CHUQI-Miner: Mining Correlated Quantitative High Utility Itemsets

Mourad Nouioua
School of Humanities and Social Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
University of Bordj Bou Arreridj, Algeria
mouradnouioua@gmail.com

Philippe Fournier-Viger*
School of Humanities and Social Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
philfv8@yahoo.com

Jun-Feng Qu
School of Computer Engineering
Hubei University of Arts and Science
Xiangyang, China
qmxwt@163.com

Jerry Chun-Wei Lin
Dept. of Computing, Mathematics and Physics
Western Norway U. of Applied Sciences (HVL)
Guangzhou, Norway
jerrylin@ieee.org

Wensheng Gan
School of Computer Science
Jinan University
Guangzhou, China
wsgan001@gmail.com

Wei Song
College of Computer Science and Technology
North China University of Technology
Beijing, China
songwei@ncut.edu.cn

***Abstract*—To discover patterns of high importance in data, a popular data science task is high utility itemset mining (HUIM). It aims at discovering all sets of values that have a high utility (importance) in database records. A key application is to find products purchased together in online stores that yield a high profit (utility), as it can provide insights for marketing and product recommendation. But HUIM has two key limitations. First, the discovered patterns do not provide information about the quantities of items. But in real-life, quantities are important (e.g. buying 1 bread is not the same as buying 12 breads). Second, it is observed in real shopping data that many itemsets yield a high utility (profit) but contain weakly correlated items. Such itemsets can be misleading as their joint sale may just appear by chance. This paper addresses both issues by proposing a novel algorithm called CHUQI-Miner (Correlated High Utility Quantitative Itemset-Miner). It extends the state-of-the-art HUQI-Miner algorithm for quantitative high utility itemset mining with the bond correlation measure. This allows finding strongly correlated high utility itemsets with quantities. Experiments on retail data show that the algorithm is efficient and can filter a huge amount of spurious itemsets.**

***Index Terms*—high utility itemsets, correlation, quantitative itemsets, bond measure**

I. Introduction

Data has become a key part of modern information systems. To gain insights from large datasets, many data science algorithms have been developed. In recent years, high utility itemset mining (HUIM) has become a key data science task [1]–[3]. It aims at discovering all high utility itemsets (sets of values) that have a high importance (utility) in data. The concept of utility may take various forms for different applications. The most representative HUIM application is to find sets of products bought together by customers that yield a high profit. In that context, a high utility itemset (HUI) is a set of items that yield a profit that is equal to or greater than a user-defined minimum utility threshold ($minutil$). For a decision-maker, HUIM can reveal useful information about customer behavior that can be used for marketing. For example, it can be found that the items $\{pen, eraser\}$ bought together generate a high profit. However, a problem with traditional HUIM is that HUIs do not give any information about purchase quantities. But this information can be useful for marketing purposes. For instance, it is not the same to buy 1 pen or a box of 24 pens. As a solution, HUIM was recently generalized as the task of high utility quantitative itemset mining (HUQIM) [4]–[7]. The output is a set of HUIs with quantity information, called high utility quantitative itemsets (HUQIs). For instance, a HUQI may be $\{$*pen:12-24, eraser:1*$\}$ indicating that purchasing 12 to 24 pens with one eraser is profitable. The problem of HUQIM is much more difficult than HUIM because not only various items may be combined to make itemsets but also different quantities and ranges of quantities.

Though HUQIM is an emerging research area and many algorithms were designed [4]–[7], a major problem is that HUQIs often contain items that are weakly correlated. For instance, a HUI $\{pen : 12 - 24, diamond\}$ may be profitable but there are no relationship between pens and diamonds. Hence, this itemset is a HUQI simply because diamonds are expensive and pens are purchased by many customers. A traditional HUQIM algorithm can thus output many spurious patterns, which can be misleading for decision-makers.

This paper addresses the above limitation by proposing a novel algorithm named CHUQI-Miner (Correlated High Utility Quantitative Itemset-Miner). The algorithm integrates a measure of correlation named *the bond* in the search for HUQIs. This allows filtering spurious patterns to only find strongly correlated HUQIs (CHUQIs). As it will be shown in the experimental evaluation on real shopping data, this allows filtering out a huge amount of spurious patterns.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 describes preliminaries and the

*Corresponding author.

problem definition. Section 4 presents the developed CHUQI-Miner algorithm. Section 5 describes experiments. Finally, Section 6 concludes the paper.

II. Related Work

Since the 1990s, itemset mining has attracted the attention of many researchers. The first algorithms were designed to find frequent itemsets [8], that is sets of values appearing frequently in data. Frequent itemset mining (FIM) algorithms efficiently find frequent itemsets without checking all possibilities by relying on the anti-monotonicity property of the support. This property states that the support (occurrence frequency) of an itemset cannot be greater than that of its subsets. Because frequent itemsets are not always interesting, other pattern selection criteria have been proposed. In high utility itemset mining (HUIM), a generalization of FIM, the goal is to find itemsets that have a high utility (importance), where the utility can be defined in terms of various criteria such as the profit or the time [9]–[11]. HUIM is a difficult problem because the utility function for selecting patterns is not anti-monotonic nor monotonic. To cope with this challenge, HUIM algorithms rely on anti-monotonic upper bounds on the utility to reduce the search space such as the transaction weighted utility (TWU) [10] and the remaining utility upper bound [12]. HUIM algorithms can be classified into two categories: Two-phase based algorithms and one-phase based algorithms [1], [2]. As indicated in its name, the two-phase based algorithms are based on two phases [10], the first phase may overestimate some low utility itemsets, but it never underestimates any high utility itemset. In the second phase, these algorithms perform another database scan to calculate the exact utility of candidates and filter low-utility itemsets. The first two phase based algorithm is TP [10]. Based on TP algorithm, other algorithms were proposed such as: IHUP [13] and UPGrowth [14]. It was shown that two-phase based algorithms generate a very large number of candidate itemsets in the first phase. To address this problem, one-phase based algorithms have emerged from the appearance of HUI_Miner algorithm [12]. These algorithms immediately identify high utility itemsets in only one phase without need to generate candidate itemsets. Several one-phase based algorithms have been proposed such as: FHM [9], mHUIMiner [11], HMiner [15] and ULB-Miner [16].

