Towards Target High-Utility Itemsets

Jinbao Miao, Wensheng Gan*, Shicheng Wan, Yongdong Wu, Philippe Fournier-Viger

Abstract—For applied intelligence, utility-driven pattern discovery algorithms can identify insightful and useful patterns in databases. However, in these techniques for pattern discovery, the number of patterns can be huge, and the user is often only interested in a few of those patterns. Hence, targeted mining of high-utility itemsets has emerged as a key research topic, where the aim is to find a subset of patterns that meet a target pattern constraint instead of all patterns. This is a challenging task because efficiently finding tailored patterns in a very large search space requires a targeted mining algorithm. A first algorithm called TargetUM has been proposed, which adopts an approach similar to post-processing using a tree structure, but the running time and memory consumption are unsatisfactory in many situations. In this paper, we address this issue by proposing a novel list-based algorithm with pattern matching mechanism, named THUIM (Targeted High-Utility Itemset Mining), which can quickly match high-utility itemsets during the mining process to select the target patterns. Extensive experiments were conducted on different datasets to compare the performance of the proposed algorithm with state-of-the-art algorithms. Results show that THUIM performs very well in terms of runtime and memory consumption, and has good scalability compared to TargetUM.

Index Terms—applied intelligence, pattern discovery, target pattern, high-utility itemset.

I. INTRODUCTION

Frequent pattern mining (FPM) [1]–[4] is a research topic that has been well-studied in the last decades. FPM techniques focus on counting the support (number of occurrences) of patterns in a database to identify frequent ones. The mining result is not only intuitive, but also easy to understand. Hence, it has been applied to different kinds of databases, like quantitative transaction databases [5], [6], streaming databases [7], and time series databases [8]. However, only considering the support metric brings the problem that all items (data elements) in a pattern are considered as having the same relative importance. To address this issue, weighted pattern mining (WPM) was proposed [9]–[12]. In quantitative transaction databases, the concept of weights refers to some important factors, such as the unit profits of items. In this framework, even if some items have a low support, they are still viewed as valuable if they have large weights. Nevertheless, WPM techniques do not take the quantities of items into account. Thereby, it still cannot satisfy some practical needs. For example, market retailers are strongly interested in discovering itemsets that yield a high revenue, which depends on both unit prices and purchase quantities.

In view of this, a generic and superior framework named high-utility itemset mining (HUIM) [13], [14] was proposed. Each pattern has an internal utility (e.g., purchase quantity) and external utility (e.g., unit price). The utility metric attracts a lot of interest for two reasons: compared with FPM, UPM (Utility Pattern Mining) has the advantages of WPM; and compared with WPM, it considers the importance of distinct items more accurately. An itemset is called a high-utility itemset (abbreviated as HUI) if its real utility (importance) is no less than a user-specified minimum utility (simplified as minUtil) threshold; otherwise, it is a low-utility itemset, which is uninteresting. After more than a decade of development, mining HUIs in quantitative transaction databases has been utilized in many applications such as user behavior analysis [15], website click-stream analysis [16], [17], biomedical analysis [18], and cross-marketing analysis [19], [20]. In general, discovering HUIs is a difficult task since the downward-closure property [5] does not hold. In other words, it is not guaranteed that subsets of an HUI are HUIs. Another challenge is how to effectively prune the search space and efficiently capture all HUIs without missing any, for a very large search space, especially when databases contain lots of long transactions. As a solution to the first problem, Liu et al. [13] proposed an overestimation upper-bound named transaction weighted utilization (abbreviated as TWU). For brevity, this new upper-bound satisfies the downward-closure property, which means a low-TWU itemset cannot have high-TWU (and also high-utility) supersets. For the second issue, many researchers have designed improved utility data mining algorithms. Many utility mining studies [19], [21]–[24] apply the TWU to facilitate the mining process and use efficient data structures and pruning strategies to improve the performance of algorithms. There are already several algorithms in the high-utility itemset mining (HUIM) literature, such as list-based (i.e., HUI-Miner [23], FHM [25], FHN [22], and HUOPM [26]), tree-based (i.e., IHUP [27], UP-Growth [19] and MU-Growth [28]), and projection-based (i.e., EFIM [24]) algorithms.

However, intuitively, most of HUIM algorithms aim to discover as much hidden information as possible from databases. However, in some cases, the requirements are the opposite. For instance, in the marketing analysis field, retailers are interested in goods which yield a high profit, and hence to find as much HUIs as possible. On the contrary, customers are more concerned about only some of the HUIs because of affordability and interest. Therefore, mining user-specified
itemsets was proposed as a new interesting task, named targeted utility mining (abbreviated as TaUM) [29], [30]. To our best knowledge, there are some studies about target-based pattern mining such as target-oriented frequent itemset querying [31], target-based association rule mining [32], [33] and sequential pattern querying [34]–[36]. However, there is little information in the literature about TaUM. This paper focuses on the problem of querying special HUIs which contain target itemsets. For the TaUM task, the state-of-the-art algorithm is TargetUM [29], [30], which utilizes the utility-list data structure and a querying trie to discover target high-utility itemsets. Nevertheless, the TargetUM algorithm faces the problem of high memory consumption. This is in part because items with low TWU values are considered for generating high-level itemsets.

To address these issues, we propose a novel algorithm named THUIM (Targeted High-Utility Itemset Mining) as well as a compact data structure for efficiently mining the target HUIs in quantitative transaction databases. The major contributions of this study are:

- We first define the concept of pattern matching mechanism for targeted mining/searching interesting patterns. To the best of our knowledge, we are the first to propose a one-phase algorithm to address the very challenging problem of targeted utility mining.
- The utility-list is extended as a new data structure, which is utilized for targeted mining of high-utility itemsets without scanning the database multiple times.
- We propose an efficient one-phase algorithm, namely THUIM, that can significantly reduce the search space using the matching mechanism.
- Extensive experiments were done on real and synthetic datasets. Results show that the proposed THUIM algorithm is efficient for the problem of targeted mining of high-utility itemsets in large-scale databases.

The rest of the paper is organized as follows. Related work is described in Section II. Some key preliminaries are introduced in Section III. The details of the proposed algorithm are presented in Section IV. Then, extensive experiments and a comprehensive analysis of results are presented in Section V. Finally, a conclusion is drawn and future work is discussed in Section VI.

