# Mining Correlated High-Utility Itemsets Using Various Measures

Philippe Fournier-Viger[1], Yimin Zhang[2], Jerry Chun-Wei Lin[2], Duy-Tai Dinh[3], Hoai Bac Le[4]

[1] School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), China
[2] School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), China
[3] Japan Advanced Institute of Science and Technology
[4] Faculty of Information Technology, University of Science, Vietnam
philfv8@yahoo.com, mrzhangym@126.com, jerrylin@ieee.org, taidinh@jaist.ac.jp, lhbac@fit.hcmus.edu.vn

**Abstract.** Discovering high-utility itemsets consists of finding sets of items that yield a high profit in customer transaction databases. An important limitation of traditional high-utility itemset mining is that only the utility measure is used for assessing the interestingness of patterns. This leads to finding several itemsets that have a high profit but contain items that are weakly correlated. To address this issue, this paper proposes to integrate the concept of *correlation* in high-utility itemset mining to find profitable itemsets that are highly correlated, using the all-confidence and bond measures. An efficient algorithm named FCHM (Fast Correlated High-utility itemset Miner) is proposed to efficiently discover correlated high-utility itemsets. Two versions of the algorithm are proposed, named FCHM$_{all\text{-}confidence}$ and FCHM$_{bond}$ based on the all-confidence and bond measures, respectively. An experimental evaluation was done using four real-life benchmark datasets from the high-utility itemset mining litterature: *mushroom*, *retail*, *kosarak* and *foodmart*. Results show that FCHM is efficient and can prune a huge amount of weakly correlated high-utility itemsets.

**Keywords:** high-utility itemset mining, correlation, bond measure, all-confidence measure, correlated high-utility itemsets

## 1 Introduction

*High-Utility Itemset Mining* (HUIM) [3, 12, 13, 15] consists of discovering high-utility itemsets (HUIs) in a transaction database, i.e. groups of items (itemsets) that have a high utility (e.g. yield a high profit). HUIM generalizes the problem of Frequent Itemset Mining (FIM) [1] by considering that items may appear more than once in each transaction, and that weight are associated to items to represent their relative importance, or utility (e.g. unit profit). HUIM has many applications such as website click stream analysis, cross-marketing in retail stores and biomedical applications [12, 15].

An important limitation of traditional HUIM algorithms [3, 9, 11–13, 15, 17] is that only the utility measure is used to assess the interestingness of patterns. This leads to finding many itemsets yielding a high profit but containing items that are weakly correlated. Those itemsets are misleading or useless for taking marketing decisions. For example, consider the transaction database of a retail store. Current algorithms may find that buying a television and a pen is a high-utility itemset, because these two items have globally generated a high profit when sold together. But it would be a mistake to use this pattern to promote televisions to people who buy pens because these two items are rarely sold together. The reason why this pattern may be a HUI despite that there is a very low correlation between pens and televisions, is that televisions are very expensive, and thus almost any items combined with a television may be a HUI. This limitation of current HUIM algorithms is important. In the experimental section of this paper, it will be shown that often less than 1% of the patterns found by traditional HUIM algorithms contains items that are correlated. It is thus a key research problem to design algorithms for discovering HUIs having a high correlation.

This paper addresses this issue by introducing the concept of correlation in high-utility itemset mining, using the all-confidence and bond measures [8, 7]. The contributions of this paper are threefold. First, the concept of *all-confidence and bond* correlation measures used in FIM is combined with the concept of HUIs to define a new type of patterns named *correlated high-utility itemsets* (CHIs), and study their properties. Second, a novel algorithm named FCHM (Fast Correlated high-utility itemset Miner) is proposed to efficiently discover CHIs in transaction databases. Two versions of the FCHM algorithm are presented, named FCHM$_{all\text{-}confidence}$ and FCHM$_{bond}$ based on the all-confidence and bond measures, respectively. The proposed algorithms integrates several strategies to discover correlated high-utility itemsets efficiently. Third, an extensive experimental evaluation is evaluate the efficiency of FCHM. Experiments results show that FCHM can be more than two orders of magnitude faster than the state-of-the-art FHM algorithm for HUIM, and in some case discover five orders of magnitude less patterns by avoiding weakly correlated HUIs.

The rest of this paper is organized as follows. Section 2, 3, 4, and 5 respectively presents preliminaries and related work related to HUIM, the FCHM algorithm, the experimental evaluation and the conclusion.

## 2 Preliminaries and Related Work

The problem of HUIM is defined as follows [3, 13]. Let $I$ be a set of items (symbols). An itemset $X$ is a subset of $I$, that is $X \subseteq I$. A *transaction database* is a set of transactions $D = \{T_1, T_2, ..., T_n\}$ such that for each transaction $T_c$, $T_c \subseteq I$ and $T_c$ has a unique identifier $c$ called its Tid. Each item $i \in I$ is associated with a positive number $p(i)$, called its external utility (e.g. representing the unit profit of this item). For each transaction $T_c$ such that $i \in T_c$, a positive number $q(i, T_c)$ is called the internal utility of $i$ (e.g. representing the purchase quantity of item $i$ in transaction $T_c$).

*Example 1.* Consider the database of Table 1, which will be used as our running example. This database contains five transactions ($T_1, T_2 ... T_5$). For the sake of readability, internal utility values are shown as integer values beside each item, in transactions. Transaction $T_4$ indicates that items $a$, $c$, $e$ and $g$ appear in this transaction with an internal utility of respectively 2, 6, 2 and 5. Table 2 indicates that the external utility of these items are respectively 5, 1, 3 and 1.

