., weight, unit profit, and risk) often need to be considered in real-life applications [7]. As a simple example, diamonds definitely have a lower sale volume than pencils. If only the frequency metric is considered, diamonds exhibit an unpromising pattern owing to their low support. Thus, they are apparently not a wise suggestion for retailers. To address this problem, inspired by economic knowledge, scholars have proposed a novel concept known as utility, and data mining technologies based on utility metrics have been presented in the form of utility-oriented pattern mining (UPM) [7]. Previous studies [7]–[9] have developed a unified definition of high-utility pattern (HUP) as consisting of two components: external utility and internal utility. Each pattern owns a unit external utility (e.g., unit profit, risk, priority, or weight) and appears once or more in each transaction (aka internal utility, e.g., purchase quantity). Therefore, the utility of each pattern can be analyzed in a reasonable way. The UPM field has been developing for more than 20 years, and many algorithms have been designed to make the mining process more efficient, such as list-based algorithms (e.g., HUI-Miner [9], FHM [10], and FHN [11]), tree-based algorithms (e.g., IHUP [12], UP-Growth [8], and MU-Growth [13]), projection-based algorithms (e.g., EFIM [14]), and other algorithms. In UPM, a pattern is referred to as an HUP if its utility value is no less than a

Manuscript received 4 November 2021; revised 12 February 2022; accepted 16 April 2022. Date of publication 29 April 2022; date of current version 21 July 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62002136 and Grant 61902079, in part by the Natural Science Foundation of Guangdong Province under Grant 2022A1515011861, in part by the Guangzhou Basic and Applied Basic Research Foundation under Grant 202102020277 and Grant 202102020928. This paper was recommended for publication by Associate Editor Latifur Khan upon evaluation of the reviewers' comments. *(Corresponding author: Wensheng Gan.)*

Jinbao Miao, Wensheng Gan, and Jiayi Sun are with the College of Cyber Security, Jinan University, Guangzhou 510632, China (e-mail: osjbmiao@gmail.com; wsgan001@gmail.com; jiayisun01@gmail.com).

Shicheng Wan and Jiahui Chen are with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China (e-mail: cwan1998@gmail.com; csjhchen@gmail.com).

Digital Object Identifier 10.1109/TAI.2022.3171530

user-specified minimum utility threshold (abbreviated as *minUtil*, denoted as σ). In general, UPM algorithms are more complex than FPM. For example, smartphones and headphones can both generate high profits. However, most people buy these items separately instead of together because of their high prices. This indicates that selling a combination of goods may result in less profit than when selling them separately. However, if the goods are a computer and keyboard, the result will be the opposite. Therefore, utility is neither monotonic nor antimonotonic, such as frequency. To address this limitation, Liu *et al.* [15] proposed an overestimated concept known as transaction-weighted utilization (abbreviated as TWU), which has a downward closure property (i.e., antimonotonic) [2]. This means that if a subset of a pattern is less than the threshold, all supersets thereof will not meet the condition either. Subsequently, all UPM algorithms adopt the TWU concept, optimized data storage structures, and effective pruning strategies.

Nevertheless, most of the above algorithms cannot deal with target-oriented mining tasks. The users input a *minUtil* threshold, following which the algorithms return all patterns with utility values that are higher than the threshold. That is, traditional UPM technologies aim to offer wide suggestions; they do not allow users to perform targeted queries. For example, shareholders simply wish to know when their own stocks will rise to the expected price, so that they can sell stocks in time, merchants often consider how to sell their stockpiled goods as soon as possible, and customers usually have a certain shopping list before going to the supermarket, which helps them to save time when purchasing goods. Thus, users are usually not really interested in obtaining the overall information. Researchers have defined this interesting task as target-oriented pattern mining [16]. To date, several studies have been conducted on target-based pattern mining, such as target-oriented frequent itemset querying [17], target-based association rule mining [18], [19], and targeted sequential pattern querying [20]–[22]. As stated previously, customers are usually not interested in receiving all discount messages according to the recommender system. More precisely, they may prefer to gain knowledge on specific goods they need and whether these are on sale. Until recently, there has been a lack of knowledge and efficient technologies for target-based pattern mining. In this article, we first formulate a new problem, known as target-based high-utility itemset mining (THUIM), to apply querying technology more extensively. In contrast to existing target-based algorithms, the novel task of targeted utility-oriented itemset querying focuses on extracting not only the results containing the predefined target but also those with high utility values with respect to a specified σ.

It is clear that utility-driven pattern mining is a hot topic, while there is no research on targeted utility mining. Interestingly, several specific challenges of THUIM need to be addressed. First, the utility does not naturally follow the antimonotonic property, indicating that frequency-based approaches cannot be adopted directly. Second, traditional FPM and UPM algorithms generate numerous unpromising candidates. Therefore, it is difficult to discover targeted queries without tight upper bounds of utility, particularly in large-scale databases. Third, effectively saving on memory is an inescapable challenge. Thus, the design of a compact data structure is also required. Finally, there is an urgent requirement for algorithms that can process massive databases efficiently and provide acceptable querying performance.

To this end, we propose the novel **Target**-based high-**U**tility ite**M**set querying algorithm (abbreviated as TargetUM). The main contributions of our study are as follows.

1) To the best of our knowledge, this is the first algorithm that incorporates the concept of utility into target-based utility mining. We introduce several key definitions and formulate the problem of discovering the desired set of utility-driven target queries.
2) We adopt the utility-list structure to generate high-utility itemsets (HUIs) directly, which can prevent the production of redundant candidates. Furthermore, we propose a utility-based trie tree to query target itemsets conveniently and to improve the mining efficiency.
3) For further improvement in the efficiency, we use several upper bounds and pruning strategies to accelerate the calculation process.
4) We conduct experiments on several datasets to demonstrate the effectiveness and efficiency of TargetUM for mining all desired queries with a user specified *minUtil*.

Note that some key concepts and an initial algorithm were presented in a preliminary version [23] of this article.

The rest of this article is organized as follows. Related work is presented in Section II. Thereafter, key preliminaries are described in Section III. Section IV presents the details of the designed TargetUM algorithm. The experiments and a comprehensive analysis are outlined in Section V. Finally, the Section VI concludes this article.

II. Related Work

In this section, we briefly review several studies on traditional frequent itemset mining (FIM), HUIM, and target pattern querying.

A. Frequent Itemset Mining

Given a transaction database, FIM aims to identify the itemsets that appear frequently. FIM and its extended technologies have been applied extensively in numerous real-life domains [1]–[3], [24]. Retailers use frequent itemsets to promote frequently purchased goods in market basket analysis. The most representative work is Apriori [2], which is a levelwise-based algorithm. Although its drawback is obvious, namely that it generates too many candidates and has to scan the database repeatedly, it has been possible to derive many Apriori-like algorithms [15], [25]. Subsequently, FP-Growth [3] established a new tree-based algorithm. According to the FP-tree data structure, the database only needs to be scanned twice and a highly compact data tree is constructed. The FP-tree stores all key information regarding frequent super-itemsets and uses a head-table to record the frequent items. Therefore, FP-Growth can conveniently obtain all interesting itemsets, and it requires fewer resources than Apriori. In general, numerous investigators have improved FIM algorithms in recent decades [5], [24]–[27]. However, as explained in Section I, most FIM algorithms do

not consider information regarding the unit interesting factor of items. Thus, many frequent itemsets with low interest will be identified; for example, many rare itemsets with high profits may be discarded. To address this issue, UPM [7] has attracted considerable attention. In a sense, utility refers to the profit/risk that an item can bring.

B. High-Utility Itemset Mining

HUIM is a subfield of UPM. Since the first HUIM algorithm was proposed, HUIM technologies have been used extensively in many practical applications, such as user behavior analysis [28], website click-stream analysis [29], and cross-marketing analysis [30]. Two distinct types of algorithms exist in the HUIM field (two-phase and single-phase models). In the well-known two-phase model [15], the mining process (referred to as Phase I) selects HUIs after computing all of the candidates (referred to as Phase II). Two-phase incorporates an upper bound known as TWU, which has a downward closure property, meaning that super-itemsets of an itemset cannot be HUIs if their TWU values are less than *minUtil*. Other HUIM algorithms include TWU to prune unpromising items/itemsets easily, whereas these two-phase models compute HUIs that need to scan the database at least twice. Hence, IHUP [12] incorporates a compact searching tree similar to an FP-tree [3]. The new data structure saves more runtime and memory compared to two-phase. However, IHUP uses an inaccurate upper bound so that many unpromising super-itemsets are retained. UP-Growth [31] and UP-Growth+ [8] successfully solved this issue by designing a complete search tree structure to discover all real HUIs. In general, a tree data structure can avoid the database being scanned multiple times. However, when dealing with huge amounts of data, the tree will be more complex and will run out of memory. Liu *et al.* [9] proposed a breakthrough algorithm known as HUI-Miner, which is classified as a single-phase model. Compared to two-phase algorithms, the single-phase model calculates HUIs without generating candidates. HUI-Miner uses a novel data structure known as a utility-list, which can avoid the problem of numerous candidates being generated.

Subsequently, other list-based algorithms that perform better than HUI-Miner have been proposed. For example, FHM [10] uses a new list structure known as the estimated utility co-occurrence structure to reduce the expense of intersection/join operations; FHN [11] is an improved version of FHM that can mine HUIs with negative utility values; LMHAUP adopts suggested list-based structure [32]; the ULB-Miner algorithm [33] uses the utility-list buffer to reduce the memory consumption and accelerate the join process; and the state-of-the-art TopHUI [34] addresses the top-k utility mining problem without setting *minUtil*. Other new data structures and mining algorithms have also been proposed, such as HUOPM [35], TKO [36], KHMC [37], IHUI-Mine based on subsume index [38], EFIM [14], and IMHUP [39]. There are also many studies of HUIM about different interesting effectiveness and efficiency issues [40]–[45]. Further details regarding UPM can be found in [7], [46], and [47].

