Towards Efficient Discovery of Target High Utility Itemsets

1st Vincent Mwintieru Nofong
Faculty of Engineering
University of Mines and Technology, Ghana
vnofong@umat.edu.gh

2nd Priscilla Owiredu Okai
Deloitte Ghana
Accra, Ghana
owiredupriscillaokai@gmail.com

3rd Hamidu Abdel-Fatao
Faculty of Engineering
University of Mines and Technology, Ghana
habdel-fatao@umat.edu.gh

3rd Selasi Kwashie
CSIRO Data61
Adelaide, Australia
selasi.kwashie@data61.csiro.au

4th Michael Bewong
Charles Sturt University
Wagga Wagga, Australia
mbewong@csu.edu.au

5th John Wondoh
University of South Australia
Adelaide, Australia
john.wondoh@unisa.edu.au

***Abstract*—Finding High Utility Itemsets (HUIs) in databases is crucial for identifying items that are of high importance (like profit) for decision-making. However, current High Utility Itemset Mining (HUIM) algorithms often ignore the interest or target of users in favor of effectively identifying categories of HUIs using various measures and constraints. As a result, these techniques usually return a set of HUIs that are either too large or small to employ in decision making. Nevertheless, it is apparent that users are often interested in a select group of HUIs which may be among the set of HUIs reported using existing techniques. While some recent and early works have offered methods for discovering user-targeted HUIs, these methods are neither memory- nor time-efficient as they depend on post-processing or pattern matching. Additionally, throughout the discovery process, these techniques are required to scan the database twice. To address these issues, this paper proposes an efficient Target High Utility Itemset Miner (TarHUIM). In contrast to current methods, TarHUIM uses the users' target list and a single database scan to significantly reduce the search space and amount of time needed to find the user-targeted HUIs. Extensive experimental analysis show that TarHUIM is efficient and effective in discovering the set of targeted HUIs.**

***Index Terms*—frequent itemsets, high utility itemsets, target high utility itemsets.**

I. Introduction

Frequent Itemset Mining (FIM) was introduced in [1] with the aim of identifying patterns in customer transactions for market basket analysis. The goal in FIM (for any given database) is to identify items that co-occur frequently for a given user threshold. Frequent itemset mining is a relevant task as the discovered items that co-occur frequently in a database could reveal insights about events within the database for decision making. For instance, in market basket analysis, a frequent itemset such as $\{apple, pear\}$ in a customer transaction database simply imply that, customers often buy *apple* and *pear* together. Knowledge of this frequent itemset $\{apple, pear\}$ can be vital to support marketing decisions such as co-promoting *apple* and *pear*. Since its inception, research on frequent itemset mining has led to the development of a number of techniques and methodologies ([15], [22], [24], [29]) for mining different types of frequent itemsets for different decision-making.

Although frequent itemset mining techniques are able to reveal co-occurring patterns in databases, in some decision making, for instance, where interest is on items that return high profits when sold together, traditional FIM techniques will not be applicable. This is because frequent itemset mining techniques solely rely on the "*frequency*" threshold as the criterion of interest. As such, some of the frequent itemsets reported with traditional FIM techniques will not always be the high profit yielding itemsets needed in decision making.

The idea of High Utility Itemset Mining (HUIM) was thus developed to address the inapplicability of FIM techniques revealing the high profit yielding itemsets required in some decision making. Unlike frequent itemset mining (where frequency is important), high utility itemset mining (which is a generalization of frequent itemset mining) seeks to reveal co-occurring itemsets in databases that are of high utility. For example, in market basket analysis, while frequent itemset mining techniques will report the frequent co-occurring itemsets, HUIM techniques will return the high profit yielding co-occurring itemsets. Over the past years, several techniques and approaches have been proposed for discovering HUIs in works such as [4], [12], [14], [16]–[18], [23], [25], [26], [28].

Most of current HUIM approaches, however, often ignore the interest or target of users when discovering HUIs. As such, they often report either too many HUIs or too few HUIs to users for decision making. However, it is evident that users are often interested in a select group of HUIs which may be reported alongside those that are not of interest, or may not be reported at all (since the targeted HUIs may not meet the thresholds in existing techniques). Consequently, users have to either search through several reported HUIs to identify those of interest or keep adjusting the utility threshold until those of interest are reported.

Despite the fact that some early works on HUIM consider users' targets when mining targeted HUIs, they face the following challenges. To the best of our knowledge, existing

works on targeted high utility itemset mining either use *post-processing* ([20]) or *pattern matching* ([21]) during the discovery process to identify the targeted HUIs. Relying on either *post-processing* or *pattern matching* make these techniques inefficient in both memory usage and runtime as they mostly do not reduce the search space during the discovery process (see Example 11). As such, the search space in discovering targeted HUIs especially those using the *post-processing* techniques is the same as discovering all HUIs. It is also worth noting that, existing techniques which use either the *post-processing* or *pattern matching* techniques always have to scan the database twice: first to get the information about unique items and second to compute the *Transaction Weighted Utility* (TWU) of all items before subsequently sorting all transactions based on the TWUs. Scanning the database twice makes existing algorithms inefficient and time consuming in discovering targeted HUIs. Additionally, computing the TWUs of all items renders existing target HUIM techniques inefficient as some items may never co-occur with the user targeted items.

To address the aforementioned issues in existing targeted HUIM techniques, this paper proposes an efficient Target High Utility Itemset Miner (TarHUIM). Unlike existing techniques, TarHUIM scans the database once and drastically reduces the search space for the target HUIM based on the users' targeted items. Consequently, only transactions in which the targeted items co-occur are searched to identify the user targeted HUIs.

The main contributions of this paper are summarized as follows:

- An efficient Target High Utility Itemset Miner (TarHUIM) is proposed for mining the set of user-targeted high utility itemsets from databases.
- We present a strategy that enables a single database scan and effectively reduces the search space for target high utility itemset mining based on the users' target.
- Extensive experiments performed on benchmark datasets to evaluate the performance of TarHUIM show that TarHUIM is efficient and effective in discovering user targeted HUIs and can drastically reduce the search space for targeted HUIM compared to existing techniques.

The rest of this paper is organized as follows. Section II presents an overview of related work while Section III describes the problem of targeted high utility itemset mining. Section IV presents the proposed Target High Utility Itemset Miner (TarHUIM) and Section V discusses the experimental analysis as well as the results. Section VI finally concludes the paper and with some remarks and future works.

II. Related Work

Mining high utility itemsets in databases was proposed in [2] to tackle the inability of frequent itemset mining techniques to discover patterns that are of high utility (importance). Authors in [28], however, developed the mathematical technique to enable mine HUIs from databases. Since its proposition, HUIM has been widely researched on using various techniques and constraints for mining categories of HUIs for domain specific decision making. Such techniques and constraints can be found in works for mining: a.) concise representations of HUIs ([4], [10], [11], [27]), b.) HUIs with length constraints ([12]), c.) HUIs with negative utilities [3], [18], d.) correlated HUIs ([6], [8], [13], [14]), e.) top-k HUIs ([5]) and f.) traditional HUIs using various techniques ([7], [19], [30]).