To consider quantities in HUIM, several HUQIM algorithms were designed. The HUQA [7] algorithm adopts a candidate generation approach and reduces the search space using a k-support bound measure. To avoid doing many database scans, a faster algorithm called VHUQI [4] utilizes a vertical data representation. More recently, the HUQI-Miner algorithm [5] applied two upper bounds based on the TWU and remaining utility to reduce the search space. An extension called FHUQI-Miner [6] was then introduced with additional search space reduction strategies. FHUQI-Miner is to our best knowledge the state-of-the-art HUQIM algorithm [6].

To address the problem that items are often weakly correlated in itemsets in FIM and HUIM, several correlation measures have been integrated such as the *bond* [17], [18], *all-confidence* [18], [19] and *affinity* [20], [21]. However, none of them has been used in HUQIM.

TABLE I: A quantitative transaction database

T_{id}	Transaction
T_1	(A,2) (B,5) (C,2) (D,1)
T_2	(B,4) (C,3)
T_3	(A,2) (C,2)
T_4	(A,2) (B,6) (D,1)

TABLE II: External utility values of items

Item i	A	B	C	D
p_i	3	1	2	2

III. Preliminaries and Problem Definition

The next paragraph presents preliminaries and then the proposed problem of CHUQIM.

Let I be a finite set of N items (symbols), denoted as $I = \{I_1, I_2, \dots, l_N\}$. A positive number p_i indicates the relative importance (e.g. unit profit) of each item $i \in I$, and is called its *external utility*. A *quantitative transaction database* (QTDB) is a set of one or more transactions $D = \{T_1, T_2, \dots, T_M\}$. Each transaction $T_d \in D$ is a set of exact Q-items, $T_d = \{x_1, x_2, \dots, x_k\}$ and has a unique identifier d. An *exact Q-item* x is a tuple (i, q) telling that q units of item $i \in I$ have been purchased. As example, consider the QTDB of Table I, which will be used as running example. It has four transactions (T_1, T_2, T_3, T_4) made up of four items $I = \{A, B, C, D\}$. The third transaction T_3 contains two Q-items, $(A, 2)$ and $(C, 2)$, indicating that a customer bought two units of item A, and two units of item C. The external utility values of items are listed in Table II. For instance, $p_D = 2$ means that the sale of each unit of item D yields a profit of 2$.

To find HUQIs, two types of Q-items are considered, namely the range Q-items and the exact Q-items. The range Q-items do not exist in an input database but they are created when searching for itemsets by combining exact Q-items and/or range Q-items. A *range Q-item* is a tuple (i, l, u) specifying that from l to u units of item i have been purchased. It is said to have an interval (Q-interval) of size $u - l + 1$. As example, the interval size of range Q-item $(A, 2, 4)$ is 3. It can be observed that any exact Q-item (i, q) can be represented as a range Q-item (i, q, q). A *Q-itemset* is a finite set of Q-items. If a Q-itemset has k items, it is called a *k-Q-itemset*. If a Q-itemset has a Q-item with a Q-interval of size no less than 1, it is a *range Q-itemset*. As example, $[(A, 1)(C, 2, 3)]$ is a range 2-Q-itemset.

An exact Q-item $x = (i, q)$ is *contained* in a range Q-item $y = (j, l, u)$ if $i = j$ and $q \in [l, u]$. Besides, a range Q-item $z = (j', l', u')$ is contained in y if $j' = j$, $u \geq u'$ and $l \leq l'$. For instance, the exact Q-item $(A, 2)$ is contained in $(A, 1, 5)$, and this latter is contained in range Q-item $(A, 1, 6)$.

Let there be an exact Q-item x, a range Q-item y and a transaction T_d. If $x \in T_d$, then it is said that x *occurs* in T_d. If a Q-item of T_d is contained in y, then it is said that y *occurs* in T_d. As example, $(C, 2)$ occurs in T_1 and T_3, while $(C, 2, 3)$ occurs in T_1, T_2 and T_3.

Let there be a Q-itemset X and a transaction T_d. If each Q-item $x \in X$ occurs in T_d, then X is said to *occur in* T_d. The *occurrence-set* of X is the set of transactions where X occurs, denoted as $OCC(X)$. The support of X is defined as $sup(X) = |OCC(X)|$ and also called occurrence frequency. As example, let $X = [(A, 2), (C, 2)]$. Then, $sup(X) = |OCC(X)| = |\{T_1, T_3\}|$ = 2.

HUQIM aims at enumerating all the Q-itemsets having a high utility (e.g. profit) [4]. Let there be a transaction T_d. For a Q-item $x = (i, q)$, the *utility* in T_d is defined as $u(x, T_d) = \ p_i \times q$. For a range Q-item $x = (i, l, u)$, the utility in T_d is defined as $u(x, T_d) = \sum_{j=l}^{u} u((i, j), T_d)$. For a Q-itemset X, the utility in T_d is defined as $u(X, T_d) = \sum_{x \in X} u(x, T_d)$. For instance, $u((A, 2), T_3) = 3 \times 2 = 6$, $u((C, 2, 3), T_3) = u((C, 2), T_3) + u((C, 3), T_3) = 4 + 0 = 4$ and $u([(A, 2)(D, 1)], T_1) = 6 + 2 = 8$. For a database D, the utility of X is defined as $u(X) = \sum_{T_d \in OCC(X)} u(X, T_d)$ [4].

For example, $u([(B, 5, 6)(D, 1)])$ = $u([(B, 5)\ (D, 1)],\ T_1)$ $+u([(B, 6)\ (D, 1)], T_4)$ = 7 + 8=15 and $u([(A, 2)\ (C, 2)])$ = $u([(A, 2)(C, 2)]$,T_1) + $u([(A, 2)(C, 2)]$,T_3)=10 + 10=20.