II. RELATED WORK

In this section, we briefly review some studies about frequency-based itemset mining (FIM), utility-based itemset mining (UIM), and targeted itemset mining methods.

A. Frequency-based itemset mining

In the FIM domain, Apriori [5] is the most famous level-wise algorithm. It first generates low-level (short) candidates, and then iteratively constructs high-level (longer) frequent itemsets. Apriori applies the Apriori property (also known as downward-closure property) to reduce the search space, which states that an infrequent itemset cannot be a subset of a frequent itemset. Generating too many candidates and scanning the database many times are obvious drawbacks of Apriori that lead to poor performance in some situations. Some studies proposed improvements [13], [37], [38] to address these issues. Han et al. [6] proposed a tree-based FIM algorithm named FP-Growth, using a compact and extended prefix-tree structure named FP-tree. The FP-Growth algorithm reorganizes all the data from a database in support descending order and insert it into an FP-tree, which makes the data highly compressed and eliminates the need of scanning the database multiple times. In recent years, some researchers have done additional improvements [1], [39] because FIM remains a popular problem. However, most of the FIM algorithms have the critical shortcoming that the frequency metric does not capture the relative importance of items, such that for example, diamonds brings a higher profit than a pen, although they have a lower selling frequency. To address this problem of FIM, Cai et al. [9] proposed using weights to measure the relative importance of each item. In this framework, items have weights, and even if some items are infrequent, they can also be discovered if they have large weights. After that, several weighted itemset mining studies [10], [11], [40] were done, and it has become an important variant of FIM.

B. Utility-based itemset mining

Although the weighted pattern mining (WPM) framework [11], [12] has some applications, weights remain a solution that is often too simple and cannot fully capture the importance of patterns. For instance, WPM cannot address the requirements of users who are interested in mining itemsets that yield a high profit. In view of this, high-utility itemset mining (HUIM) [18] has emerged as an important topic in the last decades. Utility represents how much an item is profitable in quantitative transaction databases, and it has good interpretability. If the utility of an itemset is no less than a user-specified minimum utility minUtil threshold, it is deemed to be a high-utility itemset (abbreviated as HUI); otherwise, it is said to be a low-utility itemset, which is uninteresting. Discovering HUIs in a database is not an easy task since the downward-closure property [5] in FIM does not hold in utility mining. It is difficult to estimate if the supersets of a HUI are HUIs or not. A naive approach is to count the utilities of all itemsets by scanning the database, and then keep the HUIs. While obviously, this approach suffers from the problem of a very large search space, especially for long transactions. Hence, Liu et al. [13] proposed the transaction-weighted utilization (TWU) to solve this issue and an algorithm. This latter first calculates the TWU of all items by scanning the database, and then keep candidates that may be HUIs. In the second step, the algorithm identifies the real HUIs from those candidates with an additional database scan. Generally, we can classify early HUIM algorithms such as [6], [13], [19], [27] as two-phase-based algorithms.

Although two-phase algorithms effectively reduce the search space and efficiently find the complete set of HUIs for HUIM, runtime and memory consumption remain a major issue. As an alternative to two-phase-based algorithms, single-phase-based algorithms have emerged as an important topic. HUI-Miner [23] is the first work for mining HUIs without candidate
generation. It utilizes a compact utility-list structure to avoid scanning a database repeatedly. An intersection operation for low-level utility-lists allows obtaining the utility information of all HUIs. Many list-based algorithms have been developed for HUIM such as FHM [25] and FHN [22]. The worst shortcoming of list-based algorithms is that lists may consume much memory when dealing with huge databases containing thousands of items. Zida et al. [24] proposed the EFIM algorithm, which is inspired by the LCM algorithm [21] for FIM. EFIM adopts a depth-first search mechanism to process each itemset in linear time and space. Moreover, it integrates a novel array-based utility counting technique to calculate two new upper bounds (local-utility and subtree-utility) to reduce search space effectively. With these efficient techniques, EFIM is in general two to three orders of magnitudes faster than previous work. TOPIC [21] can handle itemsets with negative profit, defined as \( q(x, T_j) \), which is the quantity associated with \( x \) in the transaction \( T_j \) (1 \( \leq j \leq m \)); 2) the external utility (i.e., unit profit), defined as \( p(x) \), which is a positive value, for \( x_i \) in \( D \). A simple quantitative transaction database is given in Table I which is composed of seven transaction data, and include seven items \{a, b, c, d, e, f, g\}. The external utilities of these itemsets are \( p(a) = $3 \), \( p(b) = $5 \), \( p(c) = $1 \), \( p(d) = $5 \), \( p(e) = $3 \), \( p(f) = $4 \), and \( p(g) = $2 \), respectively.

**TABLE I: A quantitative transaction database.**

| TID | Transaction | Quantity |
|-----|-------------|----------|
| \( T_1 \) | \{d, e, f\} | \{2, 5, 6\} |
| \( T_2 \) | \{a, b, d, f\} | \{3, 8, 7, 1\} |
| \( T_3 \) | \{a, e, b, f\} | \{5, 3, 4, 3\} |
| \( T_4 \) | \{a, b, c, d, f\} | \{6, 4, 1, 4, 2\} |
| \( T_5 \) | \{b, e, d, f\} | \{5, 3, 7, 6, 3\} |
| \( T_6 \) | \{a, b, d, f, g\} | \{8, 7, 2, 1, 1\} |
| \( T_7 \) | \{a, b, c, e, f, g\} | \{1, 1, 6, 5, 4, 2\} |

**Definition 3.1 (utility of itemset):** Given an item \( x \) in a transaction \( T_{tid} \) is denoted as \( U(x, T_{tid}) = q(x, T_{tid}) \times p(x) \) (where \( q(x, T_{tid}) \) represents the number of occurrences in \( T_{tid} \), that is the internal utility). The real utility of \( x \) in a database \( D \) is the sum of the utility values, denoted as \( U(x) = \sum_{x \in T_{tid} \cap D} q(x, T_{tid}) \times p(x) \). Then, the utility of an itemset \( X \) is defined as \( U(X) = \sum_{x_i \in X} U(x_i, T_{tid}) \).