To a large extent however, existing works on high utility itemset mining focus mainly on the efficient discovery of categories of HUIs for domain specific applications while ignoring the target (interest) of users. Consequently, users interested in a select group of HUIs will often have to search through a large set of reported HUIs (if the thresholds are set low) or a small set of reported HUIs (if the thresholds are set high). In most cases, users of traditional HUIMs techniques often have to repeatedly vary the thresholds and search among the reported HUIs to identify those of interest to them.

Quite recently, to address this issue and enable the mining of targeted HUIs, the works in [20], [21] proposed some initial techniques for mining HUIs based on user targets using either *post-processing* or *pattern matching*. Though these works are able to discover and report target HUIs, they are inefficient in both memory and runtime during the discovery of these targeted HUIs as follows.

Firstly, for targeted HUIM techniques that employ *post-processing* [20], all high utility itemsets (based on the set thresholds) need to be discovered, and a search using the users' target is conducted among the discovered HUIs to report the target HUIs. This approach of mining all HUIs and subsequently searching based on the users' target makes target-oriented HUIM techniques using *post-processing* both memory and time inefficient. Secondly, for targeted HUIM techniques that use *pattern matching* techniques [21], all unique items (in the database) are assigned serial numbers during the discovery process. This is counter-intuitive, making such techniques inefficient in both memory and time, as not all unique items are relevant in target HUIM - only items that co-occur with the users' target should be serialized, not all unique items. Thirdly, to the best of our knowledge, existing target HUIM techniques (that use either *pattern matching* or *post-processing*), always scan the database twice. These techniques have to firstly scan the database to obtain the information about the unique items in the database as well as compute their respective *Transaction Weighted Utilities* (TWUs). Again, computing the TWUs of all items in the database is counter-intuitive and thus make these techniques both memory and time inefficient since some items may never co-occur with the users' target (see Examples 10 and 11). In the second scan of the database (aimed at reducing the search space using the users' target), all transactions are compared with the target of the user before being sorted using the computed TWUs. Though this reduces the search space for the target HUIM, it is time consuming since some transactions which do not contain the users' target are still compared with the users' target before they are pruned from the search space (see Examples 10 and 11).

To address these challenges faced by existing target HUIM techniques mentioned previously, this paper proposes and implements a novel algorithm named TarHUIM (Target High

Utility Itemset Miner). Unlike existing target HUIM techniques ([20], [21]), TarHUIM scans the database only once and subsequently reduces the search space using the users' target (see Example 13). Additionally, TarHUIM mines and report the set of target HUIs without creating the utility list, without computing the TWUs or the remaining utilities of all items (as is done in existing techniques) and as a result, making TarHUIM efficient in terms of memory and runtime.

III. Preliminaries and Problem Statement

This section introduces the concepts and preliminary steps in mining frequent and high utility itemsets. Subsequently, the problem of targeted high utility itemset mining is presented.

A. Frequent Itemset Mining

The notations associated with frequent itemset mining are as follows.

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items. A set $X_1 = \{i_a, \ldots, i_j\} \subseteq I$ (such that $a \leq j$ and $a, j \in [1, m]$), is called a pattern (or an itemset). A transaction database is a set of transactions $D = \{T_1, T_2, T_3, \ldots, T_k\}$ such that for each transaction T_a, $T_a \subseteq I$ and T_a has a unique identifier a called its transaction ID (TID).

Example 1: Let Table I be the transaction database of a store, where the set of items for this database is $I = \{a, b, c, d, e, f\}$. Transaction T_3 in Table I which has a transaction ID of 3 and three items (that is, a, b and c) is a length-3 itemset. □

TABLE I: Sample Customer Transactions

TID	Transaction
T_1	$\{d, e, f\}$
T_2	$\{b, d, f\}$
T_3	$\{a, b, c\}$
T_4	$\{b, d, e, f\}$
T_5	$\{a, c, d\}$

The *coverset* ($cov(S)$) of an itemset, S, in a database, D is defined as $cov(S) = \{TID | \forall T_{TID} \in D \wedge S \subseteq T_{TID}\}$

Example 2: In Table I, given, $S = \{b, f\}$, then $cov(S) = \{2, 4\}$ since $\{b, f\}$ occurs in transactions T_2 and T_4. □

The *support count* (also referred to as *count*) of a pattern S in D (denoted as $|cov(S)|$) is defined as the number of times S appears in the database. The *support* (also known as *absolute support*) of S (denoted as $sup(S)$) is defined as follows:

$$sup(S) = \frac{|cov(S)|}{|D|} \tag{1}$$

where $|D|$ is the size of the database.

Example 3: In Table I, given $S = \{b, f\}$, then *support count* of S becomes 2 since $\{b, f\}$ occurs in two transactions. The *support* of S, that is, $sup(S)$ can thus be evaluated as $sup(S) = \frac{2}{5} = 0.4$ since $|cov(S)| = |\{2, 4\}| = 2$ and the database size, that is, $\mid D \mid = 5$. □

A frequent itemset is defined by [1] as follows:

Definition 1: **(*Frequent itemset* [1])** Given a user defined minimum support threshold of frequency, $minsup$ (such that, $0 < minsup \leq 1$) and a database D, an itemset S, is a frequent itemset if the support of S is greater than or equal to the minimum support threshold of frequency, that is, $sup(S) \geq minsup$.

The problem of Frequent Itemset Mining is defined as follows:

Definition 2: **(*Frequent itemset mining* [1]).** Given a set of items I, a transaction database D, and a minimum support threshold (*minsup*) set by the user, the problem of Frequent Itemset Mining (FIM) is to discover all frequent itemsets in the transaction database D.

Example 4: Considering the customer transactions in Table I and a minimum *support* of 0.6, the set of frequent itemsets and their respective supports from Table I will be: $\{b\} : 0.6, \{d\} : 0.8, \{f\} : 0.6$ and $\{d, f\} : 0.6$. □

B. High Utility Itemset Mining (HUIM)

The notations for HUIM are the same as those of frequent pattern mining with the following additions.

Given $I = \{i_1, i_2, ..., i_m\}$ as a set of literals, called items. A *quantitative transaction database*, D, is a set of transactions, denoted as $D = \{T_1, T_2, \ldots, T_k\}$, where each transaction T_c is a set of items such that $T_c \subseteq I$, and has a unique identifier c called its Transaction ID (TID). Each item $i \in I$ is associated with a positive number $eu(i)$, called its *external utility*. The *external utility* of an item is a positive number representing the unit profit of the item. Moreover, every item i appearing in a transaction T_c has a positive number $n(i, T_c)$, called its *internal utility*, representing the purchase quantity of item i in transaction T_c.