The utility of a transaction T_d is $TU(T_d) = \sum_{y \in T_d} u(y, T_d)$ (The sum of its Q-items' utility). *The utility of database* D is defined as $\sigma = \sum_{T_d \in D} TU(T_d)$ (the sum of its transactions' utility). For instance, $TU(T_2) = u((B, 4), T_2) + u((C, 3), T_2) = 4 + 6 = 10$ and $\sigma = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) = 51$.

Formally, the problem of HUQIM consists of enumerating the *high utility quantitative itemsets* (HUQIs) in a database D. A Q-itemset X is a HUQI if its utility is no less than a user-defined minimum utility threshold $minutil \in [0, \sigma]$, i.e. $u(X) \geq minutil$. If $u(X) < minutil$, it is a *low utility quantitative itemset (LUQI)* [4]. For example, if $minutil = 10$, the Q-itemset $[(A, 2)(D, 1)]$ is a HUQI because $u([(A, 2)(D, 1)]) = 16 \geq 10$.

HUQIM is an extension of HUIM that is more difficult to solve. The search space in HUQIM is larger than in HUIM because not only different itemsets need to be evaluated but also different quantities for each itemset. For an item, different quantities yield different Q-items such as $B(2)$ and $B(3)$ for the item B. Moreover, various quantities for an itemset can produce different range Q-itemsets such as $B(2, 3)$ and $B(2, 4)$. Besides, another challenge is that the utility of a Q-itemset is not *anti-monotonic* nor *monotonic*, that is a Q-itemset may have a utility that is smaller, greater or equal to that of its subsets or supersets [4]. As a consequence, the utility cannot be directly used to prune the search space.

To mine HUQIs, recent algorithms apply two operations [5], [6]. The join operation appends two Q-itemsets to obtain a larger Q-itemset (a Q-itemset with one more Q-item). The merge operation takes two Q-itemsets having Q-items with consecutive quantities and merges them to obtain a Q-itemset with a larger Q-interval. For instance, a join of Q-items $(B, 4)$ and $(C, 3)$ produces the Q-itemset $[(B, 4)(C, 3)]$, while a merge of $(C, 2)$ and $(C, 3)$ yields the range Q-itemset $[(C, 2, 3)]$. An interesting property of the HUQIM problem is that pairs of LUQIs can be sometimes combined to produce a range HUQI. To avoid generating too many combinations, a Quantitative Related Coefficient (qrc) is used, such that $qrc > 0$. This coefficient restricts the size of Q-intervals produced by merge operations. Using qrc, a merge operation can be done with two Q-itemsets $X = [(x_1, l_1, u_1),\ (x_2, l_2, u_2), \ldots,$ $(x_k, l_k, u_k)]$ and $Y = [(y_1, l'_1, u'_1),\ (y_2, l'_2, u'_2), \ldots, (y_k, l'_k, u'_k)]$ to produce a range Q-itemset $Z = [(i_1, l_1, u_1),\ (i_2, l_2, u_2), \ldots,$ $(i_k, l_k, u'_k)]$ if the following conditions are met [5]:

(1) X and Y are candidate Q-itemsets, that is $\frac{minutil}{qrc} \leq u(X) \leq minutil$ and $\frac{minutil}{qrc} \leq u(Y) \leq minutil$.
(2) The first $(k-1)$ Q-items of X and Y are identical, i.e, $\forall (1 \leq i \leq k-1)$: $x_i = y_j$, $l_i = l'_j$ and $u_i = u'_j$.
(3) The Q-interval of the last Q-item to be generated should be less than or equal to qrc, i.e, $u'_k - l_k \leq qrc$.
(4) For the last Q-item, $x_k = y_k$ and $l'_k = (u_k + 1)$.

There are three methods to combine Q-itemsets, called $Combine\_All$, $Combine\_\ Min$, and $Combine\_Max$ [5], [22], which can produce different sets of HUQIs. Using $Combine\_All$, all possible range HUQIs are created by combining candidate Q-itemsets, or candidate Q-itemsets with range Q-itemsets [5]. For instance, consider a set of candidate Q-itemsets $\{[(B, 2)(D, 3)],\ [(B, 2)(D, 4)],$ $[(B, 2)(D, 5)]\}$. The $Combine\_All$ method will generate three range Q-itemsets: $[(B, 2)(D, 3, 4)]$, $[(B, 2)(D, 4, 5)]$ and $[(B, 2)(D, 3, 5)]$. After combination, only HUQIs are kept from the generated Q-itemsets.

Because of the limited space, this section only describes $Combine\_All$. The reader is referred to [5], [6] for more details about the join and merge operations.

Although HUQIM is useful, a major limitation of HUQIM algorithms is that the correlation between items is not assessed. This can lead to finding many spurious HUQIs. To address this issue, this paper proposes the task of Correlated HUQIM (CHUQIM) where *the bond correlation measure* is integrated in HUQIM to prune all weakly correlated HUQIs. While many correlation measures have been used in FIM and HUIM, the bond is selected because it is popular and easy to interpret [17], [18].

Definition 1 (bond measure): Given a Q-itemset X, the $Bond$ measure of a pattern is based on its support count (sup) and its *disjunctive support*. The *disjunctive support* of X, denoted as $disup(X)$, is defined as: $disup(X) = |\{T_d \in D | X \cap T_d \neq \emptyset\}|$. Based on these two supports, the *bond* of Q-itemset X, written as $bond(X)$, is given by $bond(X) = sup(X)/disup(X)$.

Any Q-itemset's bond value is always in the range [0,1]. Moreover, one important specificity of the *bond* measure is that it is anti-monotonic [17]. Therefore, it can be used as an upper-bound to prune non-correlated Q-itemsets with all their supersets.

Correlated High Utility Quantitative Itemset Mining (CHUQIM). The task of correlated HUQIM is to enumerate all correlated HUQIs (CHUQIs). A HUQI X is a CHUQI if $bond(X) \geq minbond$, where $minbond$ is set by the user.