C. Target Pattern Querying

Most pattern mining algorithms are designed to discover the complete itemsets in an entire database. In reality, not all of these patterns are interesting. However, users focus only on a part of the mining results. Therefore, target pattern querying [16], which is similar to an interactive query method, was introduced. In this method, users offer the target set and the system searches for related objects based on the user input. Users can repeatedly query their target patterns from the mining results. Several target-oriented querying approaches have been proposed. In 2003, Kubat *et al.* [16] introduced the target-oriented querying task and designed an approach to handle the specialized queries of users in a transaction database. They designed itemset-tree (IT), which differs from the FP-tree structure. The insertion of new transactions to update nodes incrementally provides an efficient querying function. However, the IT data structure consumes a large amount of memory owing to its poor scalability. Fournier-Viger *et al.* [19] designed an improved data structure known as the memory-efficient itemset tree (MEIT) to address this issue. The new structure uses a node compression mechanism to reduce the size of the IT node efficiently. The experiments demonstrated that the new data structure is up to 43% smaller than IT on average in terms of memory consumption. As an FP tree based algorithm, guided FP-growth [17] was proposed. It can determine the support of a given large list of itemsets (that are regarded as targets) using the target IT. Several target-based approaches have also been developed to handle sequence data based on target query constraints. Chueh *et al.* [21] presented an algorithm that uses time intervals to reverse the original sequences and to determine target sequential patterns. Chand *et al.* [20] extended targeted sequential pattern mining with monetary constraints. However, the aforementioned sequential mining algorithms focus on frequency metrics. The TUSQ algorithm [22] is the first to incorporate the utility concept into target pattern querying. Its motivation is that utility-driven sequential pattern mining algorithms often obtain several useless patterns owing to exhaustive results. At present, mining algorithms with target-querying constraints have been applied in various applications [18], [48].

To date, no research has integrated utility with target itemset querying in quantitative databases. The existing HUIM algorithms return a large number of HUIs, which may waste time for users in discovering the targeted special results. This motivated us to design an efficient target-querying approach for mining HUIs from quantitative transaction databases.

III. Preliminaries and Problem Statement

In this section, we first briefly introduce the basic concepts relating to HUIM. Most notations and definitions are provided in [9] and [11], and we formulate a new problem statement and the parameter requirements thereof.

A. Basic Preliminaries

As defined in previous works, $I = \{i_1, i_2, i_3, \ldots, i_n\}$ is a set of n distinct items in a transaction database ($\mathcal{D}$). $\mathcal{D}$ is mainly

TABLE I
SIMPLE EXAMPLE

(a) Quantitative transaction database

TID	Transaction	Quantity
T_1	$\{b, c, e, g, h\}$	$\{1, 2, 2, 1, 1\}$
T_2	$\{a, c, f, g\}$	$\{2, 1, 2, 4\}$
T_3	$\{b, c, e, h\}$	$\{5, 2, 3, 4\}$
T_4	$\{a, c, d, e, g\}$	$\{2, 1, 3, 1, 2\}$
T_5	$\{c, e\}$	$\{3, 4\}$
T_6	$\{g, h\}$	$\{1, 1\}$

(b) Utility of single items

Item	a	b	c	d	e	f	g	h
Utility	\$3	\$2	\$1	\$5	\$4	\$2	\$2	\$1

TABLE II
TRANSACTION UTILITY

TID	T_1	T_2	T_3	T_4	T_5	T_6
TU	\$15	\$19	\$28	\$30	\$19	\$3

TABLE III
TRANSACTION-WEIGHTED UTILITY

(a) Original alphabetical order

item	a	b	c	d	e	f	g	h
TWU	\$49	\$43	\$111	\$30	\$92	\$19	\$67	\$46

(b) TWU-ascending order

item	f	d	b	h	a	g	e	c
TWU	\$19	\$30	\$43	\$46	\$49	\$67	\$92	\$111

TABLE IV
HIGH-UTILITY ITEMSETS W.R.T *MINUTIL* = \$30

Itemset	Utility	Itemset	Utility
$\{d, a, g, e, c\}$	\$30	$\{b, e\}$	\$32
$\{b, e, c\}$	\$36	$\{b, h, e\}$	\$37
$\{e\}$	\$40	$\{b, h, e, c\}$	\$41
$\{e, c\}$	\$48	–	–

composed of several transactions, which are defined as $\mathcal{D} = \{T_1, T_2, T_3, \ldots, T_m\}$. We use T_{tid} to mark each transaction with a unique ID. Each transaction consists of many different items. In particular, an itemset is known as an l-itemset ($1 \leq l \leq n$) if it includes l items, and any itemset is a subset of I. Each item i_ℓ, where $1 \leq \ell \leq m$ has a nonnegative quantity $iu(i_\ell, t_j)$, which represents the occurring quantity (known as *internal utility*). The unit utility (referred to as the *external utility*) of i_ℓ is defined as $eu(i_\ell)$.

Definition 3.1: (*Utility of item*) In each transaction T_j, the utility of an item $i_\imath$ is denoted as $u(i_\imath, T_j)$, where $u(i_\imath, T_j) = eu(i_\imath) \times iu(i_\imath, T_j)$. According to the previous formula, the utility of the item $i_\imath$ in the database, denoted as $u(i_\imath)$, consists of $\sum_{i_\imath \in T_j \wedge T_j \subseteq \mathcal{D}} u(i_\imath, T_j) = \sum_{T_j \in \mathcal{D}} (eu(i_\imath) \times iu(i_\imath, T_j))$.

For example, in the second transaction in Table I, $u(c, T_2) = eu(c) \times iu(c, T_2) = \$1 \times 1 = \$1$. When considering the third transaction, $u(c, T_3) = eu(c) \times iu(c, T_3) = \$1 \times 2 = \$2$. Therefore, the total utility of item c in $\mathcal{D}$ is $u(c) = u(c, T_1) + u(c, T_2) + u(c, T_3) + u(c, T_4) + u(c, T_5) = \$9$.

Definition 3.2: (*Utility of an itemset*) The utility of an itemset X_i in a transaction T_j is denoted as $u(X_i, T_j) = \sum_{i_\imath \in X_i \wedge X_i \subseteq T_j} u(i_\imath, T_j)$. Similar to $u(i_\imath)$, the utility of an itemset in the database is defined as $u(X_i)$ and calculated as $u(X_i) = \sum_{X_i \subseteq T_j \wedge T_j \subseteq \mathcal{D}} u(X_i, T_j) = \sum_{X_i \subseteq T_j \wedge T_j \in \mathcal{D}} u(X_i, T_j) = \sum_{X_i \subseteq T_j \wedge T_j \in \mathcal{D}} \sum_{i_\imath \in X_i} u(i_\imath, T_j)$.

For example, in Table I, $u(\{c, e\}, T_1) = u(c, T_1) + u(e, T_1) = \$1 \times 2 + \$4 \times 2 = \$10$, $u(\{c, e\}, T_3) = u(c, T_3) + u(e, T_3) = \$1 \times 2 + \$4 \times 3 = \$14$, and $u(\{c, e\}, T_4) = u(c, T_4) + u(e, T_4) = \$1 \times 1 + \$4 \times 1 = \$5$. Furthermore, $u(\{c, e\}) = u(\{c, e\}, T_1) + u(\{c, e\}, T_3) + u(\{c, e\}, T_4) + u(\{c, e\}, T_5) = \$10 + \$14 + \$5 + \$19 = \$48$, and $u(\{g, h\}) = u(\{g, h\}, T_1) + u(\{g, h\}, T_6) = \$3 + \$3 = \$6$.

Definition 3.3: (*Utility of transaction*) The summed utility of all items in transaction T_j is denoted as the utility of T_j, which is computed as $tu(T_j) = \sum_{i_\imath \in T_j} u(i_\imath, T_j)$ where $1 \leq j \leq m$.

For example, $tu(T_1) = u(b, T_1) + u(c, T_1) + u(e, T_1) + u(g, T_1) + u(h, T_1) = \$2 \times 1 + \$1 \times 2 + \$4 \times 2 + \$2 \times 1 + \$1 \times 1 = \$15$, $tu(T_3) = u(b, T_3) + u(c, T_3) + u(e, T_3) + u(h, T_3) = \$2 \times 5 + \$1 \times 2 + \$\times 3 + \$1 \times 4 = \$28$, and the remaining utilities of transactions are presented in Table II.

Definition 3.4: (*Transaction-weighted utilization*) Given an itemset X_i belonging to $\mathcal{D}$, the TWU of X_i is denoted as $\text{TWU}(X_i)$ [9], which determines whether X_i is a potential HUI. This is the sum of the utilities of all transactions containing X_i in $\mathcal{D}$, where $\text{TWU}(X_i) = \sum_{T_j \in \mathcal{D} \wedge X_i \subseteq T_j} tu(T_j)$. It is an upper bound on real utility. The details of the proof can be found in [9].

For example, in Table I, $\text{TWU}(a) = (tu(T_2) + tu(T_4)) = \$19 + \$30 = \$49$, $\text{TWU}(c) = tu(T_1) + tu(T_2) + tu(T_3) + tu(T_4) + tu(T_5) = \$15 + \$19 + \$28 + \$30 + \$19 = \$111$, and the TWU values of all items are listed in Table III.

Definition 3.5: (*Utility-list*) The utility-list data structure [9] of itemset X_i consists of several tuples (including *tid*, *iutil*, and *rutil* for each transaction T_{tid} that includes X_i). The term *iutil* is the total utility of X_i in T_{tid}, and the *rutil* element is denoted by $\sum_{x_j \in T_{\text{tid}} \wedge x_j \notin X_i} u(x_j, T_{\text{tid}})$ [9], [11].