Example 5: Let Table II be a quantitative transaction database of a store, the set of items for this database is $I = \{a, b, c, d, e, f, g\}$. Table III depicts the profit of each item, that is, the external utility of each item in Table II. In Table II, transaction T_1 which is *'d e f:7:4 2 1'* correspondingly imply that, the customer bought 4 units (pieces) of d, 2 units (pieces) of e and 1 unit (piece) of f with 7 as the total number of items bought (that is, $4 + 2 + 1$). Table III which contains the external utility imply that the profit gained for a unit of a sold is 3 while the profit gained for a unit of f sold is 9. □

TABLE II: Sample Quantitative Customer Transactions

TID	Transaction
T_1	*d e f:7:4 2 1*
T_2	*b d:4:1 3*
T_3	*a b c:3:1 1 1*
T_4	*b d e f:7:1 3 1 2*
T_5	*a c d:6:1 4 1*
T_6	*b d e f:5:1 1 1 2*
T_7	*a d g:7:1 2 4*
T_8	*a b d g:6:2 1 2 1*
T_9	*a c d g:7:3 1 1 2*
T_{10}	*b d g:8:2 2 4*

A utility function is used to compute the utility of each item - that is, a measure of the relevance of an item in a database to the user. The utility in this case is defined as the profit generated by items that are bought together. Formally, the utility of an item in a transaction is defined as:

Definition 3: **(Utility of an item)** The utility of an item i in a transaction T_c (denoted as $u(i, T_c)$) is the product of

TABLE III: External Utilities* of Items in Table II

Item	External Utility
a	3
b	2
c	5
d	3
e	7
f	9
g	10

*The external utility database, in reality, stores items and their utilities in the form *item:utility* per line, e.g., *b:2*

the external utility and internal utility of the item in that transaction, formally expressed as:

$$u(i, T_c) = eu(i) \times n(i, T_c) \tag{2}$$

Example 6: Given the item $\{d\}$ which occurs in transaction T_1, based on Tables II and III, its utility in T_1 will be evaluated using Equation 2 as $u(d, T_1) = 3 \times 4 = 12$. □

The utility of an itemset in a transaction is defined as:

Definition 4: **(Utility of an itemset)** The utility of an itemset S in a transaction T_c (denoted as $u(S, T_c)$) is the sum of the products of the external and internal utilities of the items in S for transaction T_c, formally expressed as:

$$u(S, T_c) = \sum_{i \in S} u(i, T_c) \tag{3}$$

Example 7: Given the itemset $S = \{a, b\}$ which was bought in transaction T_3, based on Tables II and III, its utility will be evaluated using Equation 3 as:

$$u(a, b, T_3) = u(a, T_3) + u(b, T_3) = (3 \times 1) + (2 \times 1) = 5. \square$$

The total utility of an itemset in a database is defined as:

Definition 5: **(Total utility of an itemset)** The total utility of an itemset S in a database D (denoted as $u(S, D)$) is the sum of the products of the external and internal utilities of items in the itemset S for all transactions in D where S appears, formally expressed as:

$$u(S, D) = \sum_{T_c \in p(S)} u(S, T_c) \tag{4}$$

where $p(S) \subseteq D$ is the set of all transactions in the database containing itemset S.

For brevity, we use $u(S)$ as the total utility for the rest of the paper.

Example 8: Given the itemset $S = \{a, b\}$ in Table II, its total utility will be evaluated using Equation 4 as:

$$u(\{a, b\}) = \sum_{T_c \in p(\{a,b\})} u(\{a, b\}, T_c)$$

$$= u(\{a, b\}, T_3) + u(\{a, b\}, T_8))$$

$$= 5 + 8$$

$$hence, u(a, b) = 13$$

□

Formally, a high utility itemset is defined as:

Definition 6: **(High utility itemset)** [8] An itemset S is a high-utility itemset if its total utility $u(S)$ is no less than a user-specified minimum utility threshold (denoted as *minutil*) set by the user (i.e., $u(S) \geq minutil$). Otherwise, S is a low-utility itemset.

Example 9: For the quantitative customer transactions in Table II, their external utility values shown in Table III, given $minutil = 10$, a total of 36 high-utility itemsets will be reported as listed in Table IV. □

TABLE IV: HUIs from Table II given *minutil =10*

S	u(S)	S	u(S)	S	u(S)	S	u(S)
$\{a\}$	24	$\{a, d\}$	39	$\{d, f\}$	69	$\{b, d, e\}$	30
$\{b\}$	14	$\{a, g\}$	88	$\{d, g\}$	131	$\{b, d, f\}$	52
$\{c\}$	30	$\{b, d\}$	45	$\{e, f\}$	73	$\{b, d, g\}$	68
$\{d\}$	57	$\{b, e\}$	18	$\{a, b, c\}$	10	$\{b, e, f\}$	54
$\{e\}$	28	$\{b, f\}$	40	$\{a, b, d\}$	14	$\{c, d, g\}$	28
$\{f\}$	45	$\{b, g\}$	56	$\{a, b, g\}$	18	$\{d, e, f\}$	97
$\{g\}$	110	$\{c, d\}$	31	$\{a, c, d\}$	43	$\{a, b, d, g\}$	24
$\{a, b\}$	13	$\{c, g\}$	25	$\{a, c, g\}$	34	$\{a, c, d, g\}$	37
$\{a, c\}$	45	$\{d, e\}$	52	$\{a, d, g\}$	103	$\{b, d, e, f\}$	66

In Example 9, a total of 36 HUIs are reported for the given thresholds from a database with only ten (10) transactions (comprising of 7 unique items). It is worth noting that, if the unique items should increase alongside with the number of transactions, the reported HUIs can be exponential. In such a situation, it will be a burden to decision makers if all HUIs are reported, as not all reported HUIs might be needed in decision making to the user who might be interested in only a select group of HUIs. This thus calls for the the need to selectively mine user targeted high utility itemsets.

Though some recent works have proposed techniques for mining user targeted HUIs, as mentioned previously, they are inefficient in memory consumption and runtime as explained in the Examples 10 and 11.

Example 10: For works that employ the *post-processing* technique [20], given the user target HUIs must contain $\{e, f\}$, with the database in Table II, such techniques will mine all the 36 HUIs reported in Example 9 and subsequently search among the 36 reported HUIs to return the target HUIs as $\{e, f\}, \{b, e, f\}, \{d, e, f\}$ and $\{b, d, e, f\}$. The *post-processing* technique searches the entire database to mine all HUIs and subsequently search for and report the target HUIs. However, from Table II, it is only in transactions T_1, T_4 and T_6 that target items $\{e\}$ and $\{f\}$ co-occur. As such, it becomes pointless and a waste of time to search for the targeted HUIs among transactions that do not contain the targeted items. □

Example 11: For works that employ the *pattern matching* techniques [21], given the target HUIs must contain $\{e, f\}$, with the database in Table II, such techniques will scan the database to compute the TWUs and remaining utilities of all unique items though some items may not co-occur with the target items. For instance, with $\{e, f\}$ as the target, though items $\{a\}, \{c\}$ and $\{g\}$ never co-occur with the target items in the entire database, their TWUs and remaining utilities will still be computed. Additionally, all transactions are scanned

twice, including transactions that do not contain the target items ($\{e, f\}$). That is, transactions $T_2, T_3, T_5, T_7, T_8, T_9$ and T_{10} will still be scanned twice even though the target items do not co-occur in these transactions. Therefore, it is a waste of both time and memory; computing the TWUs and remaining utilities of items that never co-occur with the users' target, as well as scanning and comparing transactions that do not contain the target items. □

C. Problem Statement

HUIM techniques always discover and report all itemsets that are of high utility (based on thresholds) in the database. However, users, in some cases, are only interested in HUIs that contain some targeted items. As illustrated in Examples 10 and 11, techniques that use either *post-processing* or *pattern matching* to discover targeted HUIs are not efficient as they, in most cases, still search for the targeted HUIs in transactions that do not contain the target of users, hence the need for an efficient approach to discover targeted HUIs.