For example, if $minutil = 10$ and $minbond = 0.5$, the CHUQIs are: $[(A, 2) : 18, 1]$, $[(B, 4, 6) : 15 : 1]$, $[(B, 5, 6) :$

11 : 1], $[(C,2,3) : 14 : 1]$, $[(A,2)(C,2) : 20 : 0.66]$, $[(A,2)(D,1) : 16 : 0.66]$,$[(B,4)(C,3) : 10 : 1]$ where the first number after the colon represents the pattern's utility and the second number is the pattern's bond.

IV. The CHUQI-Miner algorithm

Traditional HUQIM algorithms such as HUQI-Miner [5] and FHUQI-Miner [6] discover a set of itemsets with their quantities that yield to high profits in a transaction database. However, these algorithms do not take into consideration the correlation between items. Because of that, HUQIM algorithms may find a large set of Q-itemsets having high utilities where most are weakly correlated in the database. To address this limitation, a novel algorithm is designed, called CHUQI-Miner (Correlated High Utility Quantitative Itemset Miner). CHUQI-Miner is based on FHUQI-Miner with additional pruning strategies to keep only CHUQIs. In the following, the Q-item utility-list structure is first introduced. Then, we present the pruning strategies adopted by CHUQI-Miner to prune low utility Q-itemsets. Finally, the CHUQI-Miner algorithm is described with pruning strategies adopted to prune weakly correlated Q-itemsets.

Q-items utility-lists. Similarly to FHUQI-Miner, CHUQI-Miner is also a utility-list based algorithm. A utility-list is a structure that is used to represent each pattern (Q-itemset in our case) of the mining problem. It is used since it allows to quickly find the utility of patterns during the mining process without repetitively reading the database [6], [12]. Formally, given a database D and also a total order relation $\prec$ on distinct Q-items in D, the utility-list of a Q-itemset $X \subseteq D$ is denoted as $UL(X)$. $UL(X)$ stores the utility information of X in all transactions that contain X. The utility information is stored in a set of tuples that have the form $(T_d, Eutil, Rutil)$ where d is the identifier of a transaction T_d such that $X \subseteq T_d$, $Eutil$ is the utility of X in T_d, i.e, $Eutil(X, T_d) = u(X, T_d)$, and $Rutil$ is the remaining utility of X in T_d, denoted as $Rutil(X, T_d)$. $Rutil(X, T_d) = \sum_{i \in T_d/X} u(i, T_d)$, where T_d/X contains the set of all Q-items that come after Q-items of X according to the predefined order $\prec$. Moreover, $UL(X)$ contains also two additional values $SumEutil$ and $SumRutil$. $SumEutil$ is the sum of all $Eutils$ of X. It gives the exact utility of X in D, i.e, $u(X)$, while $SumRutil$ is the sum of all $Rutils$ of X. $SumRutil$ will be further used to prune the search space using the remaining utility pruning strategy.

In CHUQI-Miner, utility-lists not only store the utility information. But they are also adopted to store information for computing the $bond$ measure to identify correlated patterns. More precisely, it is necessary to find the disjunctive support ($disup$) and support count (sup) of different Q-itemsets during the search for patterns. The support count of a Q-itemset X can be easily calculated from its utility list. It is equal to the number of tuples in $UL(X)$, i.e. $|UL(X)|$. To calculate the disjunctive support, a naive approach is to scan the database for each pattern. However, this is impracticable and requires a long time. To overcome this problem, we adopt *the disjunctive bit vector structure* [18]. More precisely, each utility-list contains a bit vector that stores the occurrence set of its corresponding Q-itemset. Given a Q-itemset X, the i^{th} bit of the bit vector of X is set to 1 if $\exists j \in X$ such that $j \in T_i$. Otherwise, this bit is set to 0. First, the disjunctive bit vectors of initial Q-items are calculated during the first database scan. Then, during the recursive mining search, disjunctive bit vectors of larger Q-itemsets can be easily constructed by performing a logical OR operation on smaller Q-itemsets.

Back to our running example, suppose that the order $\prec$ between Q-items is: $(A,2) \prec (B,4) \prec (B,5) \prec (B,6) \prec (C,2) \prec (C,3) \prec (D,1)$, the utility-list of Q-item $[(A,2)]$ has three tuples which are: $(T_1,6,11)$, $(T_3,6,4)$ and $(T_4,6,8)$ with $SumEutil = 18$ and $SumRutil = 23$. The disjunctive bit vector of $[(A,2)]$ is 1011. The utility of Q-item $[(C,2)]$ contains two tuples $(T_1,4,2)$ and $(T_3,4,0)$ with $SumEutil = 8$ and $SumRutil = 2$. The disjunctive bit vector of $[(C,2)]$ is 1010. The utility-list of Q-itemset $[(A,2)(C,2)]$ has two tuples $(T_1,10,2)$ and $(T_3,10,0)$ with $SumEutil = 20$ and $SumRutil = 2$. The disjunctive bit vector of $[(A,2)(C,2)]$ is obtained by performing a logical OR between the disjunctive bit vectors of $[(A,2)]$ and $[(C,2)]$. More precisely, The disjunctive bit vector of $[(A,2)(C,2)]$= 1011 OR 1010=1011.

Using utility-lists allows to overcome the problem of repetitively scanning the database and allows to obtain exact utilities of different Q-itemsets. However, it is impracticable to construct utility-lists for all possible Q-itemsets, especially, with databases that contain a large set of items. Therefore, CHUQI-Miner adopts some pruning strategies to prune low utility and weakly correlated Q-itemsets with their transitive extensions that are unpromising as well.

Pruning Strategies. CHUQI-miner considers that Q-itemsets should have high utilities and should contain correlated Q-items. Thus, there are two categories of pruning strategies, one for eliminating low utility Q-itemsets and the other for eliminating non correlated Q-itemsets. In this section, we explain the pruning strategies adopted by CHUQI-Miner to eliminate low-utility Q-itemsets. The other strategies that are used to prune non correlated Q-itemsets will be discussed in the next section with the CHUQI-miner algorithm.

Strategy 1: TWU Pruning Strategy.