For example, in Table I \( U(e, T_1) = q(e, T_1) \times p(e) = 5 \times 3 = $15, U(e, T_2) = $12, U(e, T_5) = $21 \) and \( U(e, T_7) = $15 \). Therefore, the utility of item \( e \) in \( D \) is \( U(e) = T_1 + T_3 + T_5 + T_7 = $15 + $12 + $21 + $15 = $63 \). Setting itemset \( X = \{f, g\} \), \( U(X) = U(f) + U(g) = q(f, T_5) \times p(f) + q(f, T_6) \times p(f) + q(g, T_7) \times p(g) + q(g, T_6) \times p(g) + q(g, T_5) \times p(g) \times p(f, T_5) \times p(f, T_6) = $24 + $4 + $16 + $6 + $2 + $4 = $56 \). And if \( X = \{e, g\} \), then \( U(e, g) = $27 \).

**Definition 3.2 (transaction-weighted utilization [23]):** The utility of a transaction \( T_{tid} \in D \) is the sum of the utility of all items contained in it, denoted as \( TWU(T_{tid}) = \sum_{x_i \in T_{tid}} U(x_i, T_{tid}) \). The transaction-weighted utilization of itemset \( X \) is defined as \( TWU(X) = \sum_{X \subseteq T_{tid}} TWU(T_{tid}) \).

\( TWU \) is a useful upper bound and can help cut off unpromising items in advance. Due to the space limitation, the proof and details can be found in Ref. [23]. For example, in Table I \( TWU(T_1) = U(d, T_1) + U(c, T_1) + U(f, T_1) = 2 \times $5 + 5 \times $3 + 6 \times $4 = $49 \). Therefore, \( TWU(a) = TWU(T_2) + TWU(T_3) \).
+ $TU(T_1) + TU(T_2) + TU(T_3) = $88 + $54 + $71 + $75 + $49 = $337, and also we have $TWU(e) = $49 + $54 + $91 + $49 = $243.

Definition 3.3 (high-utility itemset [23, 18]): The minimum utility threshold (abbreviated as minUtil) is a user-specified parameter. If the utility of an itemset $X$ is no less than minUtil, where $U(X) \geq \text{minUtil}$, we call $X$ a high-utility itemset (abbreviated as HUI); otherwise, it is a low-utility itemset, which we are not interested in. In this paper, minUtil ($\sigma$) is used to discover target HUI directly.

Definition 3.4 (target high-utility itemset [29, 30]): The target itemset is a user-specified set of items $T^i \preceq <x_1, x_2, \ldots, x_n>$ ($i \in [1, n] \land x_i \in I$) which the user(s) are interested in. It should be pointed out that $|T^i| \geq 1$. Given a high utility itemset $X$, and some target itemset $T'$, if $T' \subseteq X$, we say that $X$ is a target high-utility itemset (abbreviated as THUI).

B. Problem statement

In general, HUI-Miner algorithms aim to find a complete set of HUIs in quantitative transaction databases. However, in real applications, customers always need a part of those HUIs, and sometimes they may have different requirements at different times. For example, a user query may seek all HUIs containing $X$ which can quickly and efficiently mine all high-utility itemsets.

IV. THE THUIM ALGORITHM

It was observed that the prior TargetUM algorithm has relatively poor performance, and can have long runtime and high memory consumption for dense as well as large datasets. Therefore, a more efficient algorithm is essential for the targeted pattern mining problem. In this section, we propose the THUIM algorithm, which is based on the HUI-Miner algorithm [23]. By integrating an efficient targeted pattern matching mechanism in HUI-Miner, targeted high-utility itemsets can be mined directly. THUIM is a more efficient than TargetUM as it will be demonstrated.

A. Matching mechanism

The HUI-Miner algorithm uses the utility and remaining utility as judgment conditions to reduce the search space, which can quickly and efficiently mine all high-utility itemsets. For this reason, we introduce a matching mechanism based on HUI-Miner in the proposed THUIM algorithm. Assume a total order $\prec$ on items, that is the $TU$ ascending order. That is to say for any two items $x_i$ and $x_j$ if $x_j$ is after $x_i$ we have $TU(x_j) \geq TU(x_i)$. For example, from Table I we have $TU(a) \geq TU(c)$, and after sorting, $a$ is after $c$.

TABLE II: Transaction-weighted utility.

| Item | a   | b   | c   | d   | e   | f   | g   |
|------|-----|-----|-----|-----|-----|-----|-----|
| TU   | $337$ | $428$ | $120$ | $374$ | $243$ | $477$ | $215$ |

Definition 4.1 (serial number on items): According to the $TU$-ascending order, a unique serial number is assigned to each item. More precisely, consecutive integer serial numbers are assigned to all items having a $TU$ that is no less than $\sigma$ in increasing order starting from 1, denoted as $sn(x_i)$ that is $x_i \in I \land TU(x_i) \geq \sigma$.

For the running example, we get that $TU(c) = $120 $TU(g) = $215 $TU(e) = $243 $TU(a) = $337 $TU(d) = $374 $TU(b) = $428 $TU(f) = $477. Thus, the total order $\prec$ on items is $c < g < e < a < d < b < f$. If we set $\sigma = $130, we have $sn(g) = 1, sn(e) = 2, sn(a) = 3, sn(d) = 4, sn(b) = 5$, and $sn(f) = 6$. The matching mechanism in THUIM is implemented using the serial numbers. According to Definition 4.1 for each item $x_i$ such that $TU(x_i) \geq \sigma$, there is a unique serial number. This is to be able to distinguish between items in the case where multiple items have the same $TU$. To facilitate pattern matching, preprocessing is required for the target pattern $T'$, so that it is in line with the overall order of the algorithm that is the $TU$-ascending order.

Definition 4.2 (matching): For two itemsets $X$ and $Y$, if $Y$ is a subsequence of $X$, that is to say $Y \subseteq X$, then $Y$ matches with $X$ and $X$ is matched by $Y$. If $Y \subseteq X$, then $Y$ and $X$ mismatch, and if $Y$ is partly contained in $X$ and the other part is indeterminate, then $X$ and $C$ can be matched.