Problem Statement: The goal of targeted high utility itemsets mining is to efficiently identify all HUIs based on users' targeted list of items (as well as utility thresholds) among the set of transactions that contain the users' target list of items while ignoring those that do not contain the users' target list. For example, given that the target HUIs must contain $\{e, f\}$, with the database in Table II, only in transactions T_1, T_4 and T_6 (in which $\{e\}$ and $\{f\}$ co-occur) should be used in mining the set of targeted HUIs.

Section IV presents details of our proposed Target High Utility Itemset Miner (TarHUIM) - which addresses the above problem statement.

IV. The TarHUIM Algorithm

Our proposed Target High Utility Itemset Miner (TarHUIM) for mining user targeted high utility itemsets is shown in Algorithm 1. TarHUIM which takes as input the quantitative transaction database (D), the external utility database (eD), the user target list (*TargetList* - for our running example, $TargetList = \{e, f\}$), the minimum utility threshold ($minUtil$) and a user threshold (α) discovers and reports the Target High Utility Itemsets (THUIs) as follows.

For any given inputs $(D, eD, minUtil, TargetList, \alpha)$, Line 1 (of Algorithm 1) creates the required data structures needed. That is,

- *iCov*: a dictionary for storing the unique itemsets and an ordered list of their respective coversets.
- *eUtil*: a dictionary for storing the unique length-1 items and their respective external utilities.
- *iUtil*: a dictionary for storing the unique length-1 items and an ordered list of their respective internal utilities.
- *iList*: for storing the candidate high utility itemsets.
- *TarIDs*: for storing the TIDs of transactions where the users' targeted items co-occur.
- THUIs: for storing the discovered targeted HUIs.

Subsequently, Line 2 creates the variables *cLine* and *cLen* to keep track of the transaction IDs (in D) and the number of items in *iList* respectively. Function 2 (FindLen1Items()) in Line 3 obtains the unique length-1 items and their respective coversets as well as quantities (which are stored in $iCov$ and $iUtil$ respectively) while Function 3 (FindTargetTransIDs()) in Line 4 is employed to identify the transaction IDs of transactions in which the users' target list of items co-occur (see Example 13 for illustration). Given the target list of items do not co-occur in the database (that is, $|TarIDs| < 1$), Line 6 obtains the external utilities of the target items from the external utility database, and Line 7 subsequently mines length-1 THUIs (based on the users' target) and TarHUIM terminates. If the users' target list of items co-occur in the database (that is, $len(TarIDs) \geq 1$)), Function 4 (PruneSearchSpace()) in Line 9 is used to identify the set of transactions which will be used in the target HUIM and subsequently add the unique items which co-occur with the users' target list to $iList$. Line 10 obtains the external utilities of the items that co-occur with the items in the target list while Line 11 mines the length-1 target HUIs. Function 7 (MineRemTHUIs()) in Line 13 subsequently mines the remaining target HUIs that have lengths more than one (1). When all targeted HUIs are mined, Line 14 returns THUI, the set of all discovered targeted high utility itemsets based on the users' target.

Algorithm 1: TarHUIM(D, eD, *minUtil, TargetList*, α)

Input: D, eD, *minUtil, TargetList*, α
Output: Target high utility itemsets, THUI

```
1  Create iCov, eUtil, iUtil, iList, TarIDs, THUI
2  Create cLine = 0, cLen = 0
3  FindLen1Items(D, iCov, iUtil, cLine)
4  FindTargetTransIDs(iCov, TargetList, TarIDs)
5  if len(TarIDs) < 1 then
6      GetExtUtil(eD, eUtil, TargetList)
7      FindLen1THUIs(TargetList, eUtil, iUtil)
8  else
9      PruneSearchSpace(iList, iCov, TarIDs)
10     GetExtUtil(eD, eUtil, iList)
11     FindLen1THUIs(TargetList, eUtil, iUtil)
12     cLen = len(iList)
13     MineRemTHUIs(cLen, iList, eUtil, iUtil)
14 return THUI
```

The following subsections discuss in details the how the sub-functions in TarHUIM (Algorithm 1) work in mining the user targeted high utility itemsets.

A. Finding Length-1 Items

Function 2 (FindLen1Items()) is used in TarHUIM (see Line 3 of Algorithm 1) to find the set of length-1 items with their respective *coversets* and *internal utilities* from the quantitative transaction database (D). Taking as input the quantitative database (D), the dictionaries $iCov$ and $iUtil$, and the variable $cLine$, Function 2, returns the set of length-1 items with their respective *coversets* and *internal utilities* from D as follows.

Function 2: FindLen1Items()

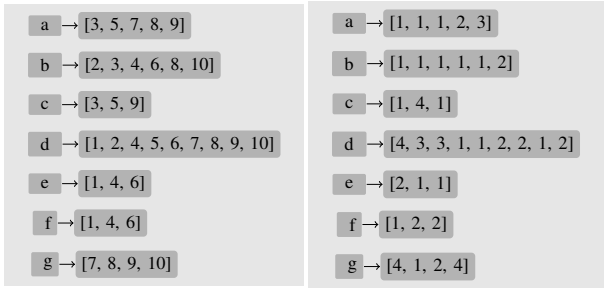
Input: D, $iCov, iUtil, cLine$
Output: $iCov, iUtil$

1 **for** ***each*** *transaction,* T_k *in* D **do**
2 $cLine+ = 1$
3 Let Lsp = T_k.split(":")
4 Let $trans$ = Lsp[0].split(" ")
5 Let $transUtil$ = Lsp[2].split(" ")
6 **for** ***each*** *index,* i *in* $trans$ **do**
7 Let $item = trans[i]$ and $inutil = transUtil[i]$
8 **if** *item is not in* $iCov$ **then**
9 $iCov[item] = [cLine]$
10 $iUtil[item] = [inutil]$
11 **else**
12 $iCov[item].update([cLine])$
13 $iUtil[item].update([inutil])$
14 **return** $iCov, iUtil$

For each transaction in the dataset D, Lines 3, 4 and 5 split the transactions into the items bought ($trans$) and their respective internal utilities ($transUtil$). Take for example the first transaction, T_1 = *d e f:7:4 2 1* in Table II, Lines 3, 4 and 5 will result in $trans$ = ['d', 'e', 'f'] and $transUtil$ = ['4', '2', '1'].