Given a Q-itemset X, the transaction weighted utility of X, denoted as $TWU(X)$, is the sum of transactions utilities of transactions containing X. Formally, $TWU(X) = \sum_{T_d \in OCC(X)} TU(T_d)$. The TWU is anti-monotonic and it is an upper bound on the utility of Q-itemsets and their supersets. Therefore, it can be used to prune unpromising Q-itemsets.

Given a Q-itemset X, if $TWU(X) < \frac{minutil}{qrc}$, then X with all its transitive extensions can be eliminated because they are not promising.

Back to our example, $TWU([(A,2)(B,6)])$=$u(T_4)$= 14. If $minutil = 30$ and $qrc = 2$, then $TWU([(A,2)(B,6)]) = 14 < \frac{30}{2}$. Thus, $[(A,2)(B,6)]$ can be pruned with all its extensions because they are not promising.

Strategy 2: Remaining utility pruning strategy.

This pruning strategy is based on the $SumRutil$ of a Q-itemset stored in its utility-list.

Given a Q-itemset X, if $SumEutil(X)+SumRutil(X) < minutil$ then X with all its supersets can be eliminated because they are LUQIs. For example, $SumEutil([(A,2)(D,1)])$=16 and $SumRutil([(A,2)(D,1)])$=0. If $minutil$ = 30, then $[(A,2)\ (D,1)]$ can be eliminated with all its extensions.

Strategy 3: Co-occurrence pruning strategy based on the utility measure.

The co-occurrence pruning strategy allows to eliminate LUQIs with their transitive extensions without even the need to construct their utility-lists. It is based on *the TQCS structure*, (TWU of Q-items Co-occurrence based Structure), which is constructed during the second scan of the database. The TQCS stores all pairs of Q-items, i.e, 2-Q-itemsets, that have co-occurred in the database with their utility information [6].

Formally, the TQCS contains a set of tuples that have the form (a, b, c, d), where a and b are two Q-items that co-occurred in the database, c is the TWU of the Q-itemset $[ab]$, i.e, $c = TWU([ab])$, and d is the support count of Q-itemset $[ab]$, i.e, $d = sup([ab])$.

Using the TQCS, there are two strategies to prune low utility Q-itemsets, namely *Exact Q-items Co-occurrence Pruning Strategy (EQCPS)* and *Range Q-items Co-occurrence Pruning Strategy (RQCPS)*. EQCPS prunes unpromising exact Q-itemsets while RQCPS is used for pruning unpromising range Q-itemsets. The reader can refer to [6] to see a detailed explanation of these strategies with illustrative examples.

The Algorithm. This section describes the main steps of CHUQI-Miner to find CHUQIs including the raising strategies that are used. CHUQI-Miner (Algorithm 1) has five input parameters: (1) The transaction database D, (2) The user-defined minimum utility threshold ($minutil$), (3) The user-defined minimum bond ($minbond$) which should be selected between 0 and 1, (4) The combining method CM and (5) The quantitative related coefficient qrc. CHUQI-Miner provides the set of correlated HUQIs in D.

CHUQI-Miner starts by scanning the database to calculate the TWU of each Q-item in D (line 1). Then, CHUQI-Miner applies the TWU pruning strategy (strategy 1) to keep only promising Q-items and remove non promising Q-items (line 2).

The second database scan is then performed to create initial utility-lists of promising Q-items (line 3). Besides, promising Q-items are first ordered according to the pre-defined order $\prec$. Then, initial utility-lists of these Q-items are built and the TQCS structure is also constructed.

Then, CHUQI-Miner checks the utility of each promising Q-item. If the current Q-item has a utility that is greater than or equal to the $minutil$ value, then this Q-item is output because it is a CHUQI (lines 5-7). Otherwise, two tests are performed to put this Q-item either in the set of candidate Q-items (C) (line 10) or in the set of Q-items to be explored (E) (line 9).

Note that, CHUQI-Miner does not check the bond of promising Q-items because the bond of any Q-itemset that has

Algorithm 1: The CHUQI-Miner algorithm

Input : D: The quantitative transaction database, $minutil$: The user-defined minimum utility threshold, $minbond$: The user-defined minimum $bond$ value, CM: The combining method ($Combine\_Min$, $Combine\_Max$ or $Combine\_All$), qrc: The quantitative related coefficient.
Output: The set of correlated HUQIs.

```
1  First database scan to calculate the $TWU$ of each Q-item;
2  Create initial set of promising Q-items $P^*$ such that
     $\forall x \in P^* : TWU(x) \geq \frac{minutil}{qrc}$ and discard unpromising
     Q-itemsets; // Strategy 1
3  Second database scan to create utility-lists of promising Q-items $ULs(P^*)$
     and build the TQCS structure;
4  foreach $x \in P^*$ do
5      if $UL(x).SumEutil \geq minutil$ then
6          $H = H \cup x$;
7          Output $x$;
8      else
9          if $UL(x).SumEutil + UL(x).SumRutil \geq minutil$ then
             $E = E \cup x$ ;
10         if $\frac{minutil}{qrc} \leq UL(x).SumEutil \leq minutil$ then
             $C = C \cup x$ ;
11 Discover High Utility range Q-itemsets ($HR$) using $CM$ and $C$;
12 $QI_s \leftarrow sort(H \cup E \cup HR)$;
13 Recursive_Mining_Search($\emptyset$, $QI_s$, $UL_s(QI_s)$, $P^*$, $minutil$,$minbond$,
     $CM$, $qrc$);
```

only one Q-item, i.e, Q-item, is always 1. Thus, if a Q-item is a HUQI, it is a CHUQI.

If the set of candidate Q-items (C) is not empty, CHUQI-Miner performs the combination process to generate correlated high utility range Q-itemsets (line 11). It is worth noticing that, range Q-itemsets do not explicitly exist in the database but they are obtained after combining exact or range Q-items. Thus, it is assumed that the bond measure of range Q-itemsets is equal to 1 and it suffices to just check their utilities.

At this point, CHUQI-Miner calls the recursive mining procedure which is presented in Algorithm 2 (line 13).