For example, the utility-list of $\{g\}$ is $\{T_1, \$2, \$1\}, \{T_2, \$8, 0\}, \{T_4, \$4, 0\}$, and $\{T_6, \$2, \$1\}$. The utility-list of $\{e\}$ is $\{T_1, \$8, \$3\}$, $\{T_3, \$12, \$4\}$, $\{T_4, \$4, \$4\}$, and $\{T_5, \$16, 0\}$. Therefore, the utility-list of $\{e, g\}$ is $\{T_1, \$10, \$1\}$ and $\{T_4, \$8, 0\}$.

Definition 3.6: (*High-utility itemset*) The minimum utility threshold (abbreviated as *minUtil*) is a special constraint that is predefined by users. An itemset X_i is known as an HUI if its utility value is no less than *minUtil* σ (i.e., $u(X_i) \geq \sigma$). Otherwise, we suppose that X_i is an uninteresting itemset and ignore it. In our work, we require two thresholds to select HUIs from target itemsets: *minUtil* is denoted as σ, whereas target *minUtil* is defined as ξ.

For example, if we suppose that σ is \$30, we can observe all HUIs from Table IV. In all 1-itemsets, only e is an HUI because $u(e) = \$8 + \$12 + \$4 + \$16 = \$40$ according to Table I.

Definition 3.7: (*Target high-utility itemset*) In practice, target itemsets are a user-specified set T'. The term target means that these itemsets are of special interest. It should be noted that $\mid T' \mid$ is no less than 1. The formulaic definition is $T' = < i_1, i_2, \ldots, i_\imath, \ldots, i_p > (\imath \in [1, p] \wedge i_\imath \in I)$. Suppose that an itemset X_i is an HUI, and T' is a given target itemset; if $\forall\, \imath$ that $i_\imath \in \mathcal{D}$

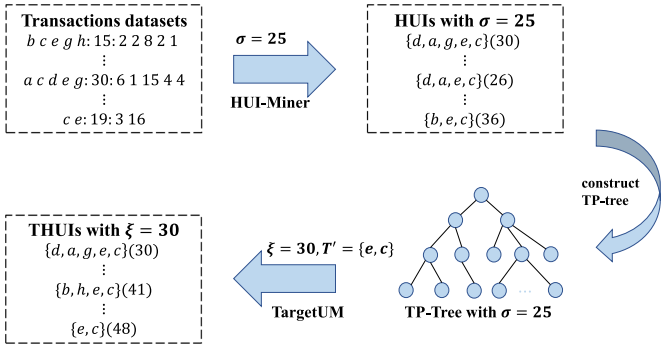


Fig. 1. TargetUM framework.

and $i_i \in T'$, where $i_i \in X_i$, we regard X_i as a target high-utility itemset (abbreviated as THUI), in which $u(X_i) > \xi$ and $T' \subseteq X_i$.

B. Problem Statement

At present, utility mining algorithms generally discover all HUPs with utility values of no less than σ. However, in real applications, not all HUPs are required by the user. For example, the user may wish to determine the HUPs that include a certain item or itemset. We refer to such patterns as target patterns regarding a certain item or itemset. Thus far, we have introduced the related basic concepts and preliminaries. We formulate the addressed problem in detail below.

Problem statement: The goal of THUIM is to identify all THUIs with two prespecified minimum utility thresholds by the user. Specifically, the problem of THUI mining can be interpreted as discovering all itemsets that contain target items and have a utility value of no less than σ and ξ.

For example, according to Table IV, if we set $T' = \{b, e\}$ and $\xi = \$30$, the THUIs are $\{b, e, c\}$ $(= \$36)$, $\{b, h, e\}$ $(= \$37)$, and $\{b, h, e, c\}$ $(= \$41)$, respectively. Moreover, if we set $T' = \{e, c\}$, we obtain $\{d, a, g, e, c\}$ $(= \$30)$, $\{b, h, e, c\}$ $(= \$41)$, $\{b, e, c\}$ $(= \$36)$, and $\{e, c\}$ $(= \$48)$, respectively. Obviously, ξ can also be lower than σ, depending on the user's requirement.

IV. TargetUM Algorithm

In this section, we present the target-based itemset querying algorithm known as TargetUM. The storage of THUIs mainly depends on the trie tree, and, eventually, the target itemsets are identified by on-the-fly querying in the tree. TargetUM can also flexibly query different target itemsets repeatedly with the same σ. In the following section, we discuss pruning strategies and outline the detailed steps of TargetUM.

A. Target Pattern Tree

The flowchart of the proposed TargetUM framework is illustrated in Fig. 1. First, the general HUIs are obtained from a transaction database by the HUI-Miner algorithm using the minimum utility threshold (σ). Then, TargetUM builds the target pattern tree (TP-tree) for all the HUIs. Finally, the TP-tree is spanned under the constraints of the minimum target utility threshold (ξ) and the target pattern (T') to obtain the final THUIs. As a key part of the TargetUM algorithm, the TP-tree structure is actually a trie tree. In an earlier study, MEIT [19] used a compact node compression mechanism to reduce the memory usage. In contrast to MEIT, TP-tree stores the complete HUIs after completing the mining process on the entire database instead of each transaction, which takes full advantage of the highly compact character of the trie tree. Finally, all HUIs containing the target items or itemsets can be identified.

TABLE V
HIGH-UTILITY PATTERNS W.R.T. $\sigma = \$25$

HUI	Utility	Remaining utility
$\{d, a, g\}$	$\{\$15, \$21, \$25\}$	$\{\$15, \$9, \$5\}$
$\{b, e\}$	$\{\$12, \$32\}$	$\{\$31, \$4\}$
$\{d, a, g, e\}$	$\{\$15, \$21, \$25, \$29\}$	$\{\$15, \$9, \$5, \$1\}$
$\{b, e, c\}$	$\{\$12, \$32, \$36\}$	$\{\$31, \$4, 0\}$
$\{d, a, g, e, c\}$	$\{\$15, \$21, \$25, \$29, \$30\}$	$\{\$15, \$9, \$5, \$1, 0\}$
$\{h, e\}$	$\{\$6, \$25\}$	$\{\$28, \$4\}$
$\{d, a, g, c\}$	$\{\$15, \$21, \$25, \$26\}$	$\{\$15, \$9, \$5, 0\}$
$\{h, e, c\}$	$\{\$6, \$25, \$29\}$	$\{\$28, \$4, 0\}$
$\{d, a, e\}$	$\{\$15, \$21, \$25\}$	$\{\$15, \$9, \$1\}$
$\{a, g, c\}$	$\{\$12, \$24, \$26\}$	$\{\$18, \$6, 0\}$
$\{d, a, e, c\}$	$\{\$15, \$21, \$25, \$26\}$	$\{\$15, \$9, \$1, 0\}$
$\{e\}$	$\{\$40\}$	$\{\$8\}$
$\{b, h, e\}$	$\{\$12, \$17, \$37\}$	$\{\$31, \$26, \$4\}$
$\{e, c\}$	$\{\$40, \$48\}$	$\{\$8, 0\}$
$\{b, h, e, c\}$	$\{\$12, \$17, \$37, \$41\}$	$\{\$31, \$26, \$4, 0\}$

Definition 4.1: (*Target pattern tree*) Each node in the TP-tree contains *name*, *parent*, *twu*, *sumIu*, *sumRu*, *isEnd*, and *link*. As indicated in Fig. 2 , *name* represents each item and *link* is the unique chain of each item. The *children* and *parent* of a node in a TP-tree link other nodes and all of these are used to construct the trie. Obviously, all nodes in the branches are sorted by TWU-ascending order. The *twu*, *sumIu*, and *sumRu* are used to record the TWU, total utility, and total remaining utility of each item/node, respectively. Finally, *isEnd* indicates whether the current node is the last item of the HUI, because the TP-tree is always traversed from top to bottom.

Definition 4.2: (*Priority of items*) To facilitate the mining process in TargetUM, all of the items in each transaction are sorted by TWU-ascending order, and each transaction is revised. Table III(b) presents the final status of all items.

Definition 4.3: (*Remaining utility of itemset*) Given an itemset X_i and a transaction T_j with $X_i \subseteq T_j$, all items behind X_i in T_j are known as the remaining items (T_j/X_i). Subsequently, the remaining utility of X_i in T_j is denoted as $ru(X_i, T_j) = \sum_{X_i \subseteq T_j \wedge x_i \in T_j/X_i} u(x_i, T_j)$ [9].

For example, in Table I, if we set $X_i = \{b, h\}$, T_1 / $X_i = \{g, e, c\}$ and T_3 / $X_i = \{e, c\}$. Therefore, $ru(X_i, T_1) = u(g, T_1)$ + $u(e, T_1)$ + $u(c, T_1) = \$2 \times 1 + \$4 \times 2 + \$1 \times 2 = \$12$, and $ru(X_i, T_3) = u(e, T_3)$ + $u(c, T_3) = \$4 \times 3 + \$1 \times 2 = \$14$. When setting $\sigma = \$25$, all HUPs can be identified using an existing HUIM algorithm, as indicated in Table V.

It should be noted that the mining process and TP-tree construction are executed simultaneously (i.e., every discovered HUI is immediately inserted into the TP-tree). Taking Table I as an example, the details of how to construct a TP-tree are introduced in the following.