In Lines 6 through 13, the dictionaries $iCov$, and $iUtil$ are updated as follows. Line 7 gets the $item$ at i^{th} index in $trans$ with its corresponding internal utility ($inutil$) at the i^{th} index in $transUtil$. If the $item$ is not in $iCov$, Line 9 creates a new entry in $iCov$ with the *key* as *"item"* and *value* as an ordered list containing *"cLine"* (the variable $cLine$ keeps track of the transaction IDs of items) while Line 10 creates a new entry in $iUtil$ with the *key* as *"item"* and the *value* as an ordered list containing *"inutil"*. If on the other hand, the $item$ is found in $iCov$, Line 12 updates the ordered coverset of $item$ in $iCov$ with *"cLine"* while Line 13 updates the ordered internal utility of $item$ in $iUtil$ with *"inutil"*.

For illustration purposes, Figure 1 shows the content of $iCov$ (Figure 1a) and $iUtil$ (Figure 1b) respectively after Function 2 scans Table II.



(a) $iCov$ (b) $iUtil$

Fig. 1: $iCov$ and $iUtil$ after Function 2 Scans Table II

B. Finding Target Transactions

TarHUIM uses Function 3 (FindTargetTransIDs()) to identify all transaction IDs in which the users' target list of items co-occur in the database as follows.

Function 3: FindTargetTransIDs()

Input: $iCov$, *TarIDs, TargetList*
Output: *TarIDs*

1 **if** *TargetList contains only one item, i* **then**
2 Get $cov(i)$ from $iCov$, update *TarIDs* $\leftarrow cov(i)$
3 **else**
4 Get coversets of all items in *TargetList* from $iCov$
5 Get intersection of coversets of items in *TargetList*
6 Update *TarIDs* $\leftarrow$ {intersection of coversets}
7 **return** *TarIDs*

If the list of user targeted items, *TargetList* has only one item, its coverset is obtained from $iCov$ and subsequently stored in $TarIDs$ in Line 2. If *TargetList* has more than one item, the set of target transaction IDs are obtained from Lines 4 to 6 by getting the coversets of each item from $iCov$ and subsequently finding the intersection of all coversets. The intersection of all coversets (which is the set of all transactions in which all the target items co-occur) is then added to $TarIDs$ in Line 6 while Line 7 returns the set of targeted transaction IDs.

Example 12: In our running example, with the target list of items as $\{e, f\}$, that is, $TargetList = \{e, f\}$, from Figure 1a, Line 5 will return the intersection as $[1, 4, 6]$ since $cov(e) = [1, 4, 6]$ and $cov(f) = [1, 4, 6]$. Subsequently, $[1, 4, 6]$ will be added to $TarIDs$ in Line 6 while in Line 7, $TarIDs = [1, 4, 6]$ will be returned as the set of targeted transactions IDs needed in the targeted HUI mining.

C. Pruning Search Space

TarHUIM uses Function 4 (PruneSearchSpace()) to identify the set of all length-1 items which co-occur with the targeted transactions IDs obtained in Function 3. Function 4, which does not scan the database again as existing works, only searches in $iCov$ to identify the length-1 items that occur in the obtained targeted transactions ($TarIDs$) as follows.

For every item, i in $iCov$, for a given threshold (α), if i occurs in the targeted transactions (that is, $|cov(i) \cap TarIDs| \geq \alpha$, where α is a user threshold), i is added to $iList$ in Line 4 while: a.) Line 5 updates the entry of i in $iCov$ with new values ($cov(i) \cap TarIDs$, that is, an ordered list of the transaction IDs of i in the targeted transactions), and b.) Line 6 updates the entry of i in $iUtil$ with the ordered corresponding internal utilities of i in the targeted transactions. If the item i does not occur in the targeted transactions, its entry is removed from both $iCov$ and $iUtil$ in Line 8. Line 9 then sorts $iList$ in ascending order while Line 10 returns the sorted $iList$ as well as the updated $iCov$ and $iUtil$. We illustrate this process in Example 13.

Function 4: PruneSearchSpace()

Input: $iList, iCov$, *TarIDs*
Output: $iList$, $iCov$

1 **for** ***each*** *item,* i *in* $iCov$ **do**
2 Get $cov(i)$ from $iCov$
3 **if** $cov(i) \cap$ *TarIDs* $\geq \alpha$ **then**
4 Add i to $iList$
5 Update $iCov[i] = [cov(i) \cap TarIDs]$
6 Update $iUtil[i] =$ [corresponding utilities]
7 **else**
8 Remove i entry from both $iCov$ and $iUtil$
9 Sort $iList$ in ascending order
10 **return** $iList, iCov, iUtil$

Example 13: From our running example, where the target items, *TargetList*= $\{e, f\}$, $iCov$ and $iUtil$ in Figures 1a and 1b respectively, the obtained $TarIDs = [1, 4, 6]$ and $\alpha = 2$, item $\{a\}$ and its entries will be removed from $iCov$ and $iUtil$ since $|cov(a) \cap TarIDs| < 2$ (that is, $[3, 5, 7, 8, 9] \cap [1, 4, 6] = \emptyset$ in Line 3). After searching through the rest of the items in $iCov$, Line 10 will return $iList = \{b, d, e, f\}$ with the updated $iCov$ and $iUtil$ as shown in Figures 2a and 2b respectively.

(a) $iCov$ (b) $iUtil$

Fig. 2: Updated $iCov$ and $iUtil$ after Running Function 4

The updated $iCov$ and $iUtil$ in Figure 2 are what will the be used in the targeted high utility itemset mining. Compared to existing techniques [20], [21], TarHUIM drastically reduces the search space based on the targeted items. For instance, items $\{a\}, \{c\}$ and $\{g\}$ which do not co-occur in the database with the user target list will not be employed in the discovery process. Additionally, the targeted HUIs will not be searched for in transactions $T_2, T_3, T_5, T_7, T_8, T_9$ and T_{10} since the target list of items do not occur these transactions. □

D. Getting External Utilities

The external utilities of items are obtained from the external utility database in TarHUIM using Function 5 as follows.

For each line in the external utility database, it is split into the *item* and its *external utility* in Lines 2 and 3. If the *item* is found in $iList$[1] (that is, the item occurs in the targeted transactions containing the targeted items), $eUtil$ is updated in Line 5 with the item and its external utility. After scanning the content in eD, Line 6 returns $eUtil$ which contains the external utilities of only items co-occurring with the targeted

[1]Note that if $len(TarIDs) < 1$, *TargetList* will be used instead of $iList$

Function 5: GetExtUtil(*eD, eUtil, iList*)

Input: eD, $eUtil, iList$
Output: $eUtil$

1 **for** ***each*** *line,* L_n *in* eD **do**
2 Lsplit = L_n.split(":")
3 Let $item$ = Lsplit[0] and $utility$ = Lsplit[1]
4 **if** *item is in iList* **then**
5 $eUtil[item] = utility$
6 **return** $eUtil$

items. For our running example where $iList = \{b, d, e, f\}$ and the external utility database in Table III, Line 6 will return $eUtil = \{b : 2, d : 3, e : 7, f : 9\}$

E. Mining Length-1 Target HUIs

TarHUIM uses Function 6 (FindL1THUIs()) to mine the length-1 targeted HUIs. Though the target list might occur with some other length-1 items, this function only evaluates the utilities of only the targeted length-1 items since the user is only interested in the utility of the target items. Function 6 mines the length-1 target high utility itemsets as follows.