The recursive mining search algorithm of CHUQI-Miner is similar to that used in FHUQI-Miner [6] with some changes to discover HUQIs that are correlated. The recursive mining search algorithm is a depth-first search algorithm that has eight parameters as input: (1) P: The prefix Q-itemset, (2) QI_s: The set of extensions of P, (3) $UL(QI_s)$: Utility-lists of QI_s, (4) P^*: The list of promising Q-itemsets (5) $minutil$: The user-defined minimum utility threshold, (6) $minbond$: The user-defined minimum $bond$ value, (7) CM: The combining method and (8) qrc: The quantitative related coefficient.

The recursive mining search algorithm explores the search space starting from smaller Q-itemsets to larger Q-itemsets based on the following principle: Given a prefix Q-itemset P and a list of Q-itemsets to be explored QI_s, for each Q-itemset Px in QI_s, the algorithm traverses all extensions Py such that $x \prec y$ to explore larger Q-itemsets that have the form Pxy (lines 1-3). For each extension Pxy, the algorithm performs a series of three pruning strategies to decide whether it is necessary to construct the utility list of Pxy or to pass directly to the next extension Py. More precisely, the algorithm starts by applying Strategy 3 to check the TWU of xy (line 4). Then, Strategy 4 is applied to check if xy is strongly or weakly correlated (line 5).

Strategy 4: Co-occurrence pruning strategy based on the

Algorithm 2: *Recursive_Mining_Search*

Input : P: The prefix Q-itemset, QI_s: The list of Q-itemsets, $UL_s(QI_s)$: Utility lists of Q-itemsets, P^*: The list of promising Q-itemsets, $minutil$: The user-defined minimum utility threshold, $minbond$: The user-defined minimum $bond$ value, CM: The combining method, qrc: The quantitative related coefficient.
Output: The set of CHUQIs with respect to prefix P.

```
1  foreach $[Px]$ such that $x \in QI_s$ do
2      $QI_s \leftarrow \emptyset$; $newP^* \leftarrow \emptyset$;
3      foreach $[Py]$ such that $y \in P^*$ and $y \succ x$ do
4          Apply Co-occurrence pruning strategy based on the utility measure;// Strategy 3
5          Apply Co-occurrence pruning strategy based on the bond measure;// Strategy 4
6          $Bit\_vector(Pxy) \leftarrow PerformOR(Bit\_vectors(Px, Py))$;
7          if $\frac{min(sup(Px), sup(Py))}{|Bit\_vector(Pxy)|} < minbond$ then
8              // Strategy 5
9              Go to next $Py$;
10         $Z \leftarrow [Pxy]$; $UL(Z) = Construct(P, Px, Py)$;
11         if $UL(Z).TWU \geq \frac{minutil}{qrc}$ and $bond(Z) \geq minbond$ then
12             $newP^* = newP^* \cup Z$;
13             if $UL(Z).SumEutil \geq minutil$ then
14                 Output $Z$;
15                 $H = H \cup Z$;
16             else
17                 if $UL(Z).SumEutil + UL(Z).SumRutil \geq minutil$ then $E = E \cup Z$;
18                 if $\frac{minutil}{qrc} \leq UL(Z).SumEutil \leq minutil$ then // Strategy 2
19                     $C = C \cup Z$;
20     Discover High Utility range Q-itemsets $HR$ using $CM$ and $C$;
21     $QI_s \leftarrow (H \cup E \cup HR)$;
22     Recursive_Mining_Search($Px$, $QI_s$, $UL_s(QI_s)$, $newP^*$, $minutil$, $minbond$, $CM$, $qrc$);
```

correlation measure.

This strategy is inspired by the EQCPS used in strategy 3. However, it is used to prune the weakly correlated Q-itemsets. Given a prefix Q-itemset P and two consecutive Q-itemsets Px and Py. If $\frac{min\{sup(Px), sup(Py), sup(xy)\}}{max\{disup(Px), disup(Py)\}} < minbond$, the utility list of Pxy should not be constructed because Q-itemset Pxy is weakly correlated with all its supersets. Note that, $sup(xy)$ is extracted from the TQCS structure. Mathematically, the left end of the inequality is an upper bound on the bond measure of Pxy, i.e, $\forall P, x, y \in D$, $\frac{min\{sup(Px), sup(Py), sup(xy)\}}{max\{disup(Px), disup(Py)\}} \geq bond(Pxy)$. Therefore, the left end of the inequality can be used to prune the search space.

For instance, if $P = \emptyset$, $Px = [(A, 2)]$, $Py = [(D, 1)]$ and $minbond = 0.8$, then $\frac{min\{3,2,2\}}{max\{3,2\}}$=0.66. This value is less than $minbond = 0.8$. Thus, the algorithm can pass directly to the next Py extension.

After applying the two above co-occurrence pruning strategies (Strategies 3 and 4), the algorithm performs Strategy 5 which is applied just before the construction process (lines 6-8).

Strategy 5: Pruning supersets of non correlated Q-itemsets. An additional strategy is used in CHUQI-Miner to prune the weakly correlated Q-itemsets with their supersets without constructing their utility-lists. Based on the fact that the bond measure is anti-monotonic [17], [18], if $\frac{min\{sup(Px), sup(Py)\}}{disup(Pxy)} < minbond$, then a Q-itemset Pxy can be pruned with all its extensions because these Q-itemsets are weakly correlated. $disup(Pxy)$ is extracted by performing a logical OR between disjunctive bit vectors of Px and Py.

For example, suppose that, $P = \emptyset$, $Px = [(A, 2)]$, $Py = [(B, 5)]$ and $minbond = 0.5$, $\frac{min\{3,1\}}{|1011\ OR\ 1000|}$=$\frac{1}{3}$ $= 0.33 < 0.5$. Thus, the extension $[(A, 2)(B, 5)]$ can be pruned with all its supersets.

If the combination Pxy is not pruned by the above pruning strategies, the algorithm performs the join process to build $UL(Pxy)$ from $UL(P)$, $UL(Px)$ and $UL(Py)$ (line 9).

During the construction process, Strategy 6 is applied.