Initially, the root node of the TP-tree is always empty. After determining the first HUI_1 $\{d, a, g\}$, the remaining utility values

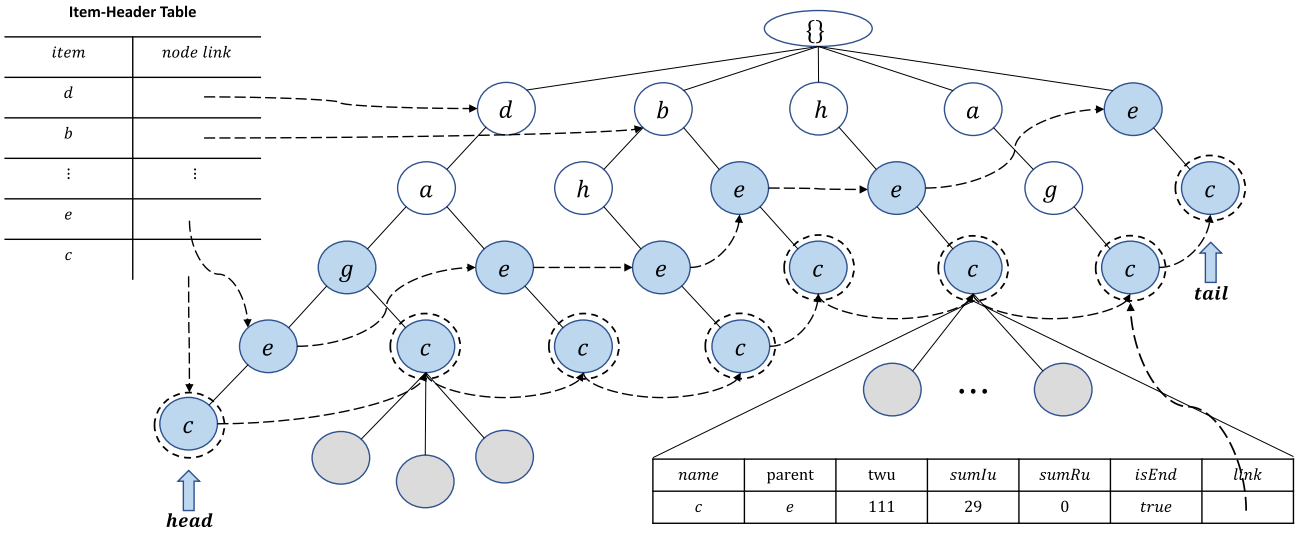


Fig. 2. Constructed TP-tree with $\sigma = \$25$.

are also recorded. Subsequently, three nodes (d, a, and g) are inserted into the TP-tree. As the priority of nodes, d is the first one. At this time, it will be verified whether there already exists a node d that is directly connected to the root node. If so, HUI_1 shares a common prefix $< d >$ with the remaining items. Otherwise, a new node d is created and the relative parameters are set as *d.name* $= d$, *d.parent* $=$ *root*, *d.twu* $=$ \$30, *d.sumIu* $=$ \$15, and *d.sumRu* $=$ \$15. As d is not the last item in HUI_1, *d.isEnd* is *false*. An item header table of d is created simultaneously. The function of the item header table can quickly locate the position of an item in the TP-tree. Because item d is first added to the item header table, it is marked as a head node and tail node together. The remaining items/itemsets are processed and repeated using the same steps. Thus far, a branch $\{d, a, g, e, c\}$ of the TP-tree has been successfully created. Based on this branch, the method of inserting another HUI_2 $\{d, a, e\}$ is considered.

It can easily be observed that HUI_1 and HUI_2 share the common prefix $< d, a >$; thus, HUI_2 creates a new branch from HUI_1. Owing to the characteristics of the TP-tree, the overall key information regarding $u(X)$ and $ru(X)$ of itemset X is fixed in the same branch. Therefore, we only need to create a new node e and set the remaining parameters as *e.name* $= e$, *e.parent* $= a$, *e.twu* $=$ \$92, *g.sumIu* $=$ \$25, *g.sumRu* $=$ \$1, and *e.isEnd* $=$ *true*. Because item e already exists in the item header table, we link the new node and then set the last e as the new tail node directly. Similarly, the processes of the other HUIs follow the same steps. Finally, a complete TP-tree is constructed, as depicted in Fig. 2.

B. Search Space for Mining THUIs

In this section, we explain the search space of a TP-tree constructed with HUIs. The TP-tree is an n-array tree rather than a binary tree. As indicated in Table III(b), all nodes/items are sorted by TWU-ascending order. To reduce the searching consumption, TargetUM discovers target itemsets in a bottom-to-top manner. Several important properties and lemmas of the TP-tree are listed below.

Property 4.1: The TWU of each node in the TP-tree is no less than its parent node.

Property 4.2: Each HUI is a unique path/branch of the TP-tree and this path starts with the root node.

Property 4.3: The insertion of an HUI into the TP-tree does not update the utility and remaining utility values of the existing nodes.

Lemma 1: In the TP-tree, each node represents an item. Let X^k be a k-itemset (node) and let its parent node be denoted by X^{k-1}. According to the TWU property [15], we obtain $\text{TWU}(X^k) \leq \text{TWU}(X^{k-1})$.

Proof: X^k is constructed by joining the pairs of X^{k-1}, and the transaction containing X^k also will include X^{k-1}. Furthermore, the number of transactions containing X^{k-1} is always higher than or equal to that of X^k. Therefore, considering the definition of TWU, we obtain $\text{TWU}(X^k) \leq \text{TWU}(X^{k-1})$. ■

Lemma 2: Every branch of the TP-tree is a unique path and all of the HUIs with σ can be derived from the TP-tree. We can discover all THUIs with ξ and T' from the TP-tree.

Proof: In the TP-tree, each node except for the root only has one parent node. In the construction process of the TP-tree, each HUI is mapped to a unique path in the tree. Accordingly, the TP-tree contains all itemsets with σ and provides the unique determination of each THUI. ■

C. Efficient Pruning Strategies

In the designed TP-tree, itemset X^k is obtained by performing the join operation of the utility-lists of itemsets X^{k-1}. This not only effectively accelerates the candidate generation but also saves time when querying targeted itemsets. In the preceding subsection, we described how to use the TP-tree to store key information and then to discover HUIs. In this subsection, we introduce several pruning strategies.

As mentioned in [10], [14], and [15], TWU is a tight upper bound that has three important properties (overestimation, antimonotonicity, and pruning). Let X_i and X_j be two distinct itemsets. If $X_i \subset X_j$, TWU(X_i) $\geq$ TWU(X_j) because of the antimonotonicity. The overestimation property means that the TWU of an itemset is always higher than or equal to its real utility.

Strategy 1: Based on the antimonotonicity and overestimation, it is clear that if TWU of an itemset X_i is less than *minUtil* (σ in this article), X_i will be a low-utility itemset as well as its supersets [15].

Let the sum of the utilities in the utility-list of an itemset X_i be denoted as *sumIu*(X_i). If *sumIu*(X_i) is no less than *minUtil*, X_i is an HUI; otherwise, it is an uninteresting itemset. An item x_j after X_i in T_j that can be appended to X_i is known as the remaining item of X_i. Moreover, *sumRu*(X_i) is referred to as the remaining utility of X_i, and it represents the sum of the utilities of the remaining items such that *sumRu*(X_i) = $\sum_{T_j \in \mathcal{D} \wedge X_i \in T_j \wedge x_j \in T_j / X_i} u(x_j, T_j)$ [9].

Strategy 2: If the sum of *sumIu* and *sumRu* in the utility-list of X_i is higher than or equal to *minUtil* (σ and ξ in this article), the extension of X_i and their transitive extensions are the potential HUIs. Otherwise, they are low-utility itemsets, and X_i and its extensions can safely be ignored [9].

It can be observed from Fig. 2 that the TP-tree becomes increasingly luxurious with increasing nodes. In fact, not all branches of the TP-tree contain target itemsets; therefore, TargetUM needs to cut off several useless branches in advance to avoid wasting construction and querying time. First, TargetUM needs to determine whether or not the current branch is worth using to locate the corresponding item using the first pruning strategy.

The TargetUM algorithm is intended to identify all HUIs that contain all target itemsets. When determining the target HUIs, it should be ensured that the search branches include the target patterns. Otherwise, the current querying operation is meaningless. In the TP-tree, each item of any path is sorted in TWU-ascending order. Accordingly, the TWU value of each node in the TP-tree is no less than that of its parent node. After sorting, the position of any node in the TP-tree can rapidly be located according to the item header table. In the following section, we describe how to filter out the useless branches.

Strategy 3: Select the item p with $\text{TWU}_{\max}$ in the target itemsets, locate the position of node p and current branch γ in the TP-tree through the item header table, and query upwards to obtain the target utility pattern X. In TargetUM, T' is sorted by the TWU-ascending order and *p.twu* $\geq$ *p.parent.twu* according to Property 4.1. Thus, for i_i, $i_i \in T'$, if i_i may exist in γ, *currentNode.twu* $\geq$ TWU(i_i). If *currentNode.twu* $<$ TWU(i_i), i_i does not exist in γ, and neither is T'; thus, γ is discarded. Note that more than one item may have the same TWU; thus, if *currentNode.twu* = TWU(i_i), it is also necessary to determine whether *currentNode.name* = i_i.

For example, assume an HUI = $\{d, a, g, e, c\}$ and $T' = \{e, c\}$. The TWU of item c is the largest, and node e is the parent node of c. In this case, T' is a subset of the HUI (*c.twu* $\geq$*e.twu*). If we set $T' = \{b, c\}$, the TWU of c is the largest. Subsequently, we need to determine b and obtain *e.twu*, *g.twu*, and *a.twu* $>$ TWU(b). Finally, we obtain *d.twu* $<$ TWU(b), which means that b is not in the HUI and neither is $T' = \{b, c\}$.

Algorithm 1: Construct TP-Tree Procedure.