Function 6: FindLen1THUIs()

Input: $TargetList, eUtil, iUtil$
Output: THUI

1 **for** ***each*** *item,* i *in* $TargetList$ **do**
2 Let $i\_exUtil = eUtil[i]$
3 Let $i\_utiList = iUtil[i]$
4 Compute total utility, $u(i)$ of i
5 **if** $u(i) \geq minUtil$ **then**
6 THUI$[i] = u(i)$
7 **return** THUI

For each item, i in the *TargetList*, its external and internal utilities are obtained from $eUtil$ and $iUtil$ in Lines 2 and 3 to compute the total utility in Line 4. Given the total utility computed is not less than the minimum utility threshold, the item is added to the set of targeted HUIs in Line 6.

For our running example, where $\{e\}$ is in *TargetList*, Lines 2 and 3 will return $\{e\}\_exUtil = [7]$ and $\{e\}\_utiList = [2, 1, 1]$. The total utility of $\{e\}$ will then be computed using Equation 4 as $u(e) = (7 \times 2) + (7 \times 1) + (7 \times 1) = 28$. Given the minimum utility threshold is 10, $\{e\}$ will be added to THUI. In our running example, after scanning all items in $TargetList$, Line 7 will return THUI $= \{\{e\} : 28, \{f\} : 45\}$

F. Mining the Remaining Target HUIs

The remaining targeted HUIs are then mined using Function 7 from $iList, TarIDs, eUtil$ and $iUtil$ as follows.

Firstly, if the number of items in the target list of items that co-occur with the user targeted list ($iList$) is less than or equal to one (1) (that is, is $cLen \leq 1$, the targeted HUIM terminates and only the set of length-1 THUIs is returned in Line 19. If the number of items in $iList$ is greater than one

Function 7: MineRemTHUIs()

Input: $cLen, iList, eUtil, iUtil$, *TarIDs*
Output: THUI

```
1  while cLen > 1 do
2      Create temList as a List
3      for each item, S1 in iList do
4          for each item, S2 in iList do
5              if S1 & S2 meet cand gen property then
6                  Get cov(S1) & cov(S2) from iCov
7                  Let cov(S) = cov(S1) ∩ cov(S2)
8                  if |cov(S) ∩ TarIDs| ≥ α then
9                      Let S = S1 ∪ S2
10                     iCov[S] = [cov(S)]
11                     Update temList ← {S}
12                     if |S ∩ TargetList| > 0 then
13                         Find total utility u(S) of S
14                         if u(S) ≥ minUtil then
15                             THUI[S] = u(S)
16     iList.clear()
17     Copy items in temList into iList
18     Set cLen = len(iList)
19 return THUI
```

(1) the remaining THUIs are mined from Lines 1 to 18 as follows.

While $iList$ has more than one item (that is, $cLen > 1$), a temporary list, $temList$ is created in Line 2 and the targeted HUIs are repeatedly mined from $iList$ as follows. For any two pair of items in $iList$, S_1 and S_2, $cov(S_1)$, $cov(S_2)$, and $cov(S)$ (where $S = S_1 \cup S_2$) are obtained in Lines 6 and 7 respectively, provided they satisfy the Apriori candidate generation property. If the candidate target high utility itemset co-occurs with the users' targeted items (that is, $|cov(S) \cap TarIDs| \geq \alpha$ in Line 8), then the candidate itemset S is generated in Line 9 and $iCov$ updated in Line 10 with S and its coverset. The candidate HUI S is also added to $temList$ in Line 11. If the candidate THUI, S contains any of the target items (that is, $|S \cap TargetList| > 0$), its total utility, $u(S)$ is computed in Line 13 using $eUtil$ and $iUtil$. Given the total utility of S is greater than the minimum utility threshold, it is added to THUI in Line 15. After iterating through all items in $iList$, the content in $iList$ is cleared in Line 16. In Line 17, the content of $temList$[2] is copied into $iList$. The variable $cLen$ is again assigned to the length of $iList$ (that is, number of items in $iList$). Given that $cLen$ is greater than 1, the process from Lines 2 to 18 is repeated until $cLen \leq 1$ when the targeted HUIM process terminates. Line 14 of Algorithm 1 (TarHUIM) thus returns THUI as the set of all targeted HUIs. We illustrate this process in Example 14.

Example 14: Given $iList = \{b, d, e, f\}$ with the updated $iCov$ and $iUtil$ in Figures 2a and 2b as well as $eUtil = \{b : 2, d : 3, e : 7, f : 9\}$, the remaining target HUIs are mined using Function 7 as follows. Since $cLen = 4$, in the first instance, $\{b\}$ will be picked as S_1 while $\{d\}$ as S_2 (Note that the selections of S_1 and S_2 are based on the position indexes; as such, for any given index, $S_1 \neq S_2$). Since $\{b\}$ and $\{d\}$ meet the Apriori candidate generation property, $cov(b)$ and $cov(d)$ will be obtained from $iCov$ as $cov(b) = [4, 6]$ and $cov(d) = [1, 4, 6]$ respectively (hence, $cov(b, d) = [4, 6]$). Given $|cov(b, d) \cap TarIDs| = |[4, 6]|$ is not less than α (which is 2), $S = \{b, d\}$ will be created in Line 9 while $iCov$ updated with $\{b, d\} : [4, 6]$ in Line 10 and $\{b, d\}$ added to $temList$ in Line 11. In Line 12, given $|S \cap TargetList| = \emptyset$, the utility of $\{b, d\}$ is not computed since $\{b, d\}$ does not contain any of the users' targeted items.

With S_1 still as $\{b\}$ the next item in $iList$ is selected as S_2, that is, $\{e\}$. The process described above is repeated for $\{b\}$ and $\{e\}$. For $\{b\}$ and $\{e\}$, $iCov$ will be updated with $\{b, e\} : [4, 6]$ in Line 10 while $\{b, e\}$ added to $temList$ in Line 11. Since $\{b, e\} \cap TargetList = \{e\}$, the utility of $\{b, e\}$ will be computed. Here, the internal utilities of $\{b\}$ and $\{e\}$ will be obtained from $iUtil$ using $cov(b, e)$ (that is, with $cov(b, e) = [4, 6]$) as $\{b\} : [1, 1]$ and $\{e\} : [1, 1]$. Equation 4 will then be used to compute the total utility based on their external utilities. Given $u(b, e) = 18$ which is greater than $minUtil$ (which is 10), $\{b, e\} : 18$ is added to THUI in Line 15.