Table 1: A transaction database

| TID | Transaction |
|---|---|
| $T_1$ | $(a,1),(b,5),(c,1),(d,3),(e,1),(f,5)$ |
| $T_2$ | $(b,4),(c,3),(d,3),(e,1)$ |
| $T_3$ | $(a,1),(c,1),(d,1)$ |
| $T_4$ | $(a,2),(c,6),(e,2),(g,5)$ |
| $T_5$ | $(b,2),(c,2),(e,1),(g,2)$ |

Table 2: External utility values

| Item | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |
|---|---|---|---|---|---|---|---|
| Unit profit | 5 | 2 | 1 | 2 | 3 | 1 | 1 |

**Definition 1 (Utility of an item).** *The utility of an item $i$ in a transaction $T_c$ is denoted and defined as $u(i, T_c) = p(i) \times q(i, T_c)$.*

*Example 2.* The utility of the item $a$ in transaction $T_1$ is $u(a, T_1) = p(a) \times q(a, T_1) = 5 \times 1 = 5$.

**Definition 2 (Utility of an itemset).** *The utility of an itemset $X$ in a transaction $T_c$ is denoted and defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$. The utility of an itemset $X$ is denoted as $u(X)$ and defined as $u(X) = \sum_{T_c \in g(X)} u(X, T_c)$, where $g(X)$ is the set of transactions containing $X$.*

*Example 3.* The utility of the itemset $\{a, c\}$ in $T_1$ is $u(\{a, c\}, T_1) = u(a, T_1) + u(c, T_1) = 5 + 1 = 6$. The set of transactions containing itemset $\{a, c\}$ is $g(\{a, c\}) = \{T_1, T_3, T_4\}$. The utility of the itemset $\{a, c\}$ is $u(\{a, c\}) = u(\{a, c\}, T_1) + u(\{a, c\}, T_3) + u(\{a, c\}, T_4) = 6 + 6 + 16 = 28$.

**Definition 3 (High-utility itemset mining).** *The problem of high-utility itemset mining is to discover all high-utility itemsets. An itemset $X$ is a* high-utility itemset *if $u(X) \geq minutil$, where $minutil$ is a user-specified minimum utility threshold.*

*Example 4.* If $minutil = 30$, the complete set of HUIs is $\{a, c, e\} : 31$, $\{a, b, c, d, e, f\} : 30$, $\{b, c, d\} : 34$, $\{b, c, d, e\} : 40$, $\{b, c, e\} : 37$, $\{b, d\} : 30$, $\{b, d, e\} : 36$, and $\{b, e\} : 31$, where each HUI is annotated with its utility.

HUIM algorithms such as Two-Phase [13], BAHUI [11], and UPGrowth+ [15] utilize the *Transaction-Weighted Utilization (TWU)* [13,15] measure to prune the search space. They first identify candidate high-utility itemsets by considering their TWUs. Then, in a second phase, they scan the database to calculate the exact utility of all candidates to filter those having a low utility. The TWU measure and its pruning property are defined as follows.

**Definition 4 (Transaction-weighted utilization).** *The utility of a transaction $T$ is defined as $TU(T) = \sum_{i \in T} u(i,T)$. The transaction-weighted utilization (TWU) of an itemset $X$ is defined as the sum of the transaction utility of transactions containing $X$, i.e. $TWU(X) = \sum_{T_c \in g(X)} TU(T_c)$.*

*Example 5.* The utility of a transaction $T_3$ is $TU(T_3) = u(a,T_3) + u(c,T_3) + u(d,T_3) = 5 + 1 + 2 = 8$. The utility of transactions $T_1$, $T_2$, $T_4$ and $T_5$ are respectively $TU(T_1) = 30$, $TU(T_2) = 20$, $TU(T_4) = 27$ and $TU(T_5) = 11$. The TWU of item $a$ is $TWU(a) = TU(T_1) + TU(T_3) + TU(T_4) = 30 + 8 + 27 = 65$. TheTWU of items $b$, $c$, $d$, $e$, $f$ and $g$ are respectively 61, 96, 58, 88, 30 and 38. $TWU(\{c,d\}) = TU(T_1) + TU(T_2) + TU(T_3) = 30 + 20 + 8 = 58$.

The following property of the TWU measure is used by previous algorithms to reduce the search space.

*Property 1 (Pruning search space using the TWU).* Let $X$ be an itemset, if $TWU(X) < minutil$, then $X$ and its supersets are low utility. [13]

A drawback of algorithms working in two phases is that they generate a huge amount of candidates. Recently, an efficient depth-first search algorithm named *HUI-Miner*[12] was proposed, which discovers HUIs using a single phase. A faster version was proposed called FHM [3]. It was shown that FHM can be 6 times faster than HUI-Miner. FHM associates a structure named *utility-list* to each itemset [3,12]. Utility-lists allow calculating the utility of an itemset quickly by making join operations with utility-lists of shorter patterns. The utility-list structure is defined as follows.

**Definition 5 (Utility-list).** *Let $\succ$ be any total order on items from $I$. The* utility-list *$ul(X)$ of an itemset $X$ in a database $D$ is a set of tuples such that there is a tuple $(tid, iutil, rutil)$ for each transaction $T_{tid}$ containing $X$. The iutil element of a tuple is the utility of $X$ in $T_{tid}$. i.e., $u(X, T_{tid})$. The rutil element of a tuple is defined as $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$.*

*Example 6.* Assume that $\succ$ is the alphabetical order. The utility-list of $\{a\}$ is $\{(T_1, 5, 25)$, $(T_3, 5, 3)$, $(T_4, 10, 17)\}$. The utility-list of $\{a, d\}$ is $\{(T_1, 11, 3)$, $(T_3, 7, 0)\}$.