Input: X': the prefix of HUI; *IUs*: the utility list of X'; *RUs*: the remaining utility list of X'; *TUs*: the *twu* list of X'; x: the last item of HUI; α: the utility of x; β: the remaining utility of x; θ: the *twu* of x.

Output: a TP-tree.

```
1  listNodes ← root.childs;
2  currentNode ← NULL;
3  parentNode ← NULL;
4  for each element e ∈ X', e.iu ∈ IUs, e.ru and e.twu ∈ RUs do
5      currentNode ← use binary search method to find e in listNodes;
6      if currentNode == NULL then
7          currentNode ← create a new node as Node(e, e.iu, e.ru, e.ru);
8          append currentNode to listNodes;
9          call Item-Header Table(e, currentNode);
10     end
11     parentNode ← currentNode;
12     listNodes ← currentNode.childs;
13 end
14 currentNode ← use binary search method to find e in listNodes;
15 if currentNode == NULL then
16     currentNode ← create a new node which Node(x, α, β, θ);
17     call Item-Header Table(e, currentNode);
18 else
19     currentNode.sumIu ← α;
20     currentNode.sumRu ← β;
21     currentNode.twu ← θ;
22 end
23 currentNode.isEnd ← TRUE.
```

D. Proposed Targeted Querying Algorithm

Based on the previous introduction, the main framework of the TargetUM algorithm is given here. There are many ways to discover general HUIs from a database, and TargetUM adopts HUI-Miner [9]. Algorithm 1 describes the process of constructing a TP-tree; Algorithms 3 and 4 describe the process of querying the TP-tree to discover THUIs. The proposed novel TargetUM algorithm can be described as follows.

1) Target Pattern Trie Construction: The TP-tree plays a key role during the mining process. Algorithm 1 demonstrates how to construct a TP-tree. It takes an HUI $X'x$ (X' is initialized as null and x is the last item of $X'x$), and other information of X', such as *IUs*, *RUs*, *TUs*, $u(x)$ (which refers to α), and *twu*(x) (which refers to β) as input. A set of HUIs is obtained by HUI-Miner [9]. Once a new HUI is discovered, it is inserted into a

Algorithm 2: Item-Header Table Procedure.

Input: x: an item; *node*: a node named x; *mapItemNode*: store the header node of each item-header table; *mapItemLastNode*: store the tail nodes of each item-header table.
Output: an item-header table of x.

```
1  lastNode ← a node named x ∈ mapItemLastNode;
2  if lastNode != NULL then
3  |  lastNode.link ← node;
4  end
5  append node to mapItemLastNode of x;
6  headNode ← a node named x ∈ mapItemNode;
7  if lastNode == NULL then
8  |  append node to mapItemLastNode of x;
9  end
10 return an item-header table of x;
```

Algorithm 3: Output Prefix Procedure.

Input: T': a target itemset; *mapItemNode*: the header node of each item-header table; ξ: user-specified target *minUtil*; *mapItemToTwu*: a map of *twu* values of each item.
Output: *THUIs*: a list of target high-utility itemsets.

```
1  currentNode ← NULL;
2  posToMatch ← |T'|;
3  node ← the last item of T';
4  while node != NULL do
5  |  if (node.sumIu + node.sumRu) ≥ ξ then
6  |  |  the current HUI X ← node.name;
7  |  |  posToMatch decrease 1;
8  |  |  currentNode ← node.parent;
9  |  |  while currentNode != NULL do
10 |  |  |  if posToMatch ≥ 0 then
11 |  |  |  |  a new item y ← T'[posToMatch];
12 |  |  |  |  if currentNode.twu < y.twu then
13 |  |  |  |  |  break;
14 |  |  |  |  end
15 |  |  |  |  if currentNode.twu == Y.twu AND
   |  |  |  |   currentNode.item == y then
16 |  |  |  |  |  posToMatch decrease 1;
17 |  |  |  |  end
18 |  |  |  end
19 |  |  |  X ← X ∪ currentNode.name;
20 |  |  |  currentNode ← currentNode.parent;
21 |  |  end
22 |  |  if posToMatch == -1 then
23 |  |  |  if node.sumRu ≥ ξ AND node.isEnd ==
   |  |  |   TRUE then
24 |  |  |  |  new THUIs ← THUIs ∪ X;
25 |  |  |  end
26 |  |  |  call Output Suffix(node.childs, X);
27 |  |  end
28 |  |  node ← the next node of T';
29 |  end
30 end
```

TP-tree with target *minUtil* σ. Lines 1–3 provide the three initial parameters. For each item $e \in X'$, we can quickly determine the position (w.r.t. the index) of e in the TP-tree (line 5). If e does not exist, a new node e is created and inserted into the TP-tree, following which Algorithm 2 records the relative message of e (lines 6–9). After processing all items of X', a complete TP-tree of X' is constructed (lines 4–13). Thereafter, in line 14, x is determined by the binary search method. If no new node is identified, Algorithm 2 is called (lines 15 to 17). In contrast, if x is already stored in the TP-tree, it only updates certain relative information (lines 19 to 21). Finally, after dealing with the last item x, it will be the end of itemset $X'x$, where $X'x$ is a complete HUI (line 23).

2) Item Header Table Construction: The item header table plays a key role in the construction of a TP-tree. Algorithm 2 lists the details regarding the construction procedure. The function of the item header table aims to build the node chains of interesting items. According to the mark header and tail indexes, with the binary searching method, the item header table makes it easy to obtain the real position of any item (lines 1–10). Finally, Algorithm 2 returns an updated item header table containing x (line 10).

3) Output Function: All the target HUIs can be discovered after constructing the complete TP-tree. As any THUI must be an HUI ($\forall$THUI $\subseteq$ HUIs), several situations may exist.

1) The THUI is located at the end of the HUI.
2) The THUI is located in the primary of the HUI.
3) The THUI is located in the HUI.
4) The THUI continuously occurs in the HUI.
5) The THUI discontinuously occurs in the HUI.

To this end, we design two methods to deal with these five situations. To ensure the correctness and completeness of the discovered results, we adopt the depth-first searching method. The details are provided below.

We first determine in advance whether a prefix of the THUI exists (Algorithm 3). The last item of the target itemset T' is set as the *node*. According to *node* and *mapItemNode*, we can easily obtain all the information regarding the current HUI (a branch includes *node*). The *posToMatch* parameter is used to record the current two items (lines 1–3). Subsequently, we traverse each item of T'. As demonstrated in strategy 2, if *sumIu* + *sunRu* is no less than ξ, the current itemset X may be a THUI, and the remaining items should be compared further (lines 4–30). However, in line 12, if the TWU of *currentNode* is less than the current target item y, the comparison process can be directly aborted (strategy 3). Once the *posToMatch* is equal to –1, we have identified an entire target itemset, where $T' \subseteq X$, following which TargetUM can call Algorithm 4 to output this new THUI (lines 22–27).

Algorithm 4 is more succinct than Algorithm 3. It is a recursive method for exploring all the suffix nodes of X. The expanded HUI X' can be obtained for each *node* $\in$ *Us* (line 2). Subsequently, TargetUM filters out nontarget HUIs and stores the THUIs using a list (lines 3–4). Finally, according to strategy

Algorithm 4: Output Suffix Procedure.

Input: Us: a branch that containing target itemset; X: the prefix of a HUI; ξ: the target *minUtil*.
Output: *THUIs*: a list of target high-utility itemsets.

```
1 for each node ∈ Us do
2     X' ← X ∪ node.name;
3     if node.sumIu ≥ ξ AND node.isEnd == TRUE then
4         update THUIs ← THUIs ∪ X';
5     end
6     if node.childs != NULL AND node.sumIu +
        node.sumRu ≥ ξ then
7         call outputSuffix(node.childs, X');
8     end
9 end
```

2, it determines whether to continue exploring the suffix nodes (lines 6–7).

V. Performance Evaluation

To the best of our knowledge, THUIM has not yet been studied. Therefore, there is no suitable algorithm for comparison. To determine the effectiveness of our novel algorithm, we conducted an extensive experiment to compare TargetUM with the HUI-Miner algorithm [9] directly. The function of HUI-Miner in the experiments was executed as target postprocessing. This can be regarded as a benchmark experiment to verify the accuracy of the results discovered by TargetUM. Both real-life and synthetic datasets were used to verify the effectiveness of TargetUM.

A. Target Postprocessing

In short, only simple modifications were made to the original HUI-Miner algorithm, and the user-defined target patterns were extracted from the HUIs using a postprocessing mechanism. The HUIs that were discovered by HUI-Miner were checked by T'. If each item in T' was included in the HUIs, this HUI could be said to be a THUI. For example, given two HUIs, $\{d, a, g, e\}$ and $\{d, a, g, e, c\}$, and the user-defined target pattern string was $\{e, c\}$. Following postprocessing, $\{d, a, g, e, c\}$ contained the target pattern $\{e, c\}$; thus, it was a THUI. However, $\{d, a, g, e\}$ could not be a THUI because it did not contain item c in T'. Note that this postprocessing was used as a verification algorithm to verify the accuracy of the results of TargetUM. The HUI-Miner algorithm with a postprocessing mechanism is denoted as HUI-mp.

B. Experimental Setup and Dataset

Experiments were conducted on four real-life datasets, as shown in Table VI. The characteristics of these datasets include the following:

1) the name of the dataset;
2) the size of each dataset;
3) the number of transactions for each dataset;
4) the number of items for each dataset;
5) the average length (AvgLen) of the transaction;
6) the maximum length (MaxLen) of the transaction.