After the first nested for-loops, $iList$ is cleared in Line 16 and the content in $temList$ is copied into iList in Line 17. The process repeats until there is no itemset in $iList$ or only one itemset. For our running example, given $minUtil = 10$ and $TargetList = \{e, f\}$, the set of discovered target high utility itemsets will be as shown in Table V.

TABLE V: THUIs from Table II given *minutil =10* and $TargetList = \{e, f\}$

S	u(S)	S	u(S)	S	u(S)	S	u(S)
$\{e\}$	28	$\{b, f\}$	40	$\{e, f\}$	73	$\{b, e, f\}$	54
$\{f\}$	45	$\{d, e\}$	52	$\{b, d, e\}$	30	$\{d, e, f\}$	97
$\{b, e\}$	18	$\{d, f\}$	69	$\{b, d, f\}$	52	$\{b, d, e, f\}$	66

Section V discusses the experimental setup and analysis performed on TarHUIM to illustrate its effectiveness in discovering targeted high utility itemsets.

V. Experimental Analysis

A. Experimental Setup

TABLE VI: Features of Experimental Datasets

Dataset	Total Transactions	Unique Items	Nature
Foodmart	4,141	1,559	Sparse
Kosarak10k	10,000	10,094	Sparse
Mushroom	8,416	119	Partly Dense
Chess	3,196	75	Very Dense

Experiments were conducted on four datasets[3] with the characteristics as shown in Table VI.

[2] $temList$ will always contain items to be used in generating the next candidate THUIs

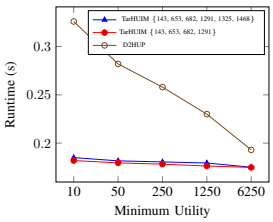
[3] These datasets were obtained from [9] with only the internal utilities as such we synthetically generated the external utilities.

The algorithms used in the experimental analysis are TarHUIM and D2SUP [19] since the authors of [20] and [21] did not readily make available their implementations for comparison. However, it is worth noting that, though [20] and [21] will report same targeted HUIs as TarHUIM, in the best case scenario, the runtimes as well as memory used by both [20] and [21] will be same as D2SUP. TarHUIM is implemented in Python while D2SUP (one of the fastest HUIM techniques available - obtained from [9]) is implemented in Java.

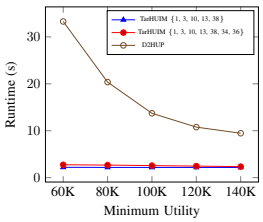
For the sparse datasets (Foodmart and Kosarak10k), we set $\alpha = 1$, which means, an item must occur at least once in the transactions in which the users' target list of items co-occur. Also, in Line 12 of Function 7, the threshold of intersection is set to 1 (meaning, the candidate item must have at least one of the users' targeted items). For the dense datasets (Mushroom and Chess), we set $\alpha = len(TarID)$, meaning the item must occur in all the transactions that the users' targeted items co-occur. Additionally, the threshold of intersection is set to $len(TargetList)$ in Line 12 of Function 7. The thresholds are set above for the dense datasets to avoid reporting an exponential number of THUIs. All experiments were carried out on a Core i7 Windows 10 PC with 16 GB of RAM. The following subsections details the findings of our experimental analysis.

B. Runtime Efficiency

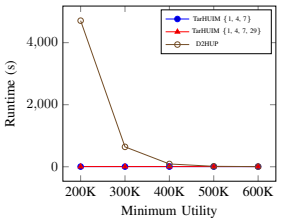
The runtime comparison of TarHUIM (for different target items) and D2HUP are shown in Figure 3.



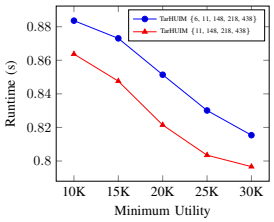
(a) Runtime Foodmart



(b) Runtime Mushroom



(c) Runtime Chess



(d) Runtime Kosarak10K

Fig. 3: Runtime Comparison

It can be observed that irrespective of the type of dataset (dense or sparse), the runtime to discover HUIs is directly affected by the minimum utility threshold. That is, the smaller the threshold, the longer the runtime and vice versa. However, as can be observed in Figures 3a, 3b, 3c and 3d, TarHUIM is more efficient compared to D2HUP irrespective of the target list. It is worth noting that though TarHUIM's implementation is a modified version of the Apriori technique, it is more efficient due to the drastic search space reduction technique employed. It was observed (reference to Figures 3a, 3b, 3c and 3d) that, though the runtime for TarHUIM for the given target list were not significantly different, its runtime is directly affected by the number unique items that co-occur with the users' target list. That is, the more the number of unique items co-occurring with the users' target, the more time TarHUIM takes to discover the targeted HUIs.

C. Reported HUIs and Memory Analysis

1) Reported HUIs: We compare the number of reported targeted high utility itemsets using TarHUIM based on users' target with that of all high utility itemsets reported with D2HUP. Table VII shows the reported HUIs for the respective minimum utility thresholds.

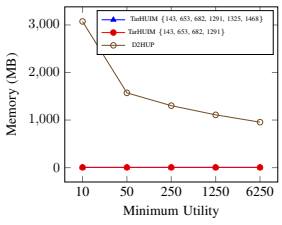
TABLE VII: Reported High Utility Itemsets

	Dataset and Reported HUIs at Given Utility					
	Foodmart		Mushroom		Chess	
	10	6250	60K	140K	200K	600K
D2HUP	233231	4710	4722776	343836	90002856	583
TarHUIM T_1	63	52	36	9	258	1
TarHUIM T_2	15	10	14	9	131	4

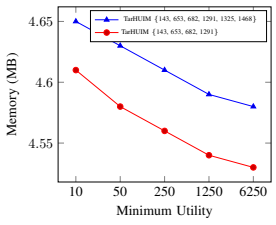
T_1 and T_2 are the user target list for TarHUIM as shown in the legends of Figures 3a, 3b and 3c

As can be observed in Table VII, for any given threshold and users' target list, TarHUIM, as expected, reports a smaller number of targeted HUIs compared to all HUIs reported by D2HUP. For both TarHUIM and D2HUP, as expected and shown in Table VII, the higher the minimum utility thresholds, the smaller the number of reported HUIs and vice versa. It is worth noting that, for targeted HUIM techniques that use the *post-processing*, all the HUIs reported by D2HUP will be detected for the given thresholds before the search among the reported HUIs is performed to identify the targeted HUIs.

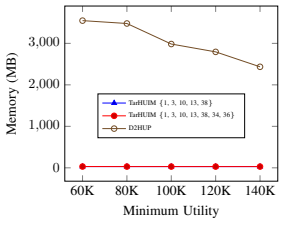
2) Memory Usage: For lack of space, we only show the memory usage for the Foodmart and Mushroom datasets.