The FHM algorithm scans the database once to create the utility-lists of itemsets containing a single item. Then, the utility-lists of larger itemsets are constructed by joining the utility-lists of smaller itemsets. The join operation for single items is performed as follows. Consider two items $x, y$ such that $x \succ y$, and their utility-lists $ul(\{x\})$ and $ul(\{y\})$. The utility-list of $\{x, y\}$ is obtained by creating a tuple $(ex.tid, ex.iutil + ey.iutil, ey.rutil)$ for each pair of tuples $ex \in ul(\{x\})$ and $ey \in ul(\{y\})$ such that $ex.tid = ey.tid$. The join operation of itemsets containing more than 1 item is done in a similar way (see [12] for details).The utility-list structure allows to calculate the utility-list of itemsets and prune the search space as follows.

*Property 2 (Calculating the utility of an itemset using its utility-list).* The utility of an itemset is the sum of *iutil* values in its utility-list [12].

*Property 3 (Pruning search space using a utility-list).* Let $X$ be an itemset. Let the *extensions* of $X$ be the itemsets that can be obtained by appending an item $y$ to $X$ such that $y \succ i$, $\forall i \in X$. If the sum of *iutil* and *rutil* values in $ul(X)$ is less than *minutil*, $X$ and its extensions are low utility [12].

A key problem of current HUIM algorithms is that they may find a huge amount of itemsets containing weakly correlated items (as it will be shown in the experimental study). Thus, this paper considers the concept of correlation as an additional constraint in high-utility itemset mining. There have been several studies on mining correlated patterns in frequent pattern mining, using various correlation measures such as the any-confidence, all-confidence [14, 5], frequence affinity [2], coherence [6], and bond [5, 8, 7, 14]. To our knowledge, only Ahmed et al. [2] and Lin et al. [10] have used measures of correlation in high-utility itemset mining by employing the measure of *frequency affinity*.

In this paper, we rely on the *all-confidence and bond* measures, which have recently attracted more attention [8, 7, 14]. The all-confidence is defined as follows.

**Definition 6 (Support).** *Let the support of an itemset $X$ in a database $D$ be $support(X) = |\{T | T \in D \wedge X \subseteq T\}|$.*

**Definition 7 (All-confidence).** *According to the all-confidence, an itemset is interesting if **all** association rules that can be produced by partitionning that itemset have a confidence greater than or equal to a minimum all-confidence threshold. The all-confidence of an itemset $X$ is*

$$all\text{-}confidence(X) = \frac{support(X)}{argmax\{support(Y) | \forall Y \subset X \wedge Y \neq \emptyset)\}}$$

*Example 7.* The support of itemset $\{a, d\}$ is $support(\{a, d\}) = |\{T_1, T_3\}| = 2$. The all-confidence of itemset $\{a, d\}$ is calculated as $all\text{-}confidence(\{a, d\}) = \frac{support(\{a,d\})}{argmax\{support(\{a\}), support(\{d\})\}} = \frac{2}{3} = 0.67$.

It's easy to notice that the denominator of the all-confidence is always equal to the support of an item in $X$. Note that the all-confidence of single items is 1. The bond measure for an itemset $X$ is the support of $X$ divided by its disjunctive support. Formally:

**Definition 8 (Disjunctive support).** *Let the disjunctive support of an itemset $X$ in a database $D$ be defined as $dissup(X) = |\{T \in D | X \cap T \neq \emptyset\}|$. In other words, the disjunctive support of $X$ is the number of transactions containing at least one item from $X$.*

**Definition 9 (Bond).** *The bond [5] measure is the support of $X$ divided by its disjunctive support, that is $bond(X) = support(X)/dissup(X)$.*

*Example 8.* The disjunctive support of itemset $\{a, d\}$ is $dissup(\{a, d\}) = |\{T_1, T_2, T_3, T_4\}| = 4$. The bond of itemset $\{a, d\}$ is $bond(\{a, d\}) = \frac{2}{4} = 0.5$.

It was shown that for any itemset $X$, all-confidence($X$) $\geq$ bond($X$)[5].

*Property 4 (Anti-monotonicity of the all-confidence and bond measures).* For any two itemsets $X$ and $Y$ such that $X \subseteq Y$, $bond(X) \geq bond(Y)$ and $all\text{-}confidence(X) \geq all\text{-}confidence(Y)$ [7].

Based on the above definitions, we define the problem of mining Correlated High-utility Itemsets (CHIs) using the all-confidence or bond measures. **The problem of correlated high-utility itemset mining** is to discover all correlated high-utility itemsets. An itemset $X$ is a **correlated high-utility itemset** if it is a high-utility itemset and its bond or all-confidence is no less than a user-defined threshold. For example, if $minutil = 30$ and $minbond = 0.6$, CHIs are: $\{b, e\}$ and $\{b, c, e\}$, having a bond of 0.75 and 0.6, respectively. And if $minutil = 30$ and $minall\text{-}confidence = 0.6$, CHIs are $\{b, d\}$, $\{b, e\}$ and $\{b, c, e\}$, having an all-confidence of 0.66, 0.75 and 0.6, respectively.

## 3 The FCHM Algorithm

This section presents the proposed FCHM algorithm. The main procedure is described, which is based on the FHM [3] algorithm, and discovers all HUIs. Then, the procedure is modified to find only CHIs.