Strategy 6 (Early Abandoning Utility-Lists Construction). This strategy is performed during the construction of the Q-itemset Pxy. More precisely, the construction process is stopped if a specific condition is met because it is sure that the resulted Pxy is not promising. Given the $minbond$ value and $disup(Pxy)$ which has been previously calculated by performing the logical OR operator between disjunctive bit vectors of Px and Py, the Q-itemset Pxy can be a CHUQI if and only if $\frac{sup(Pxy)}{|disup(Pxy)|} \geq minbond$, i.e., $sup(Pxy) \geq |disup(Pxy)| * minbond$. Therefore, if $sup(Pxy) < |disup(Pxy)| * minbond$, then the construction of Pxy is stopped. To see more details how Strategy 6 works with the construction process, we suggest the reader to refer to [23].

If the construction process is not stopped by Strategy 6, the extension Pxy is put in the list of new promising Q-itemsets $newP^*$ (line 11) and the algorithm performs similar tests as in Algorithm 1. More precisely, based on the utility of Pxy, Pxy can be: A CHUQI ($Pxy \in H$), a candidate Q-itemset ($Pxy \in C$) or a Q-itemset to be explored ($Pxy \in E$)(lines 12-17).

After traversing all extensions of P, the algorithm performs the combination process to discover correlated high utility range Q-itemsets (line 18). Then, the new QI_s is created from sets H,C and HR (line 19).

At this point, the recursive mining search algorithm performs a recursive call with the new prefix Px (line 20). Since the algorithm starts from the prefix $P = \emptyset$, CLH-Miner is able to identify all CHUQIs in the database D.

V. EXPERIMENTS

Experiments were performed to assess the performance of CHUQI-Miner. They were done on a workstation running Windows 7, having an Intel(R) i7-8700 processor and 16 GB of RAM. CHUQI-Miner was compared with the state of art FHUQI-Miner HUQIM algorithm. CHUQI-Miner was run with $minbond$ values varied from 0.1 to 0.9. Henceforth, the notation (CHUQI x) refers to CHUQI-Miner with $minbond = x$. FHUQI-Miner and CHUQI-Miner are implemented in Java.

Datasets. Algorithms were evaluated on four datasets that have various characteristics. These datasets were downloaded from the SPMF data mining library [22] and are often used in the HUQIM literature to evaluate algorithms.

Mushroom is a dense dataset with 8,416 transactions and 128 items. *Connect* is a dense dataset of long transactions. It contains 67,557 transactions and 129 items. *Foodmart* is a sparse dataset with 4,141 transactions and 1,559 distinct items.

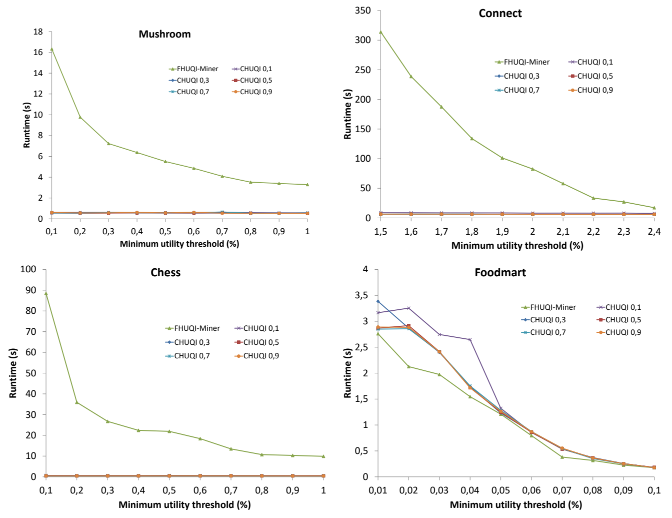

Fig. 1: Runtimes of FHUQI-Miner and CHUQI-Miner with different $minbond$ and $minutil$ values

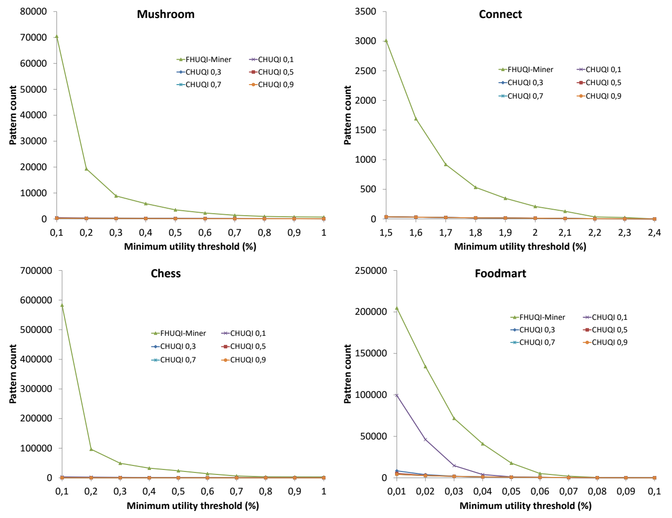

Fig. 2: Patterns found using FHUQI-Miner and CHUQI-Miner for different $minbond$ and $minutil$ values

Chess is a dense dataset with long transactions. It has 3,196 transactions and 75 items. Besides, the quantities of items are varied from 1 to 5.

Runtime comparison on different datasets. The first experiment was done to compare the execution time of CHUQI-Miner with FHUQI-Miner for different values of $minutil$ and $minbond$. Results are presented in Fig. 1. It can be observed that mining CHUQIs is much faster than mining HUQIs in case of dense datasets especially for small values of $minutil$ and large values of $minbond$. More precisely, for Mushroom, Connect and Chess, CHUQI-Miner is up to 29, 49 and 194 times faster than FHUQI-Miner. On average, CHUQI-Miner is 11.14, 17.91 and 52.67 faster than FHUQI-Miner. The main reason why CHUQI-Miner outperformed FHUQI-Miner is that these datasets contain many Q-items that are weakly correlated. As a result, a large quantity of HUQIs will be pruned using the bond during the search process because these Q-itemsets are not correlated and occur rarely in the database. For the sparse Foodmart dataset, the runtime of CHUQI-Miner is closer to FHUQI-Miner because in that dataset the Q-items are more correlated. It can be also seen from Fig. 1 that the runtimes of CHUQI-Miner variants with different values of $minbond$ have similar performance. This means that the bond measure does not affect very much the performance of the algorithm on that datasets.