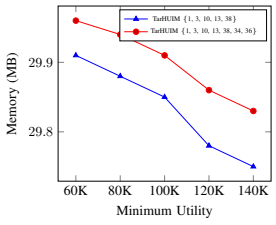
(a) Memory Foodmart



(b) Foodmart TarHUIM



(c) Memory Mushroom



(d) Mushroom TarHUIM

Fig. 4: Memory Usage Comparison

As can be observed in Figures 4a and 4c, TarHUIM is more memory efficient compared to D2HUP. Again this memory efficiency in TarHUIM is due to the search space reduction technique employed. Figure 4b and 4d show in details the memory used by TarHUIM based on the users' target list. It was observed that in the dense datasets (reference to Figure 4d, TarHUIM uses more memory compared to the sparse dataset (reference to Figure 4b since more unique items co-occur with the users' target list in the dense datasets compared to the sparse datasets.

VI. Conclusion

In this paper, we have proposed a one phase target high utility itemset miner (TarHUIM). Unlike existing target high utility itemset mining techniques, TarHUIM scans the database once and drastically reduce the search space to mine users' targeted high utility itemsets. Preliminary experimental analysis on benchmark datasets show that TarHUIM is efficient and can effectively discovers the set of targeted high utility itemsets based on users' targets. Our future works will be towards further improvement of TarHUIM through pseudo-projection and implementing TarHUIM using the FP-growth technique. Additionally, other variations of TarHUIM will be implemented to meet users desire of mining targeted high utility itemsets, for instance, discovering targeted high utility itemsets containing only targeted items.

References

[1] R. Agrawal and T. Imieliński, A. Swami, "Mining Association Rules between Sets of Items in Large Databases". ACM SIGMOD Rec. vol. 22 no. 2, pp. 207-216, 1993.

[2] R. Chan, Q. Yang and Y. D. Shen, "Mining High Utility Itemsets". IEEE International Conference on Data Mining. IEEE Computer Society 2003.

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

[4] T. L. Dam, K.Li, P. Fournier-Viger, and Q. H. Duong, "CLS-Miner: efficient and effective closed high-utility itemset mining". Frontiers of Computer Science, vol 13 no 2, pp. 357-381, 2019.

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

[6] P. Fournier-Viger , J.CW. Lin, T. Dinh and H. B. Le, "Mining Correlated High-Utility Itemsets Using the Bond Measure". In: F. Martínez-Álvarez, A. Troncoso, H. Quintián, E. Corchado, (eds) Hybrid Artificial Intelligent Systems. Lecture Notes in Computer Science, Vol 9648, pp 53–65, Springer, Cham, 2016.

[7] P. Fournier-Viger, CW. Wu, S. Zida and V.S. Tseng, "FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning". In: T. Andreasen, H. Christiansen, JC. Cubero, Z.W. Raś (eds) Foundations of Intelligent Systems. Lecture Notes in Computer Science, Vol 8502, pp. 83–92. Springer, Cham, 2014.

[8] P. Fournier-Viger, Y. Zhang, J.C.W. Lin, D.T. Dinh and H. Bac Le, "Mining correlated high-utility itemsets using various measures". Logic Journal of the IGPL, vol 28 no 1, pp. 19-32, 2020.

[9] P. Fournier-Viger, J.C.-W. Lin, A. Gomariz and T. Gueniche, A. Soltani, Z. Deng, H. T. Lam, "The SPMF Open-Source Data Mining Library Version 2". Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, Cham 2016.

[10] P. Fournier-Viger, J. C. W. Lin, C. W. Wu, V. S. Tseng and U. Faghihi, "Mining minimal high-utility itemsets". International Conference on Database and Expert Systems Applications, pp. 88-101, Springer, Cham, 2016.

[11] P. Fournier-Viger, C. W. Wu, and V. S. Tseng, "Novel concise representations of high utility itemsets using generator patterns". International Conference on Advanced Data Mining and Applications, pp. 30-43, Springer, Cham, 2014.

[12] P. Fournier-Viger, J. C. W. Lin, Q. H. Duong, and T. L. Dam, "FHM+: faster high-utility itemset mining using length upper-bound reduction". International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 115-127, Springer, Cham, 2016.

[13] P. Fournier-Viger, J. C. W. Lin, T. Dinh, and H. B. Le, "Mining correlated high-utility itemsets using the bond measure". International Conference on Hybrid Artificial Intelligence Systems, pp. 53-65, Springer, Cham, 2016.

[14] P. Fournier-Viger, Y. Zhang, J. C. W. Lin, D. T. Dinh, and H. Bac Le, "Mining correlated high-utility itemsets using various measures". Logic Journal of the IGPL, vol 28 no 1, pp. 19-32, 2020.

[15] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation", ACM SIGMOD Rec vol 29 no 2, pp. 1-12, 2000.

[16] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong and V. S. Tseng, "Efficiently Mining Uncertain High-Utility Itemsets". Soft Computing vol 21 no 11, pp. 2801-2820, 2017.

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

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

[19] J. Liu, K. Wang, and B. C. Fung, "Direct discovery of high utility itemsets without candidate generation". 12 th IEEE International Conference on Data Mining. IEEE 2012.

[20] J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, "Targeted high-utility itemset querying". IEEE Transactions on Artificial Intelligence, doi: 10.1109/TAI.2022.3171530, April 2022.

[21] J. Miao, W. Gan, S. Wan, Y. Wu and P. Fournier-Viger, "Towards Target High-Utility Itemsets". arXiv preprint arXiv:2206.06157. 2022 Jun 9.

[22] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang and D. Yang, "H-mine: Hyper-Structure Mining of Frequent Patterns in Large Databases". Proceedings IEEE International Conference on Data Mining, IEEE, USA, 2001.

[23] V. S. Tseng, B.E. Shie, C.W. Wu and S.Y. Philip, "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.

[24] F. C. Tseng, "Mining Frequent Itemsets in Large Databases: The Hierarchical Partitioning Approach". Expert Systems with Applications, vol 40 no 5, pp. 1654-1661, 2013.

[25] V. S. Tseng, C. W. Wu, B.E Shie and P.S. Yu, "UP-growth: An Efficient Algorithm for High Utility Itemset Mining". 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM 2010.

[26] V. S. Tseng, C. W. Wu, P. Fournier-Viger, and S. Y. Philip, "Efficient algorithms for mining top-k high utility itemsets". IEEE Transactions on Knowledge and data engineering, vol 28 no 1, pp. 54-67, 2015.

[27] C. W. Wu, P. Fournier-Viger, S. Y. Philip, and V. S. Tseng, "Efficient mining of a concise and lossless representation of high utility itemsets". 11th IEEE International Conference on Data Mining, pp. 824-83, 2011.

[28] H. Yao, H. J. Hamilton and C. J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases". SIAM International Conference on Data Mining, SIAM 2004.

[29] M. J. Zaki and K. Gouda, "Fast Vertical Mining Using Diffsets". Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003.

[30] S. Zida, P. Fournier-Viger, J.C.-W. Lin, C.-W. Wu and V. Tseng, "EFIM: A Fast and Memory Efficient Algorithm for High-Utility Itemset Mining". Knowledge and Information Systems vol 51 no 2, pp. 595-625, 2016.