The main procedure of FCHM (Algorithm 1) takes as input a transaction database, the *minutil* threshold and a *min_measure* threshold. The algorithm first scans the database to calculate the TWU of each item. Then, the algorithm identifies the set $I^*$ of all items having a TWU no less than *minutil* (other items are ignored since they cannot be part of a high-utility itemset by Property 3). The TWU values of items are then used to establish a total order $\succ$ on items, which is the order of ascending TWU values (as in [3]). A database scan is then performed. During this database scan, items in transactions are reordered according to the total order $\succ$, the utility-list of each item $i \in I^*$ is built and a structure named EUCS (Estimated Utility Co-Occurrence Structure) is built [3]. This latter structure is defined as a set of triples of the form $(a, b, c) \in I^* \times I^* \times \mathbb{R}$. A triple (a,b,c) indicates that TWU($\{a, b\}$) $= c$. The EUCS can be implemented as a triangular matrix that stores these triples for all pairs of items. For example, the EUCS for the running example is shown in Fig. 1. The EUCS is useful as it stores the TWU of all pairs of items, an information that will be used for pruning the search space. For instance, the top-left cell indicates that $TWU(\{a,b\}) = 30$. Building the EUCS is very fast (it is performed with a single database scan) and occupies a small amount of memory, bounded by $|I^*| \times |I^*|$. The reader is referred to the paper about FHM [3] for more details about the EUCS. After the construction of the EUCS, the depth-first search exploration of itemsets starts by calling the recursive procedure *Search* with the empty itemset $\emptyset$, the set of single items $I^*$, *minutil*, the EUCS and *min_measure*.

| **Item** | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| b | 30 | | | | | |
| c | 65 | 61 | | | | |
| d | 38 | 50 | 58 | | | |
| e | 57 | 61 | 88 | 50 | | |
| f | 30 | 30 | 30 | 30 | 30 | |
| g | 27 | 11 | 38 | 0 | 38 | 0 |

Fig. 1: The EUCS

| **Item** | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| b | 1 | | | | | |
| c | 3 | 3 | | | | |
| d | 2 | 2 | 3 | | | |
| e | 2 | 3 | 4 | 2 | | |
| f | 1 | 1 | 1 | 1 | 1 | |
| g | 1 | 1 | 2 | 0 | 2 | 0 |

Fig. 2: The Support Matrix

**Algorithm 1:** The FCHM algorithm

```
input  : $D$: a transaction database, $minutil$: a user-specified threshold,
         $min\_measure$: a user-specified threshold for the $all\text{-}confidence$ or
         $bond$ measure
output: the set of correlated high-utility itemsets
1 Scan $D$ to calculate the TWU of single items;
2 $I^* \leftarrow$ each item $i$ such that TWU($i$) $\geq minutil$;
3 Let $\succ$ be the order of TWU ascending values on $I^*$;
4 Scan $D$ to built the utility-list of each item $i \in I^*$ and build the $EUCS$;
5 Output each item $i \in I^*$ such that SUM($\{i\}.utilitylist.iutils$) $\geq minutil$;
6 Search ($\emptyset$, $I^*$, $minutil$, $EUCS$, $min\_measure$);
```

The *Search* procedure (Algorithm 2) takes as input (1) an itemset $P$, (2) extensions of $P$ having the form $Pz$ meaning that $Pz$ was previously obtained by appending an item $z$ to $P$, (3) *minutil*, (4) the EUCS and (5) *min_measure*. The search procedure operates as follows. For each extension $Px$ of $P$, if the sum of the *iutil* values of the utility-list of $Px$ is no less than *minutil*, then $Px$ is a high-utility itemset and it is output (cf. Property 4). Then, if the sum of *iutil* and *rutil* values in the utility-list of $Px$ are no less than *minutil*, it means that extensions of $Px$ should be explored. This is performed by merging $Px$ with all extensions $Py$ of $P$ such that $y \succ x$ to form extensions of the form $Pxy$ containing $|Px|+1$ items. The utility-list of $Pxy$ is then constructed as in FHM by calling the *Construct* procedure to join the utility-lists of $P$, $Px$ and $Py$. This latter procedure is the same as in FHM [3] and is thus not detailed here. Note that if the value of $Pxy$ for the correlation measure is less than $min\_measure$, its *utility-list* is not added to the set of extensions to be considered by the depth-first search. This can be done because the bond and all-confidence measures are anti-monotonic. Thus, any extensions of a non correlated itemsets can be safely eliminated from the search space. Then, a recursive call to the $Search$ procedure with $Pxy$ is done to calculate its utility and explore its extension(s). Since the $Search$ procedure starts from single items, it recursively explores the search space of itemsets by appending single items and it only prunes the search space based on Properties 3 and 4, it can be easily seen that this procedure is correct and complete to discover all high-utility itemsets. Above, how the bond and all-confidence are calculated has not been explained. Calculating these measures is done differently and requires specific optimizations. Thus, two versions of FCHM are developed called FCHM$_{all\text{-}confidence}$ and FCHM$_{bond}$, and are described next.