TABLE VI
Characteristics of Different Datasets

Dataset	Size (kB)	Trans	Items	AvgLen	MaxLen
chainstore	81102	1112950	46086	7.2	170
ecommerce	1968	14976	3468	11.6	29
retail	1845	24735	11371	10.3	74
foodmart	176	4141	1559	4.4	14

Both TargetUM and HUI-mp were implemented using the Java language and executed on a PC with an AMD Ryzen 5 3600 6-Core Processor 3.60 GHz and 8 GB of memory, running on a 64-b Microsoft Windows 10 platform.

Foodmart, chainstore, and ecommerce are three real-life customer transaction datasets with real utility values. The foodmart dataset originates from a retail store, and it was obtained and transformed from SQL-Server 2000, with 4141 transactions and 1559 items. The chainstore dataset was obtained from a major grocery store chain in California, USA, and it was transformed from the NU-Mine Bench, with 1 112 949 transactions and 46 086 items. The ecommerce dataset contains 14 975 transactions and 3486 items. Retail is a real dataset with synthetic utility values, and the internal utility values are generated using a uniform distribution in [1, 10]. Thus, retail consists mainly of customer transactions from an anonymous Belgian retail store.

In the following experiments, TargetUM was compared with several variants based on the use of different pruning strategies. These are denoted as **TargetUM**$_{13}$ (with pruning strategies 1 and 3), **TargetUM**$_{23}$ (with pruning strategies 2 and 3), **TargetUM**$_3$ (with pruning strategy 3), and **TargetUM** (with all pruning strategies), respectively. These four versions were used to evaluate the efficiency of TargetUM. In addition, for the feasibility of the experiments, the appropriate utility threshold (σ) and target utility threshold (ξ) were set for different datasets with reference to their overall utility values. When the thresholds were varied, equal increments are applied and appropriate incremental intervals are selected with reference to each dataset.

C. Itemset Analysis

TargetUM was initially proposed to solve the target HUM problem. The main goal is to obtain accurate targeted HUPs. Thus, the results were verified in the form of a comparison with HUI-mp. HUI-mp used postprocessing to verify the results. Note that, for comparison purposes, the target HUIs obtained by TargetUM are denoted as THUIs*, whereas those obtained by HUI-mp are denoted as THUIs. By analyzing the datasets, we set different thresholds and T' for different datasets.

We compared the number of itemsets on the test datasets to demonstrate the effect of varying the parameters σ, ξ, and T', and the results for various parameters are displayed in Tables VII, VIII, and IX, respectively. It is worth noting that T' refers to the target pattern, i.e., the specific item that appears in the tested dataset. As can be observed from these tables, THUIs* and THUIs obtained the same count. For Table VII, letting $\sigma = \xi$, and σ or ξ remains unchanged, with the increase in σ (ξ), the count of the THUIs became increasingly smaller. For Table VIII, with T'

TABLE VII
DERIVED ITEMSETS UNDER VARIED σ ($\sigma = \xi$)

Dataset	Pattern	# itemsets by varying threshold σ ($\sigma = \xi$)					
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
chainstore	THUIs*	267	207	170	131	106	91
{16967}	THUIs	267	207	170	131	106	91
ecommerce	THUIs*	4,643,198	2,463,092	1,641,208	1,050,396	596,385	293,757
{150561222}	THUIs	4,643,198	2,463,092	1,641,208	1,050,396	596,385	293,757
retail	THUIs*	34,240	26	19	16	14	12
{976}	THUIs	34,240	26	19	16	14	12
foodmart	THUIs*	9,214	9,196	8,833	7 727	5 896	3 494
{1340}	THUIs	9,214	9,196	8,833	7 727	5 896	3 494

TABLE VIII
DERIVED ITEMSETS UNDER VARIED T' ($\sigma = \xi$)

Dataset	Pattern	# itemsets by varying T'					
		T'_1	T'_2	T'_3	T'_4	T'_5	T'_6
chainstore	THUIs*	267	20	16	100	3	3
σ = 300000	THUIs	267	20	16	100	3	3
ecommerce	THUIs*	3,250,046	2,200,992	811,970	506,036	390,088	240,666
σ = 550000	THUIs	3,250,046	2,200,992	811,970	506,036	390,088	240,666
retail	THUIs*	34,213	34,200	34,200	34,079	34,065	34,019
σ = 1370	THUIs	34,213	34,200	34,200	34,079	34,065	34,019
foodmart	THUIs*	4 096	64	32	32	4	416
σ = 20	THUIs	4 096	64	32	32	4	416

TABLE IX
DERIVED ITEMSETS UNDER VARIED ξ

Dataset	Pattern	# itemsets by varying threshold ξ					
		ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ
chainstore	THUIs*	267	207	170	131	106	91
	THUIs	267	207	170	131	106	91
ecommerce	THUIs*	4,643,198	2,463,092	1,641,208	1,050,396	596,385	293,757
	THUIs	4,643,198	2,463,092	1,641,208	1,050,396	596,385	293,757
retail	THUIs*	34,240	26	19	16	14	12
	THUIs	34,240	26	19	16	14	12
foodmart	THUIs*	9,214	9,196	8,833	7,727	5,896	3,494
	THUIs	9,214	9,196	8,833	7,727	5,896	3,494

as a parameter, with either a single item or an itemset, multiple T' were set for each dataset. Note that no direct relationship existed between T'_k and T'_{k+1} because of the limitation of the average transaction length of the dataset, such as chainstore and foodmart.

Table IX satisfies multiple target queries under the same minimum utility threshold with varying ξ, fixed σ, and fixed T'. We set the chainstore $\sigma = 300\,000$, $T' = \{16\,967\}$, the ecommerce $\sigma = 550\,000$, $T' = \{150\,561\,222\}$, the retail $\sigma = 1370$, $T' = \{976\}$, and the foodmart $\sigma = 20$, $T' = \{1340\}$. We can obtain the same results from Table IX as those in Table VII. According to the above analysis, it can be concluded that the proposed TargetUM algorithm is effective for the THUIM problem.

D. Efficiency Analysis

To evaluate the overall performance of the proposed algorithm more effectively, the following analysis was performed in terms of the runtime, memory consumption, and candidates. The detailed results are presented in Figs. 3–5.

Fig. 3 depicts the running time of four variants of the TargetUM algorithm. We set varying σ and ξ with $\sigma = \xi$ and a fixed T'.

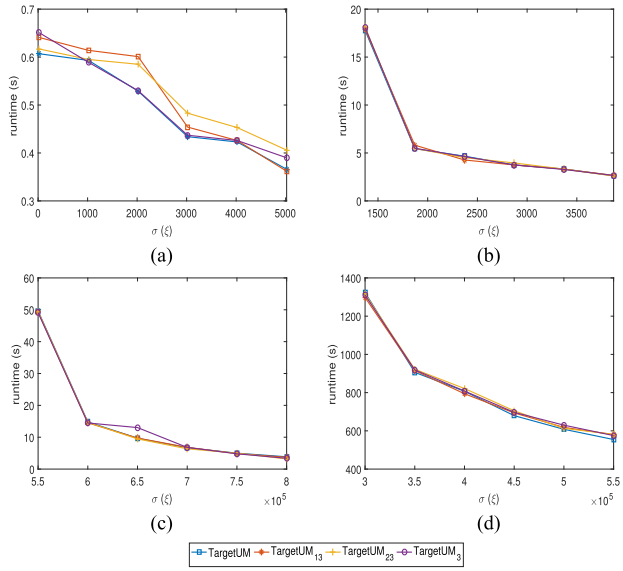


Fig. 3. Runtime under varied σ and ξ with $\sigma = \xi$ and a fixed T'. (a) foodmart ($\sigma = 20$, $T' = \{1340\}$). (b) retail ($\sigma = 1370$, $T' = \{976\}$). (c) ecommerce ($\sigma = 550\,000$, $T' = \{150561222\}$). (d) chainstore ($\sigma = 300\,000$, $T' = \{16967\}$).

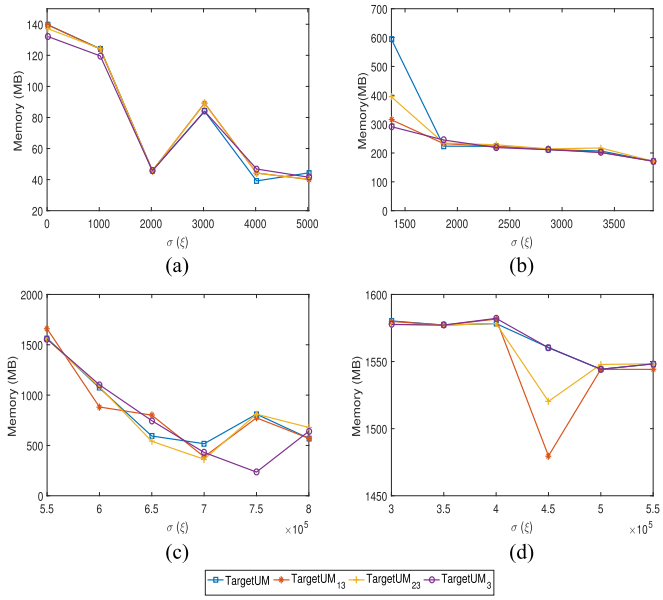


Fig. 4. Memory under varied σ and ξ with $\sigma = \xi$ and a fixed T'. (a) foodmart ($\sigma = 20, T' = \{1340\}$). (b) retail ($\sigma = 1370, T' = \{976\}$). (c) ecommerce ($\sigma = 550\,000$, $T' = \{150561222\}$). (d) chainstore ($\sigma = 300\,000$, $T' = \{16967\}$).