For example, consider the itemsets $X = \{e, b, f\}$ and $Y = \{b, f\}$. $Y \subseteq X$, thus $X$ is matched by $Y$. Thus, if the user sets $T^i = \{e, f\}$, $X$ is matched by $T^i$.

B. Utility-list structure

The HUI-Miner algorithm mainly uses the utility-list data structure to mine the full set of HUIs [23]. In the THUIM algorithm, the utility-list is still a very important component. The utility-list contains multiple tuples, each tuple containing three fields which are the id of the transaction (tid) containing the itemset $X$, the utility of $X$ in that transaction, and the remaining utility [23] of $X$ in the current transaction, that is $(tid, iu, ru)$. A utility-list $UL$ has the form $<(tid_1, iu_1, ru_1), (tid_2, iu_2, ru_2), \ldots, (tid_q, iu_q, ru_q)>$. All the utility-lists of 1-itemsets are shown in Fig. I.

The $(k+1)$-itemset(s) can be obtained from the $k$-itemset(s) by merging the $UL$ lists. Suppose the utility-list of itemset $X_a$ is $UL_a$ and the utility-list of the itemset $X_b$ is $UL_b$ when joining two $UL$ lists ($UL_a$ and $UL_b$), the first step is to ensure that the same transaction tid exists in $UL_a$ and $UL_b$. Then, sum the utility and take the remaining utility of the item corresponding to the largest $TU$. Note that when summing the utility, the utility corresponding to the common prefix
Finally, if the items can be matched as a match on Fig. 3 (a), the items to be matched are expanded. For the itemsets that follow must not contain TWU, we have $sn(a) > sn(e)$, and the extended match will be terminated. A concrete example will be shown in the algorithm section of this paper.

The utility-list is composed of multiple triples $(<tid, iu, ru>)$. Counting the sum of $iu$ of all tuples in a utility-list can get the exact utility of itemset $X$ (sumutils). Similarly, counting the sum of $ru$ of all tuples in the utility-list can get the maximum utility of itemset $X$ that can be extended (sumRutils). Therefore, computing the sum of sumutils and sumRutils yields an upper bound on utility ($\Theta$). If $\Theta \geq \sigma$, it is possible to obtain a high-utility itemset by extending the itemset $X$. Otherwise, it means that any high-utility itemsets cannot be obtained by extending the itemset $X$.

C. Efficient pruning strategies

In TargetUM [29, 30], when performing an upward query, the target pattern $T'$ and the high-utility itemsets are matched one by one. And the item's TWU determines whether the target item can be successfully matched. However, different datasets have variability, i.e., different items may have the same TWU. Thus, it is also necessary to specifically determine whether items are the same when the TWU is the same. To facilitate comparison, we introduce the concept of serial number (Definition 4.1). The pruning strategy is designed by serial number to greatly improve the efficiency of THUIM for mining the target high-utility itemsets.

**Pruning strategy:** when extending the search for a target high-utility itemset $X$, if the serial number of the extended item is larger than that of the current item to be matched in $T'$, it means that there is a mismatch, which means that $T$ must not be included in $X$. That is said for two items $x_i (x_i \in T')$ and $x_1 (x_1 \in X \land i == |X|)$ which need to be matched, if $sn(x_i) > sn(x_1)$, then the search will be terminated, and the items that follow must not contain $T'$.

**Proof:** All the single items and $T'$ are sorted in TWU ascending order, thus according to Definition 4.1 if $TWU(x_i) \succ TWU(x_j)$, we have $sn(x_i) > sn(x_j)$. For the item $x_i$ in $T'$ ($0 \leq t < |T'|$), if $sn(x_i) \succ sn(x_1)$, it indicates a successful match, and the next recursion will match $x_{i+1}$ and $x_{t+1}$. If $sn(x_i) \preceq sn(x_1)$, the match fails and the next recursion will match $x_{i+1}$ and $x_{t+1}$. Thus, if $sn(x_i) > sn(x_1)$, for any item $x_j (j > i)$, we will have $sn(x_i) > sn(x_j)$.

For example, from Fig. 3 we can get that $T' = \{e, f\}$ and mat which contains all the items that can be expanded. For the Fig. 3(a), the items to be matched are $g$ (mat) and $e$ ($T'$), have $sn(g) < sn(e)$, next will match $e$ (mat) and $e$ ($T'$). Note that for the next item to be matched is uncertain, thus for $e$ ($T'$), mat may or may not be included. Considers Fig. 3(b), have $sn(e) = sn(e)$, and next will match $a$ (mat) and $f$ ($T'$). Finally, if the items can be matched as $a$ (mat) and $e$ ($T'$) in Fig. 3(c), have $sn(a) > sn(e)$, and the extended match will be terminated. A concrete example will be shown in the algorithm section of this paper.

D. Proposed target query algorithm

In this subsection, the details of the proposed querying algorithm for mining target high-utility itemsets are discussed. Three main sub-functions are included, the data processing function (Algorithm 1), the itemset mining function (Algorithm 2), and the utility-list construction function (Algorithm 3).

**Algorithm 1: THUIM ($D, \sigma, T'$) procedure**

1. scan the database $D$ to get the basic data format and count the TWU values of all 1-itemsets, i.e. $TWU(x_i) (x_i \in I)$;
2. scan $I$ to get the set $I'$ where $x_i \in I \land TWU(x_i) \geq \sigma$;
3. scan and evaluate $T'$;
4. if $TWU(x_i) < \sigma$ (for all $x_i \in T'$) then return null;
5. sort $I'$ by the TWU ascending order $\prec$;
6. sort $T'$ by the TWU ascending order $\prec$;
7. scan $I'$ to get the order set of mapItemToOrder;
8. second scan of $D$, and sort each transaction by the TWU ascending order $\prec$ and construct the utility-list denoted as $UList$ for each item in $I'$;
9. call $THUIM(\phi, UList, \sigma, 0, T', \text{mapItemToOrder})$.

Consider Algorithm 1 which has three input parameters, which are the database ($D$), minimum utility threshold ($\sigma$) and target pattern ($T'$). The database ($D$) needs to be scanned twice, the first time to obtain the upper bound of the utility of all 1-itemsets (TWU), and filter out the 1-itemsets that do not
Fig. 2: Some ULists of k-itemsets with $T' = \{e, f\}$.