**Algorithm 2:** The $Search$ procedure

```
input : P: an itemset, ExtensionsOfP: a set of extensions of P, minutil: a
        user-specified threshold, EUCS: the EUCS structure, min_measure:
        a user-specified threshold for the all_confidence or bond measure
output: the set of high-utility itemsets
1  foreach itemset $Px \in ExtensionsOfP$ do
2      if $SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.rutils) \geq minutil$ then
3          $ExtensionsOfPx \leftarrow \emptyset$;
4          foreach itemset $Py \in ExtensionsOfP$ such that $y \succ x$ do
5              if $\exists(x, y, c) \in EUCS$ such that $c \geq minutil$ then
6                  $Pxy \leftarrow Px \cup Py$;
7                  $Pxy.utilitylist \leftarrow$ Construct $(P, Px, Py)$;
8                  if $Pxy.measure \geq min\_measure$ then
9                      $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
10                     if $SUM(Pxy.utilitylist.iutils) \geq minutil$ then output $Px$;
11                 end
12             end
13         end
14         Search ($Px$, $ExtensionsOfPx$, $minutil$);
15     end
16 end
```

## 3.1 The FCHM$_{all\text{-}confidence}$ Algorithm

Calculating the all-confidence of an itemset $X$ requires to divide its support by the maximum support of its subsets. As mentionned, the maximum support of the subsets of $X$ is always equal to the support of an item in $X$. Thus, to calculate the all-confidence of $X$, it is only necessary to find the support of $X$ and its items. The support of $X$ is easily obtained as it is the size of its

**Algorithm 3:** The Construct procedure

```
input  : $P$: an itemset, $Px$: the extension of $P$ with an item $x$, $Py$: the
         extension of $P$ with an item $y$
output: the utility-list of $Pxy$
1  $UtilityListOfPxy \leftarrow \emptyset$;
2  foreach  $tuple\ ex \in Px.utilitylist$ do
3      if $\exists ey \in Py.utilitylist\ and\ ex.tid = exy.tid$ then
4          if $P.utilitylist \neq \emptyset$ then
5              Search element $e \in P.utilitylist$ such that $e.tid = ex.tid.$;
6              $exy \leftarrow (ex.tid, ex.iutil + ey.iutil - e.iutil, ey.rutil)$;
7          end
8          else  $exy \leftarrow (ex.tid, ex.iutil + ey.iutil, ey.rutil)$;
9          $UtilityListOfPxy \leftarrow UtilityListOfPxy \cup \{exy\}$;
10     end
11 end
12 return $UtilityListPxy$;
```

utility list, i.e. $|utility\text{-}list(X)|$. The support of single items is obtained from their utility-lists, constructed at line 4 of Algorithm 1. Using this information, it is straightforward to find the maximum support of the subsets of an itemset $X$, and calculate its all-confidence. The all-confidence of an itemset $X$ is thus calculated as $|utility\text{-}list(X)|/argmax\{|utility\text{-}list(i)|\}\forall i \in X$. For example, $all\text{-}confidence(ab) = |utility\text{-}list(ab)|/argmax\{utility\text{-}list(a), utility\text{-}list(b)\}$.

Finding $maxSubset(X)$, the maximum support of items in an itemset $X$ requires $|X| - 1$ comparisons. To reduce the number of comparisons, the maximum support of subsets of each itemset can be stored in its utility-list. Then, when constructing the utility-list of an itemset $Pxy$, the maximum support of its subsets can be obtained by doing one comparison as $maxSubset(Pxy) = argmax\{maxSubset(Px), maxSubset(Py)\}$. To further improve the performance of FCHM$_{all\text{-}confidence}$, the next paragraphs describe three strategies to more efficiently discover CHIs.

**Strategy 1. Directly Outputting Single items (DOS).** HUIs containing a single item are directly output since their all-confidence is 1.

**Strategy 2. Pruning Supersets of Non correlated itemsets (PSN).** Because the all-confidence measure is anti-monotonic (Property 4), if the all-confidence of an itemset $Pxy$ is less than $min\_measure$, any extensions of $Pxy$ should not be explored.

**Strategy 3. Pruning using the Upper-Bound (PUB) version 1.** The third strategy is introduced to avoid constructing the utility-list of an itemset $Pxy$, and prune all its extensions. During the second database scan, a novel structure named *Support Matrix* is created to store the support of all itemsets containing two items from $I^*$. The design of this structure is similar to the EUCS. The Support Matrix is formally defined as follows. The Support Matrix is a set of triples of the form $(a, b, c) \in I^* \times I^* \times \mathbb{R}$. A triple (a,b,c) indicates that $support(\{a, b\}) = c$. Note that it only stores tuples of the form

$(a, b, c)$ such that $c \neq 0$. For example, the Support Matrix for the running example is shown in Fig. 2. For $FCHM_{all\text{-}confidence}$, Line 5 of Algorithm 1 is modified to add the condition that the utility-list of $Pxy$ is only built if $\frac{Min\{support(Px), support(Py), support(xy)\}}{Max\{maxSubset(Px), maxSubset(Py)\}} \geq min\_measure$. Because the left end of the inequality is an upper-bound on the *all-confidence*$(Pxy)$, this pruning condition preserves the correctness and completeness of FCHM.

## 3.2 The FCHM$_{bond}$ Algorithm

Calculating the bond can be done using a naive approach by scanning the database for each HUI to calculate its support and disjunctive support, and then its bond. However, this is inefficient. A better approach, used in FCHM, is the following. A structure called *disjunctive bit vector* [8] is appended to each utility-list to efficiently calculate the disjunctive support of any itemset.