As can be observed in Fig. 3, the overall running time gradually decreased with an increase in σ or ξ. The TargetUM with the three strategies had the shortest running time. Strategies 1 and 2 could prune most of the search space. It is interesting to note that the overall running time of the four versions of TargetUM did not differ significantly, owing to the limitation of the TP-tree, which is a HUI-based tree and also a set-enumeration tree. Therefore, limited by the actual storage, the number of HUIs for building the TP-tree could be too high, which led to the algorithm searching the end of the tree quickly, and only a slight difference can be observed from Fig. 3. This is clear from Fig. 3(a), in which

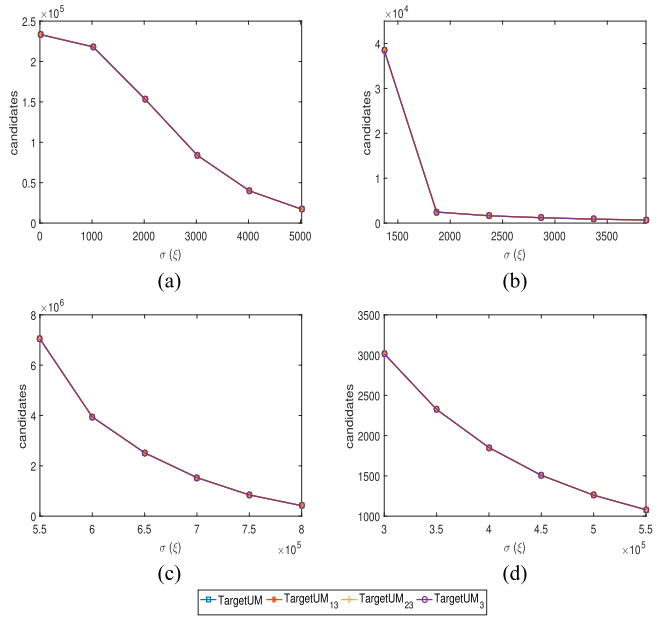


Fig. 5. Candidates of the compared methods under varied σ and ξ with $\sigma = \xi$ and a fixed T'. (a) foodmart ($\sigma = 20$, $T' = \{1340\}$). (b) retail ($\sigma = 1370$, $T' = \{976\}$). (c) ecommerce ($\sigma = 550\,000$, $T' = \{150561222\}$). (d) chainstore ($\sigma = 300\,000$, $T' = \{16967\}$).

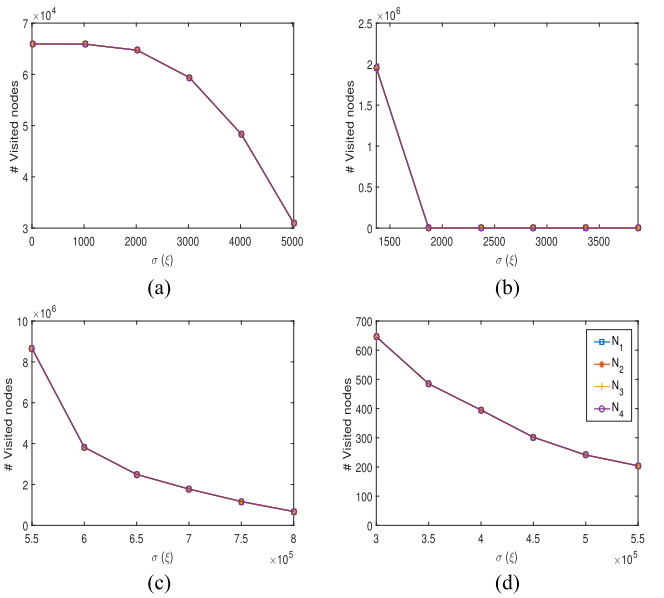


Fig. 6. Number of visited nodes under varied σ and ξ with a fixed T'. (a) foodmart ($\sigma = \xi = 20$, $T' = \{1340\}$). (b) retail ($\sigma = \xi = 1370$, $T' = \{976\}$). (c) ecommerce ($\sigma = \xi = 550\,000$, $T' = \{150561222\}$). (d) chainstore ($\sigma = \xi = 300\,000$, $T' = \{16967\}$).

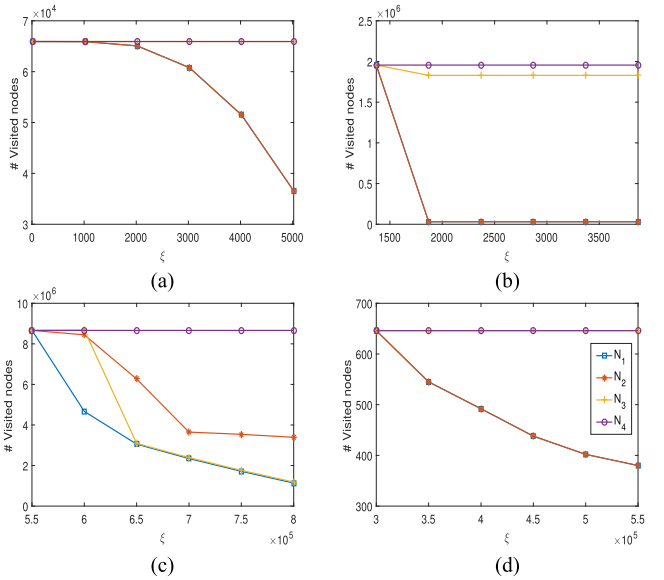


Fig. 7. Number of visited nodes under varied ξ with a fixed σ and a fixed T'. (a) foodmart ($\sigma = 20$, $T' = \{1340\}$). (b) retail ($\sigma = 1370$, $T' = \{976\}$). (c) ecommerce ($\sigma = 550\,000$, $T' = \{150561222\}$). (d) chainstore ($\sigma = 300\,000$, $T' = \{16967\}$).

foodmart was a sparse dataset and the overall running time that was required was very short; thus, the difference could be expressed. However, for the retail, ecommerce, and chainstore datasets, the difference was not easy to express.

Figs. 4 and 5 present the memory usage and number of candidate itemsets of the algorithm. It can be observed from the figures that the memory usage of the four versions of the TargetUM algorithm as a whole gradually decreased with the increase in the threshold value and tended to be consistent, because TargetUM first constructed the TP-tree using HUIs and then determined the THUIs that contained T' by searching the TP-tree. Therefore, the search process did not affect the memory usage. Similarly, as any HUI was likely to be a THUI, the HUIs were candidate itemsets and were fixed.

E. Effect of Pruning Strategies

To evaluate the pruning strategy further, the number of visited nodes in the TP-tree was compared. For comparison purposes, the number of visited nodes of TargetUM (with all pruning strategies) is denoted as N_1, TargetUM$_{13}$ is denoted as N_2, TargetUM$_{23}$ is denoted as N_3, and TargetUM$_3$ is denoted as N_4. Figs. 6 and 7 present the experimental results with different parameter settings.

Fig. 6 indicates a varying $\sigma = \xi$ with a fixed T', and σ and ξ were changed accordingly. In Fig. 7, ξ was varied with a fixed σ and a fixed T'. The following information can be obtained based on Figs. 6 and 7.

1) Given $\sigma = \xi$, equivalent to a regular query, all HUIs that contain T' must be the interesting THUIs. The pruning strategies were not very useful; thus, it can be observed that $N_1 = N_2 = N_3 = N_4$. In general, the number of visited nodes decreased as σ or ξ increased.
2) The TargetUM algorithm first constructed the TP-tree and finally processed the query on the tree. The pruning strategies mainly worked on the TP-tree, as illustrated in Fig. 7. When ξ was set to be varying with a fixed σ, and then multiple queries were performed, the number of visited nodes decreased as ξ increased.
3) Comparing N_1 and N_4, it can be observed that N_4 was always the same, because the entire TP-tree was searched on the fly. Strategy 3 was only dependent on an order, and the T' set in the experiment had only one item. For

T', which had $|T'| > 1$, strategy 3 worked. Compared to N_4, N_1 pruned a large part of the search space in the TP-tree. Thus, TargetUM$_1$ always consumed the shortest runtime among TargetUM (with all pruning strategies), TargetUM$_{13}$, TargetUM$_{23}$, and TargetUM$_3$, as indicated in Fig. 3.

4) Comparing N_2 and N_3 from Fig. 7(b) and (c), it can be concluded that both strategies 1 and 2 had a positive effect on accelerating the queries, but the results exhibited differences. Strategy 1 acted on the prefix and strategy 2 acted on the suffix, both of which played an important role. Finally, strategies 1 and 2 both had a positive effect on reducing the search space but exhibited variability with the sparsity of the dataset and $|T'|$.
5) T' is a significant factor for improving the mining performance on the THUI querying task.

VI. Conclusion

Existing utility-driven mining algorithms can only determine all HUIs that satisfy σ in the transaction database, and not all of these itemsets are interesting. In this article, we first proposed the target utility mining problem and designed an algorithm known as TargetUM to handle this task. Target pattern mining always occupies substantial memory when stored, but storage resources are limited. A utility-based trie tree, namely a TP-tree, was designed. TargetUM adopts two utility threshold constraints to obtain the desired results and can also save on the memory cost. TargetUM can discover THUIs under the constraints of T', σ, and ξ. Moreover, conducted experiments on real and synthetic datasets to demonstrate that the TargetUM algorithm can obtain complete and accurate target results.

It is a significant challenge to reduce the execution time and memory consumption in the addressed task. TargetUM first calculates all HUIs, then constructs the TP-tree, and finally determines the THUIs. Therefore, in the future, we intend to study more efficient algorithms and new data structures that can save on the execution time and memory cost while improving the performance. In particular, extending the algorithm to process Big Data in distributed systems will be a quite challenging task.