![Figure 2](image)

Fig. 3: Match with serial number for $T' = \{e, f\}$.

![Figure 3](image)

Algorithm 2: THUIM procedure

**Input:** $UL$\textsubscript{OfP}: the utility-list of itemset $P$; $UL$: the set of utility-lists of extensions of itemset $P$; the minimum utility threshold ($\sigma$); index: points to the current item in $T'$ that needs to be matched; $\beta$: the size of $T'$; mapItemToOrder: mapping of items to serial numbers.

**Output:** all the target high-utility itemsets (THUIs).

```plaintext
for each element of utility-list $x$ in $UL$ do
    initialize currentIndex ← index;
    initialize oriOrder ← 0;
    initialize patOrder ← 0;
    if currentIndex < $\beta$ then
        oriOrder ← the serial number of x.item in mapItemToOrder;
        patOrder ← the serial number of T'$[\text{index}]$ in mapItemToOrder;
        if oriOrder == patOrder then
            currentIndex increasing;
        end
    end
    if currentIndex < $\beta$ AND patOrder < oriOrder then
        break;
    end
    if x.sumutils $\geq \sigma$ AND currentIndex $\geq \beta$ then
        new THUIs ← THUIs $\cup x$;
    end
    if x.sumutils $+ x$.sumRutils $\geq \sigma$ then
        new utility-list newUList $\leftarrow$ null;
        for each element of utility-list $y$ after $x$ in $UL$ do
            newUList $\leftarrow$ newUList $\cup$ construct($UL$, $x$, $y$);
        end
    end
    call THUIM($UL$, newUList, $\sigma$, currentIndex, $|T'|$, mapItemToOrder).
end
```

Algorithm 2 shows the main process of targeted utility mining, taking six parameters as input, which are two utility-lists $UL$\textsubscript{OfP} and $UL$, where $UL$ is an extension of $UL$\textsubscript{OfP}, the minimum utility threshold ($\sigma$), the pointer of $T'$ (index), the size of $T'$ and the serial number set mapItemToOrder. Algorithm 2 performs item matching for each recursive execution, which are $x$.item and $T'$'[\text{index}]$ (lines 1-4). Firstly, we need to get the serial number with matching items oriOrder and patOrder. And if oriOrder equals patOrder, it means that items $x$.item and $T'$'[\text{index}]$ can be matched, and then the pointer index is moved back one position (lines 5-11). For the items to be matched, if patOrder $<$ oriOrder, the traversed itemset is a mismatch with $T'$ according to the pruning strategy, and the current recursion terminates (lines 12-14). If $x$.sumutils $\geq \sigma$ and currentIndex $\geq \beta$, a new THUI can be discovered. And if $x$.sumutils $+ x$.sumRutils $\geq \sigma$, extended utility-lists are built. Finally, there is a recursive call to Algorithm 2 (lines 15-24). Algorithm 3 shows how to construct a utility-list. To construct a utility list from the utility lists of some itemsets $X$ and $Y$ to generate the utility list of $XY$, it must be ensured that $X$ and $Y$ have an element $e$ with the same $tid$, that is to say, the intersection of $X$ and $Y$ is not the empty set. If there is no such element $e$, it means that the
Algorithm 3: construct procedure

```
Input: UListOfP, UListOfPX, UListOfPY: the utility-lists of itemset P, PX and PY.
Output: the utility-list of itemset PXY (UListOfPXY).
1 UListOfPXY ← null;
2 for each element ex in UListOfPX do
3     ey ← use binary search to find an element e such that e.tid == ex.tid in UListOfPY;
4     if ex.tid == ey.tid then
5         if UListOfP == null then
6             new ey ← <ey.tid, ex.iutils + ey.iutils, ey.rutils>;  
7                 else
8                     e ← use binary search to find ex from UListOfP;
9                     new ey ← <ey.tid, ex.iutils + ey.iutils - e.iutils, ey.rutils>;
10                end
11            UListOfPXY ← UListOfPXY ∪ ey;
12        end
13    end
14 return UListOfPXY
```

itemset $XY$ does not exist in the dataset. For implementation details, one can refer to HUI-Miner [23].

In Algorithm 3 it is crucial to match oriPrder and patOrder. Assume that $\sigma = 130$ and $T^* = \{e, f\}$. In Fig. 1 if $g$ and $d$ are joined to generate $gd$, then we need to compare $d$ with the first item $e$ in $T^*$ (note that $T^*$ is sorted). By looking up mapItemToOrder we get $sn(d) = 4$ and $sn(e) = 2$, thus have $sn(d) > sn(e)$. This is a mismatch because $e$ doesn’t appear in its extension anyway. If $e$ and $b$ are joined to generate $eb$, then we need to compare $b$ with the second item $f$ in $T^*$ ($e$ must match successfully when building 1-itemsets). We get $sn(b) < sn(f)$. This means that item $f$ may match for extended itemsets later, thus this itemset needs to be preserved. In Fig. 2 if $eb$ and $ef$ are joined to generate $ebf$, the match is successful.

V. Performance Evaluation

In this section, we evaluate the performance of the THUIM algorithm in detail and compares it with that of TargetUM [29], [30]. This latter constructs a TP-tree during the mining process, which makes it necessary to take this memory into account each time that the dataset is mined. Obviously, the minimum threshold ($\sigma$) is inversely proportional to the number of itemsets found, and there are fewer results when the threshold is larger. Therefore, when $\sigma$ is set to a relatively small value, TargetUM runs out of memory and the algorithm terminates. THUIM uses a pattern matching mechanism instead of searching for a complete result set, and it directly mines the target high-utility itemsets. To a certain extent, THUIM can discard itemsets that are not the target itemsets in advance, which improves its performance.