The *disjunctive bit vector* of an itemset $X$ in a database $D$ is denoted as $bv(X)$. It contains $|D|$ bits, where the $j$-th bit is set to 1 if $\exists i \in X$ such that $i \in T_i$, and is otherwise set to 0. For example, $bv(a) = 10110$, and $bv(b) = 11001$. The disjunctive bit vector of each item is created during the first database scan (Line 4 of Algorithm 1). Then, the disjunctive bit vector of any larger itemset $Pxy$ explored by the search procedure is obtained very efficiently by performing the logical OR of the bit vectors of $Px$ and $Py$. This is added before Line 1 of the utility-list construction procedure (Algorithm 3). For example, $bv(\{a, b\}) = 10110$ OR $11001 = 11111$, and thus $dissup(\{a, b\} = |11111| = 5$. An interesting observation is that the cardinality $|bv(Pxy)|$ of the disjunctive bit vector of an itemset $Pxy$ is equal to its disjunctive support, and that the cardinality $|ul(Pxy)|$ of its utility list is equal to its conjunctive support. Thus, the bond of an itemset $Pxy$ can be obtained using its utility-list and disjunctive bit vector, as follows. Let there be an itemset $X$. The bond of $X$ can be calculated as $|ul(X)|/|bv(X)|$, where $|ul(X)|$ is the number of elements in the utility-list of $X$.

Using the above property, it is possible to calculate the bond of any itemset generated by the search procedure after its utility-list has been constructed. The resulting $FCHM_{bond}$ algorithm is correct and complete for mining CHIs. To improve the performance of FCHM, the next paragraphs describe additional strategies to more efficiently discover CHIs. In $FCHM_{bond}$, Strategies 1 and 2 of $FCHM_{all\text{-}confidence}$ are applied, Strategy 3 is adapted, and two new strategies named Strategies 4,5 are proposed.

**Strategy 3. Pruning Using the upper-Bound (PUB) version 2.** This strategy is similar to the PUB version 1 strategy. It utilizes the Support Matrix. For $FCHM_{bond}$, Line 5 of Algorithm 1 is modified to add the condition that the utility-list of $Pxy$ should only be built if $\frac{Max\{support(Px), support(Py), support(xy)\}}{Min\{dissup(Px), dissup(Py), dissup(xy)\}}$ $\geq min\_measure$, where $dissup(xy) = support(x) + support(y) - support(xy)$ and $support(xy)$ is found in the Support Matrix. It can be easily seen that this pruning condition preserves the correctness and completeness of FCHM because the left-hand side of the inequality is an upper-bound on $bond(Pxy)$.

**Strategy 4. Abandoning Utility-List construction early (AUL).** The fourth strategy introduced in FCHM is to stop constructing the utility-list of an itemset if some specific condition is met, indicating that the itemset may not be a CHI. The strategy is based on the following novel observation:

*Property 5 (Required conjunctive support for an extension Pxy).* An itemset $Pxy$ is correlated only if its support is no less than the value $lowerBound(Pxy) = \lceil |dissup(Pxy)| * min\_measure) \rceil$.

This property is directly derived from the definition of the bond measure, and proof is therefore omitted. Now, consider the construction of the utility-list of an itemset $Pxy$ by the Construct procedure (Algorithm 3). As mentioned, a first modification to this procedure is to create the disjunctive bit vector of $Pxy$ by performing the OR operation with the bit vectors of $Px$ and $Py$ before Line 1. This allows obtaining the value $support(Pxy) = |bv(Pxy)|$. A second modification is to create a variable $maxSupport$ that is initialized to the conjunctive support of $Px$ ($support(Px) = |bv(Px)|$, before Line 1. The third modification is to the following lines, where the utility-list of $Pxy$ is constructed by checking if each tuple in the utility-lists of $Px$ appears in the utility-list of $Py$ (Line 3). For each tuple not appearing in $Py$, the variable $maxSupport$ is decremented by 1. If $maxSupport$ is smaller than $lowerBound(Pxy)$, the construction of the utility-list of $Pxy$ can be stopped because the support of $Pxy$ will not be higher than $lowerBound(Pxy)$. Thus $Pxy$ is not a CHI by Property 5, and its extensions can also be ignored by Property 4. As it will be shown in the experiment, this strategy is very effective, and decrease execution time and memory usage by stopping utility-list construction early. A similar pruning strategy based on the utility of itemset $Pxy$ instead of the bond measure is also integrated in FCHM. This strategy is called the LA-prune strategy and was introduced in HUP-Miner [9], and it is thus not described here.

**Strategy 5. Pruning Utility-List using upper-bound(PUL).** The $FCHM_{bond}$ algorithm can create a utility-list and a bit vector for each itemset $Pxy$. The bit vector is either created by scanning the database (for single items) or performing the OR operation with two bit vectors. The bit vector of an itemset $Pxy$ allows to obtain $dissup(Pxy)$. If $\frac{Max\{support(Px), support(Py), support(xy)\}}{dissup(Pxy)} < min\_measure$, then the utility-list of $Pxy$ is not constructed because the left-hand side is an upper-bound on $bond(Pxy)$.

## 4 Experimental Study

Experiments were performed to assess the performance of FCHM on a computer having an Intel Xeon E3-1270 v5 processor running Windows 10, and 16 GB of free RAM. The performance of FCHM$_{all\text{-}confidence}$ and FCHM$_{bond}$ was compared with the state-of-the-art FHM algorithm for mining HUIs. Four real-life datasets commonly used in the HUIM litterature were used: *mushroom*, *retail*, *kosarak* and *foodmart*. They represent the main types of data typically encountered in real-life scenarios (dense, sparse, and long transactions). Let $|I|$, $|D|$ and $A$ represents the number of transactions, distinct items and average transaction length. The characteristics of the datasets are presented in Table 3.