Number of generated patterns on different datasets In a second experiment, the number of patterns, i.e, HUQIs for FHUQI-Miner and CHUQIs for CHUQI-Miner, are recorded and depicted in Fig. 2. It can be seen that for both dense and sparse datasets, CHUQI-Miner can output much less patterns than FHUQI-Miner, especially in case of small values of $minutil$. For example, on Connect with $minutil = 1.5$, FHUQI-Miner returns 3,013 HUQIs while CHUQIs with $minbond = 0.1$ gives only 36 CHUQIs. It can be also observed that a huge reduction is achieved on Chess where CHUQI-Miner found up to 2,120 times less patterns than FHUQI-Miner. On average, for Chess, Mushroom, Connect and Foodmart, the set of CHUQIs is 290.37, 46.19, 31.70 and 17.18 times smaller than the set of HUQIs.

Overall, these results show the usefulness of CHUQI-Miner because it was able to filter out a huge amount of weakly correlated Q-itemsets encountered in real datasets. This allows to not only speed up the discovery of HUQIs but also to only keep the correlated ones. CHUQIs can be viewed as preferable to traditional HUQIs for scenarios such as the cross-promotion of items in online stores.

VI. Conclusion

A major limitation of traditional HUQIM algorithms is that many weakly correlated itemsets are often found. To address this issue, this paper proposed the novel problem of Correlated HUQIM by integrating the bond measure in HUQIM. An efficient algorithm named CHUQI-Miner was presented to find all CHUQIs. The algorithm applies a depth-first search and various pruning strategies. Experiments on four datasets have shown that CHUQI-Miner has an excellent performance compared to the state-of-the-art FHUQI-Miner algorithm and can filter out a large number of spurious patterns.

There are several opportunities for future work such as integrating other correlation measures in HUQIM such as the all-confidence [18], [19] and affinity [20], [21], integrating the concept of taxonomy [23], [24], and mining streaming data [25]–[27].

References

[1] P. Fournier-Viger, J. C.-W. Lin, T. Truong-Chi, and R. Nkambou, "A survey of high utility itemset mining," in *High-Utility Pattern Mining*. Springer, 2019, pp. 1–45.

[2] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. S. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *arXiv preprint arXiv:1805.10511*, 2018.

[3] C. Zhang, M. Han, R. Sun, S. Du, and M. Shen, "A survey of key technologies for high utility patterns mining," *IEEE Access*, vol. 8, pp. 55 798–55 814, 2020.

[4] C. H. Li, C.-W. Wu, and V. S. Tseng, "Efficient vertical mining of high utility quantitative itemsets," in *2014 IEEE International Conference on Granular Computing (GrC)*. IEEE, 2014, pp. 155–160.

[5] C.-H. Li, C.-W. Wu, J. Huang, and V. S. Tseng, "An efficient algorithm for mining high utility quantitative itemsets," in *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2019, pp. 1005–1012.

[6] M. Nouioua, P. Fournier-Viger, J. C.-W. Lin, and W. Gan, "Fhuqi-miner: Fast high utility quantitative itemset mining," *Applied Intelligence*, 2021.

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

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

[9] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, "Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *International symposium on methodologies for intelligent systems*. Springer, 2014, pp. 83–92.

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

[11] A. Y. Peng, Y. S. Koh, and P. Riddle, "mhuiminer: A fast high utility itemset mining algorithm for sparse datasets," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2017, pp. 196–207.

[12] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 55–64.

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

[14] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "Up-growth: an efficient algorithm for high utility itemset mining," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 253–262.

[15] S. Krishnamoorthy, "Hminer: Efficiently mining high utility itemsets," *Expert Systems with Applications*, vol. 90, pp. 168–183, 2017.

[16] Q.-H. Duong, P. Fournier-Viger, H. Ramampiaro, K. Nørvåg, and T.-L. Dam, "Efficient high utility itemset mining using buffered utility-lists," *Applied Intelligence*, vol. 48, no. 7, pp. 1859–1877, 2018.

[17] S. Bouasker and S. Ben Yahia, "Key correlation mining by simultaneous monotone and anti-monotone constraints checking," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 851–856.

[18] 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.

[19] E. R. Omiecinski, "Alternative interest measures for mining associations in databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 1, pp. 57–69, 2003.

[20] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and H.-J. Choi, "A framework for mining interesting high utility patterns with a strong frequency affinity," *Information Sciences*, vol. 181, no. 21, pp. 4878–4894, 2011.

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

[22] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, "Spmf: a java open-source pattern mining library," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3389–3393, 2014.

[23] P. Fournier-Viger, Y. Wang, J. C.-W. Lin, J. M. Luna, and S. Ventura, "Mining cross-level high utility itemsets," in *Proc. 33rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 2020.

[24] M. Nouioua, Y. Wang, P. Fournier-Viger, J. C. Lin, and J. M. Wu, "Tkc: Mining top-k cross-level high utility itemsets," in *Proc. 3rd Workshop on Utility Driven Mining and Learning*. IEEE, 2020, pp. 673–682.

[25] J. Lee, U. Yun, and G. Lee, "Analyzing of incremental high utility pattern mining based on tree structures," *Human-centric Computing and Information Sciences*, vol. 7, no. 1, pp. 1–9, 2017.

[26] Q.-H. Duong, H. Ramampiaro, K. Nørvåg, P. Fournier-Viger, and T.-L. Dam, "High utility drift detection in quantitative data streams," *Knowledge-Based Systems*, vol. 157, pp. 34–51, 2018.

[27] H. Nam, U. Yun, B. Vo, T. Truong, Z.-H. Deng, and E. Yoon, "Efficient approach for damped window-based high utility pattern mining with list structure," *IEEE Access*, vol. 8, pp. 50 958–50 968, 2020.