A. Experimental setup

Experiments were conducted on a PC having an AMD Ryzen 5 3600 6-Core Processor 3.60 GHz, 8 GB of memory and 64-bit Microsoft Windows 10. TargetUM and THUIM are implemented in Java. We make all source code available at GitHub [https://github.com/DSI-Lab1/TargetUM](https://github.com/DSI-Lab1/TargetUM)

Six datasets were selected in this experiment, including four real-life datasets and two synthetic datasets, namely chess, retail, BMSPOS2, ecommerce, T10I4D100K, and T40I10D100K. For each dataset, some of its basic characteristics are summarized in Table IV. In this experiment, $\sigma$ is set to specific values, thus knowing the total utility of the dataset in advance can help us determine the range of threshold values and plays a very important role.

BMSPOS2, retail, chess and ecommerce are four real-life datasets. BMSPOS2 contains several years of point-of-sale data from a major electronics retailer. Retail is transactions from a retail store. Chess is compiled from the UCI chess dataset, and ecommerce contains all transactions that occurred between January 12, 2010, and September 12, 2011 that took place in a UK registered online store. T10I4D100K and T40I10D100K are synthetic datasets generated by the IBM generator, which are relatively stable and are usually used to evaluate the scalability of algorithms. Note that for the external utility of the dataset, only ecommerce has real external utility values, and the external utility of other datasets are randomly generated.

| Dataset       | Trans | Items | MaxLen | AvgLen | TotalUtility |
|---------------|-------|-------|--------|--------|--------------|
| chess         | 3196  | 75    | 37     | 37.0   | 2,156,659    |
| retail        | 88162 | 16468 | 76     | 10.3   | 362,481,272  |
| BMSPOS2       | 515366| 1656  | 164    | 6.5    | 1,301,704,112|
| chess         | 14975 | 3468  | 29     | 11.6   | 49,701,375   |
| ecommerce     | 100000| 870   | 29     | 10.1   | 1,385,548,246|
| T40I10D100K   | 100000| 942   | 77     | 39.6   | 1,550,463,496|

B. Efficiency analysis

To evaluate the performance of THUIM, it is compared with that of TargetUM. It is clear that TargetUM and THUIM produce the same results for the same dataset, minimum utility threshold ($\sigma$) value, and target pattern ($T^*$). Therefore, this point will not be the focus of this comparison but instead three aspects, which are the running time, memory consumption, and number of candidate itemsets that are generated. As far as we know, as a tree-based algorithm, TargetUM consumes a lot of memory when constructing the TP-tree. Considering the limitations of the experimental equipment, the total utility value of each dataset was calculated so as to set an appropriate utility threshold value that ensures the actionability of experimental results. At the same time for this experiment, the
size of $T'$ is set to three to five items, for retail, $T' = \{988, 990, 991, 998, 1003\}$, for ecommerce, $T' = \{21844, 23052, 23166, 8501412\}$, for T10I4D100K, $T' = \{85, 447, 859\}$, for BMSPOS2, $T' = \{11, 23, 36\}$, for chess, $T'' = \{48, 66, 70, 72\}$, and for T40I10D100K, $T' = \{521, 872, 933\}$.

Fig. 4 shows the running times of THUIM and TargetUM on six datasets. As can be seen from the figure, THUIM always has smaller running times than TargetUM, and runtimes decrease linearly as $\sigma$ is increased. There are two main reasons why THUIM has small runtimes. First, THUIM is more efficient than TargetUM as THUIM can mine the target high-utility itemsets directly, whereas TargetUM requires the construction of a TP-tree first. Second, to ensure that TargetUM can produce some results, $\sigma$ is set to relatively large values. Despite that, it can be seen in Fig. 4 that for some datasets such as retail, ecommerce, T10I4D100K, BMSPOS2 and chess, TargetUM has no results for some threshold values. This is because TargetUM ran out of memory when constructing the TP-tree. The difference in running time between TargetUM and THUIM ranges from tens to thousands of times, which is due to the fact that TargetUM mines all high-utility itemsets that satisfy $\sigma$ when constructing the TP-tree, which wastes a lot of time. On the other hand, THUIM can directly discover the target high-utility itemsets $x$ using its pattern matching mechanism, which saves a lot of time.

Memory consumption was monitored in the experiment. Results are shown in Fig. 5. It can be seen that the memory consumption of THUIM and TargetUM do not increase linearly as the threshold value is decreased. However, from an overall perspective, the memory consumed by TargetUM is much larger than that of THUIM. This is due to the fact that when constructing utility-lists, THUIM with the pattern matching mechanism will filter out utility-lists that do not contain target items, thus reducing the construction of utility lists. On the other hand, TargetUM first finds the full set that satisfies $\sigma$, and then construct a large number of utility-lists. TargetUM consumes much memory to store the utility-lists to find the target high-utility itemsets. Therefore, THUIM has a great advantage in terms of memory consumption.

When THUIM constructs a utility-list, the utility and the remaining utility are calculated at the same time. This process based on the HUI-Miner algorithm avoids multiple database scans to mine high utility itemsets in one phase. However, the construction of utility-lists can also be regarded as candidates. Thus, in a sense, the fewer utility-lists are constructed during the mining process, the more efficient the algorithm is. And the efficiency of the algorithm should be improved as much as possible while guaranteeing the completeness of the results. To further verify the efficiency of THUIM, the number of generated candidates was counted. It is shown in Fig. 5. As can be seen in that figure, the number of candidates gradually decreases, showing a linear change as $\sigma$ is increased. Besides, THUIM generates far less candidates than TargetUM, thanks to its pattern matching mechanism. Therefore, in summary, THUIM is more efficient than TargetUM.

C. Scalability analysis

In previous experiments, we assessed the performance of THUIM on different datasets for different $\sigma$ values. Here, we further analyze the impact of the sorting strategy and scalability for the number of transactions. For this experiment, we have selected retail, T10I4D100K and T40I10D100K as the test datasets, which are all sparse datasets and are appropriate for scalability experiments.