Table 3: Characteristics of the datasets.

| **Dataset** | $D$ | $I$ | $A$ | Type |
|---|---|---|---|---|
| mushroom | 88,162 | 16,470 | 23 | dense |
| kosarak | 990,002 | 41,270 | 8.1 | sparse with long transactions |
| retail | 88,162 | 16,470 | 10.3 | sparse with many items |
| foodmart | 21,556 | 1,559 | 4.4 | sparse with short transactions |

The source code of algorithms and datasets can be downloaded as part of the SPMF open source data mining library [4] at `http://www.philippe-fournier-viger.com/spmf/`. Memory measurements were done using the Java API. FCHM was run with three different $min\_measure$ threshold values (0.1, 0.5 and 0.9). Thereafter, FCHM-bond-$x$ denotes FCHM$_{bond}$ with $min\_measure = x$ and FCHM-all-$x$ denotes FCHM$_{all\text{-}confidence}$ with $min\_measure = x$. Algorithms were run on each dataset, while decreasing the $minutil$ threshold until they became too long to execute, ran out of memory or a clear trend was observed. Fig. 3 compares the execution times of FCHM and FHM. Fig. 4, compares the number of CHIs and HUIs, respectively generated by these algorithms.

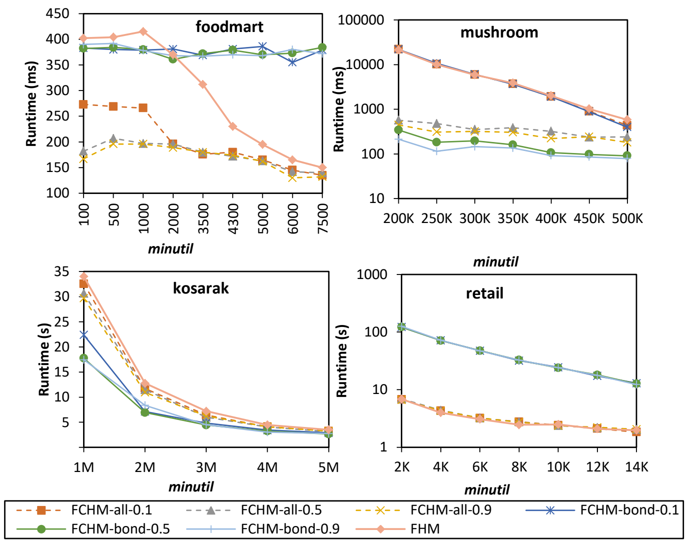

Fig. 3: Execution times

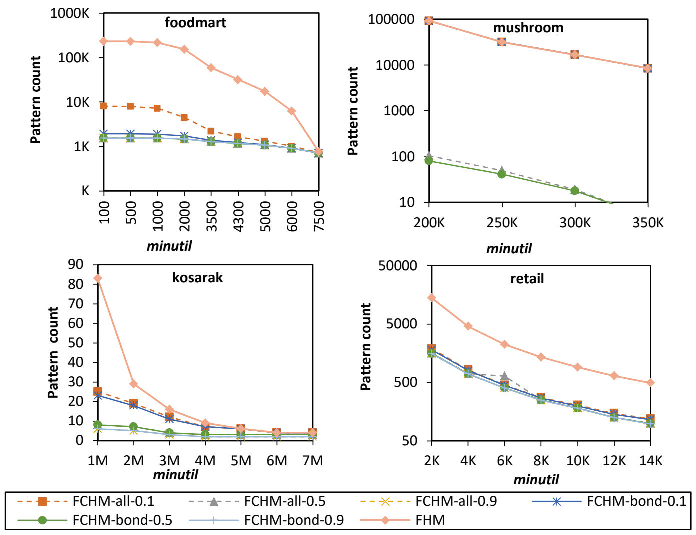

Fig. 4: Number of patterns found

It can be observed that in some cases, mining CHIs can be much faster than mining HUIs, while in other cases, the runtimes are similar. When *min_measure* is set to a high value, FCHM can prune a large part of the search space using its designed pruning strategies based on correlation measures. For datasets containing a huge amount of weakly correlated HUIs, this leads to a considerable performance improvement. For example, for the lowest *minutil* value and $min\_measure = 0.5$ on *mushroom* and *kosarak*, $FCHM_{bond}$ is respectively 100 and 2 times faster than FHM. For such datasets, when *min_measure* is increased, the gap between the runtime of FHM and FCHM increases. For the *mushroom* and *kosarak* datasets, $FCHM_{bond}$ is the fastest as the bond allows pruning a larger part of the search space compared to the all-confidence. For the other datasets, $FCHM_{all}$ is the fastest, since its pruning strategies are more efficient than those of $FCHM_{bond}$ (it does not use bit vectors).

A second observation is that the number of CHIs can be much less than the number of HUIs, even when using low *min_measure* values (see Fig. 4). For example, on *mushroom*, 92,656 HUIs are found for $minutil = 2,000,000$. But only 81 HUIs are CHIs for $min\_measure = 0.5$ and only 5 for $min\_measure = 0.9$ (about 1 CHIs for 18,500 HUIs) using the bond measure, and 105 and 6 CHIs, using the all-confidence measure. For the *foodmart*, *kosarak* and *retail* datasets, a reduction of up to 150 times, 14 times and 9 times is achieved. These overall results show that the proposed FCHM algorithm can filter a huge amount of weakly correlated itemsets encountered in real datasets, and can run faster.

Memory consumption was also compared. The peak memory usage of algorithms was measured for each dataset. The results are shown in Table 4. Several observations can be made. First, when the *min_measure* is decreased, the memory consumption of FCHM-all and FCHM-bond increases. Second, the FCHM-all algorithm generally consumes less memory than FCHM-bond. The reason is that FCHM-bond stores a disjunctive bit vector for each candidate itemset. Third, the FCHM-all algorithm generally consumes less memory than the FHM algorithm. For the foodmart, mushroom and kosarak dataset, FCHM-all is the most memory efficient, while it is the second one for the retail dataset. The reason why FCHM-all achieves good results is that it can eliminate many itemsets that are not correlated, which reduces its memory consumption.