References

[1] C. C. Aggarwal, M. A. Bhuiyan, and M. Al Hasan, "Frequent pattern mining algorithms: A survey," *Frequent Pattern Mining*, pp. 19–64, 2014.
[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th ACM Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.
[3] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, 2000.
[4] P. Fournier-Viger, J. C. W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *Wiley Interdiscipl. Rev. Data Mining Knowl. Discov.*, vol. 7, no. 4, 2017, Art. no. e1207.
[5] P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Sci. Pattern Recognit.*, vol. 1, no. 1, pp. 54–77, 2017.
[6] W. Gan, J. C. W. Lin, H. C. Chao, and J. Zhan, "Data mining in distributed environment: A survey," *Wiley Interdiscipl. Rev. Data Mining Knowl. Discov.*, vol. 7, no. 6, 2017, Art. no. e1216.
[7] W. Gan *et al.*, "A survey of utility-oriented pattern mining," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1306–1327, Apr. 2021.
[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 Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1772–1786, Aug. 2013.
[9] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 55–64.
[10] P. Fournier Viger, C. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *Proc. Int. Symp. Methodologies Intell. Syst.*, 2014, pp. 83–92.
[11] J. C. Lin, P. Fournierviger, and W. Gan, "FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits," *Knowl. Based Syst.*, vol. 111, pp. 283–298, 2016.
[12] 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 Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.
[13] U. Yun, H. Ryang, and K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Syst. Appl.*, vol. 41, no. 8, pp. 3861–3878, 2014.
[14] 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," *Knowl. Inf. Syst.*, vol. 51, no. 2, pp. 595–625, 2017.
[15] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proc. Pacific Asia Conf. Knowl. Discov. Data Mining*, 2005, pp. 689–695.
[16] M. Kubat, A. Hafez, V. V. Raghavan, J. R. Lekkala, and W. K. Chen, "Itemset trees for targeted association querying," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 6, pp. 1522–1534, Nov./Dec. 2003.
[17] L. Shabtay, R. Yaari, and I. Dattner, "A guided FP-Growth algorithm for multitude-targeted mining of Big Data," 2018, *arXiv:1803.06632*.
[18] R. Abeysinghe and L. Cui, "Query-constraint-based association rule mining from diverse clinical datasets in the national sleep research resource," in *Proc. IEEE Int. Conf. Bioinf. Biomed.*, 2017, pp. 1238–1241.
[19] P. Fournier-Viger, E. Mwamikazi, T. Gueniche, and U. Faghihi, "MEIT: Memory efficient itemset tree for targeted association rule mining," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2013, pp. 95–106.
[20] C. Chand, A. Thakkar, and A. Ganatra, "Target oriented sequential pattern mining using recency and monetary constraints," *Int. J. Comput. Appl.*, vol. 45, no. 10, pp. 12–18, 2012.
[21] H. E. Chueh *et al.*, "Mining target-oriented sequential patterns with time-intervals," *Int. J. Comput. Sci. Inf. Technol.*, vol. 2, no. 4, pp. 113–123, 2010.
[22] C. Zhang, Z. Du, Q. Dai, W. Gan, J. Weng, and P. S. Yu, "TUSQ: Targeted high-utility sequence querying," *IEEE Trans. Big Data*, pp. 1–15, 2022, doi: 10.1109/TBDATA.2022.3175428.
[23] J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, "Targeted high-utility itemset querying," in *Proc. IEEE Int. Conf. Big Data*, 2021, pp. 5534–5543.
[24] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May/Jun. 2000.
[25] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Proc. Int. Conf. Database Theory*, Springer, 1999, pp. 398–416.
[26] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for Big Data," in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 111–118.
[27] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and J. Zhan, "Mining of frequent patterns with multiple minimum supports," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 83–96, 2017.
[28] B. E. Shie, P. S. Yu, and V. S. Tseng, "Mining interesting user behavior patterns in mobile commerce environments," *Appl. Intell.*, vol. 38, no. 3, pp. 418–435, 2013.
[29] C. Chu, V. S. Tseng, and T. Liang, "An efficient algorithm for mining temporal high utility itemsets from data streams," *J. Syst. Softw.*, vol. 81, no. 7, pp. 1105–1117, 2008.
[30] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and H. Fujita, "Extracting non-redundant correlated purchase behaviors by utility measure," *Knowl. Based Syst.*, vol. 143, pp. 30–41, 2018.
[31] V. S. Tseng, C. Wu, B. E. Shie, and P. S. Yu, "UP-Growth: An efficient algorithm for high utility itemset mining," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2010, pp. 253–262.
[32] H. Kim *et al.*, "Efficient list based mining of high average utility patterns with maximum average pruning strategies," *Inf. Sci.*, vol. 543, pp. 85–105, 2021.
[33] Q. H. Duong, P. Fournier-Viger, H. Ramampiaro, K. Nørvåg, and T. L. Dam, "Efficient high utility itemset mining using buffered utility-lists," *Appl. Intell.*, vol. 48, no. 7, pp. 1859–1877, 2018.
[34] W. Gan, S. Wan, J. Chen, C. M. Chen, and L. Qiu, "TopHUI: Top-k high-utility itemset mining with negative utility," in *Proc. IEEE Int. Conf. Big Data*, 2020, pp. 5350–5359.
[35] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and P. S. Yu, "HUOPM: High-utility occupancy pattern mining," *IEEE Trans. Cybern.*, vol. 50, no. 3, pp. 1195–1208, Mar. 2020.

[36] V. S. Tseng, C. W. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining top-k high utility itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 54–67, Jan. 2016.
[37] 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," *Knowl. Based Syst.*, vol. 104, pp. 106–122, 2016.
[38] W. Song, Z. Zhang, and J. Li, "A high utility itemset mining algorithm based on subsume index," *Knowl. Inf. Syst.*, vol. 49, no. 1, pp. 315–340, 2016.
[39] H. Ryang and U. Yun, "Indexed list-based high utility pattern mining with utility upper-bound reduction and pattern combination techniques," *Knowl. Inf. Syst.*, vol. 51, no. 2, pp. 627–659, 2017.
[40] J. Lee, U. Yun, G. Lee, and E. Yoon, "Efficient incremental high utility pattern mining based on pre-large concept," *Eng. Appl. Artif. Intell.*, vol. 72, pp. 111–123, 2018.
[41] U. Yun *et al.*, "Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases," *Future Gener. Comput. Syst.*, vol. 103, pp. 58–78, 2020.
[42] Y. Baek *et al.*, "Approximate high utility itemset mining in noisy environments," *Knowl. Based Syst.*, vol. 212, 2021, Art. no. 106596.
[43] J. C. W. Lin, W. Gan, T. P. Hong, and V. S. Tseng, "Efficient algorithms for mining up-to-date high-utility patterns," *Adv. Eng. Informat.*, vol. 29, no. 3, pp. 648–661, 2015.
[44] 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," *Adv. Eng. Informat.*, vol. 30, no. 2, pp. 109–126, 2016.
[45] W. Song, C. Zheng, C. Huang, and L. Liu, "Heuristically mining the top-k high-utility itemsets with cross-entropy optimization," *Appl. Intell.*, pp. 1–16, Jul. 2021, doi: 10.1007/s10489-021-02576-z.
[46] 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 Interdiscipl. Rev. Data Mining Knowl. Discov.*, vol. 8, no. 2, 2018, Art. no. e1242.
[47] W. Gan, J. C. W. Lin, H. C. Chao, S. L. Wang, and P. S. Yu, "Privacy preserving utility mining: A survey," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 2617–2626.
[48] R. Abeysinghe and L. Cui, "Query-constraint-based mining of association rules for exploratory analysis of clinical datasets in the national sleep research resource," *BMC Med. Informat. Decis. Mak.*, vol. 18, no. 2, pp. 89–100, 2018.

Jinbao Miao received the B.S. degree in network engineering from the Jiangxi University of Science and Technology, Ganzhou, China, in 2020. He is currently working toward the Graduation degree in cybersecurity with the College of Cyber Security, Jinan University, Guangzhou, China.

His research interests include data intelligence, data mining, and Big Data.

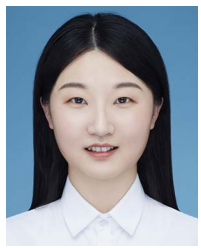
Shicheng Wan received the B.S. degree in computer science from Gannan Normal University, Ganzhou, China, in 2020. He is currently working toward the graduation degree in cybersecurity with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China.

His research interests include data intelligence, data mining, and Big Data.

Wensheng Gan (Member, IEEE) received the B.S. degree in computer science from South China Normal University, Guangdong, China, in 2013, and the Ph.D. degree in computer science and technology from the Harbin Institute of Technology (Shenzhen), Shenzhen, China, in 2019.

He is currently an Associate Professor with the College of Cyber Security, Jinan University, Guangzhou, China. His research interests include data mining, data intelligence, Big Data, and privacy. He has authored or coauthored more than 100 research papers in peer-reviewed journals (i.e., IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON CYBERNETICS, *ACM Transactions on Knowledge Discovery from Data*, *ACM Transactions on Data Science*, and *ACM Transactions on Management Information Systems*) and international conferences.

Dr. Gan is an Associate Editor for *Journal of Internet Technology*.



Jiayi Sun received the B.S. degree in information security from Qingdao University, Qingdao, China, in 2020. She is currently working toward the graduation degree in cybersecurity with the College of Cyber Security, Jinan University, Guangzhou, China.

Her research interests include data intelligence, data mining, and Big Data.

Jiahui Chen (Member, IEEE) received the B.S. degree from South China Normal University, Guangzhou, China, in 2009, and the M.S. and Ph.D. degrees from the South China University of Technology, Guangzhou, China, in 2012 and 2016, respectively.

He joined the National University of Singapore, Singapore, as a Research Scientist between 2017 and 2018. He is currently an Associate Professor with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China. His research interests mainly focus on public key cryptography, postquantum cryptography, and information security.