Processing order of items. The processing order of items for a dataset, which is used for the sorting strategy, plays a very important role in the mining process of THUIM. That is to say, choosing an appropriate sorting order can greatly improve the performance of the algorithm. THUIM adopts the TWU-ascending order. It should be clear that in THUIM, the target pattern is also sorted according to that same order, so changing the sorting strategy has no effect on the output of THUIM. We therefore measured the effects of the sorting strategy on runtime, candidate generation and memory consumption, and compared three different sorting strategies, namely the TWU-ascending order, the lexicographic order and the TWU-descending order, denoted as THUIM$_{twuas}$, THUIM$_{lex}$ and THUIM$_{nude}$. Figure 7 shows the experimental results on the retail dataset for a fixed $T' = \{988, 990, 991, 998, 1003\}$. As can be seen in that figure, THUIM$_{twuas}$ has excellent performance. Besides, THUIM$_{lex}$ performs better than THUIM$_{nude}$. Experimental results show that choosing THUIM$_{twuas}$ can greatly reduce the number of utility-lists built and reduce runtime and memory usage.

Length of datasets. To evaluate the scalability of THUIM, the datasets T10I4D100K and T40I10D100K were used, both of which have 100,000 transactions. For each dataset, six subsets were created, containing 10K (i.e., 10,000) more transactions than the previous one. The characteristics of these subsets with 50K, 60K, 70K, 80K, 90K, and 100K transactions are shown in Table IV. For T10I4D100K, parameters were set as $\sigma = 1000$ and $T' = \{85, 447, 859\}$, and for T40I10D100K, parameters are $\sigma = 30000$ and $T' = \{390, 464, 515, 611, 922\}$. Experiments were carried out to evaluate three aspects: runtime, candidate generation, and memory consumption. The detailed experimental results are shown in Fig. 8. As can be seen in Fig. 8, the performance changes linearly as the amount of data (10K) increases. Besides, the setting of the target patterns has a certain impact on the performance of THUIM. In summary, compared with TargetUM, THUIM is a quite efficient algorithm, which can quickly and effectively discover the target high-utility itemsets.

VI. Conclusion

This paper proposed the THUIM algorithm, which combines a target pattern with the HUI-Miner algorithm using a pattern matching mechanism. Targeted pattern mining is a recently proposed problem, and only the TargetUM algorithm was available. While TargetUM suffers from many limitations, and its performance is impaired by those especially for dense datasets and large datasets. This may result in problems such as out of memory errors and failing to produce the expected results. THUIM adopts a matching mechanism to obtain serial
Fig. 4: Runtimes for varied $\sigma$ values and a fixed $T'$ (a) Retail ($T' = \{988, 990, 991, 998, 1003\}$). (b) ecommerce ($T' = \{21844, 23052, 23166, 8501412\}$). (c) T10I4d100K ($T' = \{85, 447, 859\}$). (d) BMSPOS2 ($T' = \{11, 23, 36\}$). (e) chess ($T' = \{48, 66, 70, 72\}$). (f) T40I10D100K ($T' = \{521, 872, 933\}$)

Fig. 5: Memory when varied $\sigma$ is varied for a fixed $T'$ (a) Retail ($T' = \{988, 990, 991, 998, 1003\}$). (b) ecommerce ($T' = \{21844, 23052, 23166, 8501412\}$). (c) T10I4d100K ($T' = \{85, 447, 859\}$). (d) BMSPOS2 ($T' = \{11, 23, 36\}$). (e) chess ($T' = \{48, 66, 70, 72\}$). (f) T40I10D100K ($T' = \{521, 872, 933\}$)
Fig. 6: Candidates under varied $\sigma$ with a fixed $T'$ (a) Retail ($T' = \{988, 990, 991, 998, 1003\}$). (b) ecommerce ($T' = \{21844, 23052, 23166, 8501412\}$). (c) T10I4D100K ($T' = \{85, 447, 859\}$). (d) BMSPOS2 ($T' = \{11, 23, 36\}$). (e) chess ($T' = \{48, 66, 70, 72\}$). (f) T40I10D100K ($T' = \{521, 872, 933\}$).

Fig. 7: Different sorting strategies are compared for varied $\sigma$ values on the same retail dataset and a fixed $T' = \{988, 990, 991, 998, 1003\}$ (a) Runtime comparison. (b) Candidate comparison. (c) Memory consumption comparison.

Fig. 8: Influence of dataset size (with a 10K increment) for a fixed $\sigma$ (T10I4D100K $\sigma = 1000$ and T40I10D100K $\sigma = 30000$) and a fixed $T'$ (T10I4D100K $T' = \{85, 447, 859\}$ and T40I10D100K $T' = \{390, 464, 515, 611, 922\}$) (a) Runtime comparison. (b) Candidates comparison. (c) Memory consumption comparison.
numbers for all items based on the $TWU$ order, so as to better perform itemset comparison during the mining process, and to filter the high-utility itemsets that do not satisfy the constraints of $T'$ and $\sigma$ in advance, which can speed up the mining process. Experiments on different datasets and a comparison with the TargetUM algorithm have shown that THUIM has advantages over TargetUM in terms of running time and memory, and it has good scalability.

In the future, it is expected that the problem of targeted pattern mining will receive more interest from researchers and be applied in many fields such as for privacy protection, product recommendation, and intelligent search. Proposing a way to balance the relationship between an itemset and its match with a target pattern ($T'$) and its utility w.r.t the utility threshold ($\sigma$) is also worth considering. We also expect more efficient and excellent algorithms.