Table 4: Comparison of peak memory usage (MB).

| Algorithm | foodmart | mushroom | kosarak | retail |
|---|---|---|---|---|
| FCHM-all-0.1 | 29 | 423 | 1383 | 804 |
| FCHM-all-0.5 | **26** | 265 | 1370 | 781 |
| FCHM-all-0.9 | **26** | 198 | **1366** | 748 |
| FCHM-bond-0.1 | 266 | 270 | 1849 | 3451 |
| FCHM-bond-0.5 | 266 | 138 | 1901 | 3374 |
| FCHM-bond-0.9 | 266 | **110** | 1575 | 3386 |
| FHM | 67 | 425 | 1466 | **621** |

## 5 Conclusion

This paper proposed the FCHM (Fast Correlated high-utility itemset Miner) algorithm to efficiently mine correlated high-utility itemsets. Two versions of the FCHM algorithm are presented. The first version, named FCHM$_{all\text{-}confidence}$ relies on the all-confidence measure. It includes three strategies to reduce the search space named DOS (Directly Outputting Single items), PSN (Pruning Supersets of Non correlated itemsets) and PUB (Pruning using the Upper-Bound). The second version of FCHM, named FCHM$_{bond}$ relies on the bond measure. It applies the DOS and PSN strategies, as well as three additional strategies named PUB (Pruning using the Upper-Bound version 2), AUL (Abandoning Utility-List construction early) and PUL (Pruning Utility-List using upper-bound).

An extensive experimental study has shown that FCHM is up to two orders of magnitude faster than FHM, and can discover more than four orders of magnitude less patterns by only mining correlated HUIs.

This is the first paper for mining high-utility itemsets with the bond and all-confidence measures. For future work, it would be interesting to design more efficient algorithms in terms of runtime and memory consumption. In particular, using disjunctive bit vectors can consume quite a lot of memory in some cases. Designing an alternative method or structure for computing the disjunctive support would thus be beneficial. Moreover, other correlation measures should be considered. We also plan to integrate the concept of correlation to mine other types of patterns such as correlated sequential rules [16].

**Acknowledgement.** This research was supported by a Youth 1000 talent grant from the National Natural Science Foundation of China (NSFC).

## References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. Int. Conf. Very Large Databases, pp. 487–499, (1994)
2. Ahmed, C. F., Tanbeer, S. K., Jeong, B. S., Choi, H. J.: A framework for mining interesting high utility patterns with a strong frequency affinity. Information Sciences. 181(21), pp. 4878–4894 (2011)
3. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V. S.: FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proc. 21st Int. Symp. on Methodologies for Intell. Syst., pp. 83–92 (2014)
4. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu., C., Tseng, V. S.: SPMF: a Java Open-Source Pattern Mining Library. Journal of Machine Learning Research (JMLR), 15, pp. 3389-3393 (2014)
5. Omiecinski, E.R.: Alternative interest measures for mining associations in databases. IEEE Transactions on Knowledge and Data Engineering 15(1), pp. 57-69 (2003)
6. Barsky, M., Kim, S., Weninger, T., Han, J.: Mining flipping correlations from large datasets with taxonomies. In: Proc. 38th Int. Conf. on Very Large Databases, pp. 370–381 (2012)
7. Ben Younes, N., Hamrouni, T., Ben Yahia, S.: Bridging conjunctive and disjunctive search spaces for mining a new concise and exact representation of correlated patterns. In: Proc. 13th Int. Conf. Discovery Science, pp. 189–204 (2010)
8. Bouasker, S., Ben Yahia, S.: Key correlation mining by simultaneous monotone and anti-monotone constraints checking. In: Proc. 30th Symp. on Applied Computing, pp. 851-856, 2015.
9. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Systems with Applications, 42(5), 2371-2381 (2015)
10. Lin, J. C.-W., Gan, W., Fournier-Viger, P., Hong, T.-P. (2016). Mining Discriminative High Utility Patterns. Proc. 8th Asian Conference on Intelligent Information and Database Systems, Springer, 10 pages.
11. Song, W., Liu, Y., Li, J.: BAHUI: Fast and memory efficient mining of high utility itemsets based on bitmap. Int. J. Data Warehousing and Mining. 10(1), 1–15 (2014)
12. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proc. 22nd ACM Int. Conf. Info. and Know. Management, pp. 55–64 (2012)
13. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 689–695 (2005)
14. Soulet, A., Raissi, C., Plantevit, M., Cremilleux, B.: Mining dominant patterns in the sky. In: Proc. 11th IEEE Int. Conf. on Data Mining, pp. 655-664 (2011)
15. Tseng, V. S., Shie, B.-E., Wu, C.-W., Yu., P. S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25(8), 1772–1786 (2013)
16. Zida, S., Fournier-Viger, P., Wu, C.-W., Lin, J. C. W., Tseng, V.S.: Efficient mining of high utility sequential rules. in: Proc. 11th Int. Conf. Machine Learning and Data Mining, pp. 1–15 (2015)
17. Zida, S., Fournier-Viger, P., Lin, J. C.-W., Wu, C.-W., Tseng, V.S.: EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining. Proc. 14th Mexican Int. Conf. on Artificial Intelligence, pp. 530–546.