**REFERENCES**

1. J. M. Luna, P. Fournier-Viger, and S. Ventura, “Frequent itemset mining: A 25 years review,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 9, no. 6, p. e1329, 2019.
2. J. M. Luna, P. Padiño, M. Pechenizkiy, and S. Ventura, “Apriori versions based on MapReduce for mining frequent patterns on big data,” IEEE Transactions on Cybernetics, vol. 48, no. 10, pp. 2851–2865, 2018.
3. W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, “A survey of parallel sequential pattern mining,” ACM Transactions on Knowledge Discovery from Data, vol. 13, no. 3, pp. 1–34, 2019.
4. Y. Wu, C. Zhu, Y. Li, L. Guo, and X. Wu, “NetNCSNP: Nonoverlapping closed sequential pattern mining,” Knowledge-based systems, vol. 196, p. 105812, 2020.
5. R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in ACM International Conference on Very Large Data Bases. Citeseer, 1994, pp. 487–499.
6. J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” ACM SIGMIND Record, vol. 29, no. 2, pp. 1–12, 2000.
7. C. H. Lin, D. Y. Chiu, Y. H. Wu, and A. L. Chen, “Mining frequent itemsets from data streams with a time-sensitive sliding window,” in SIAM International Conference on Data Mining. SIAM, 2005, pp. 68–79.
8. J. Han, G. Dong, and Y. Yin, “Efficient mining of partial periodic patterns in time series database,” in International Conference on Data Engineering. IEEE, 1999, pp. 106–115.
9. C. H. Cai, A. W. C. Fu, C. H. Cheng, and W. W. Kwong, “Mining association rules with weighted items,” in International Database Engineering and Applications Symposium. IEEE, 1998, pp. 68–77.
10. G. C. Lan, T. P. Hong, and H. Y. Lee, “An efficient approach for finding weighted sequential patterns from sequence databases,” Applied Intelligence, vol. 41, no. 2, pp. 439–452, 2014.
11. H. Bui, B. Vo, T. A. Nguyen-Hoang, and U. Yun, “Mining frequent weighted closed itemsets using the WN-list structure and an early pruning strategy,” Applied Intelligence, vol. 51, no. 3, pp. 1439–1459, 2021.
12. U. Yun and J. J. Leggett, “WSpan: Weighted sequential pattern mining in large sequence databases,” in International IEEE Conference Intelligent Systems. IEEE, 2006, pp. 512–517.
13. 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.
14. V. S. Tseng, C. Wu, B. E. Shie, and P. S. Yu, “UP-Growth: an efficient algorithm for high utility itemset mining,” in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 253–262.
15. 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.
16. U. Yun, D. Kim, E. Yoon, and H. Fujita, “Damped window based high average utility pattern mining over data streams,” Knowledge-Based Systems, vol. 144, pp. 188–205, 2018.
17. B. E. Shie, V. S. Tseng, and P. S. Yu, “Online mining of temporal maximal utility itemsets from data streams,” in ACM Symposium on Applied Computing, 2010, pp. 1622–1626.
18. W. Gan, J. 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.
19. 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.
20. 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.
21. J. Chen, S. Wan, W. Gan, G. Chen, and H. Fujita, “TOPIC: Top-k high-utility itemset discovering,” arXiv preprint arXiv:2106.14811, 2021.
22. J. C. 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.
23. M. Liu and J. Qu, “Mining high utility itemsets without candidate generation,” in ACM International Conference on Information and Knowledge Management, 2012, pp. 55–64.
24. 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.
25. 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.
26. W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and P. S. Yu, “HUOPM: High-utility occupancy pattern mining,” IEEE Transactions on Cybernetics, vol. 50, no. 3, pp. 1195–1208, 2020.
27. 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.
28. U. Yun, H. Ryang, and K. H. Ryu, “High utility itemset mining with techniques for reducing overestimated utilites and pruning candidates,” Expert Systems with Applications, vol. 41, no. 8, pp. 3861–3878, 2014.
29. J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, “Targeted high-utility itemset querying,” in IEEE International Conference on Big Data. IEEE, 2021, pp. 5534–5543.
30. ——, “Targeted high-utility itemset querying,” IEEE Transactions on Artificial Intelligence. DOI: 10.1109/TAI.2022.3171530, pp. 1–13, 2022.
31. L. Shahbay, R. Yaari, and I. Dattner, “A guided FP-Growth algorithm for multi-targeted mining of big data,” arXiv preprint, arXiv:1803.06632, 2018.
32. R. Abeyesinghe and L. Dui, “Query-constraint-based association rule mining from diverse clinical datasets in the national sleep research resource,” in IEEE International Conference on Bioinformatics and Biomedicine, 2017, pp. 1238–1241.
12

[33] P. Fournier Viger, E. Mwamikazi, T. Gueniche, and U. Faghihi, “MEIT: Memory efficient itemset tree for targeted association rule mining,” in International Conference on Advanced Data Mining and Applications. Springer, 2013, pp. 95–106.

[34] C. Chand, A. Thakkar, and A. Ganatra, “Target oriented sequential pattern mining using recency and monetary constraints,” International Journal of Computer Applications, vol. 45, no. 10, 2012.

[35] H. E. Chueh et al., “Mining target-oriented sequential patterns with time-intervals,” International Journal of Computer Science & Information Technology, vol. 2, no. 4, pp. 113–123, 2010.

[36] C. Zhang, Q. Dai, Z. Du, W. Gan, J. Weng, and P. S. Yu, “TUSQ: Targeted high-utility sequence querying,” IEEE Transactions on Big Data, DOI: 10.1109/TBDATA.2022.3175428, pp. 1–16, 2022.

[37] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “Discovering frequent closed itemsets for association rules,” in International Conference on Database Theory. Springer, 1999, pp. 398–416.

[38] L. Szathmary, A. Napoli, and P. Valtchev, “Towards rare itemset mining,” in IEEE International Conference on Tools with Artificial Intelligence. IEEE, 2007, pp. 305–312.

[39] N. Aryabarzan, B. Minaei Bidgoli, and M. Teshnehlab, “negFIN: An efficient algorithm for fast mining frequent itemsets,” Expert Systems with Applications, vol. 105, pp. 129–143, 2018.

[40] U. Yun, “An efficient mining of weighted frequent patterns with length decreasing support constraints,” Knowledge-Based Systems, vol. 21, no. 8, pp. 741–752, 2008.

[41] T. Uno, M. Kiyomi, H. Arimura et al., “LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets,” vol. 126, 2004.

[42] 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.

[43] J. M. T. Wu, J. C. W. Lin, and A. Tamrakar, “High-utility itemset mining with effective pruning strategies,” ACM Transactions on Knowledge Discovery from Data, vol. 13, no. 6, pp. 1–22, 2019.

[44] W. Song, L. Liu, and C. Huang, “Generalized maximal utility for mining high average-utility itemsets,” Knowledge and Information Systems, vol. 63, no. 11, pp. 2947–2967, 2021.

[45] M. Kubat, A. Hafiez, V. V. Raghavan, J. R. Lekkala, and W. K. Chen, “Itemset trees for targeted association querying,” IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 6, pp. 1522–1534, 2003.

[46] G. Huang, W. Gan, and P. S. Yu, “TaSPM: Targeted sequential pattern mining,” arXiv preprint, arXiv:2202.13202, 2022.