Mining Correlated High-Utility Itemsets using the Bond Measure

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

[1] School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, China
[2] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, China
[3] Faculty of Information Technology, Ho Chi Minh city Industry and Trade College
[4] Faculty of Information Technology, University of Science, Vietnam
philippe.fournier-viger@umoncton.ca, jerrylin@ieee.org, duytai@fit-hitu.edu.vn, lhbac@fit.hcmus.edu.vn

Abstract. High-utility itemset mining is the task of finding the sets of items that yield a high utility (e.g. profit) in quantitative transaction databases. An important limitation of previous work on high-utility itemset mining is that utility is generally used as the sole criterion for assessing the interestingness of patterns. This leads to finding many 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 bond measure. An efficient algorithm named FCHM (Fast Correlated high-utility itemset Miner) is proposed to efficiently discover correlated high-utility itemsets. Experimental results show that FCHM is highly-efficient and can prune a huge amount of weakly correlated HUIs.

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

1 Introduction

High-Utility Itemset Mining (HUIM) [3, 12, 13, 15] is an important data mining task. Given a customer transaction database, HUIM consists of discovering high-utility itemsets. i.e. groups of items (itemsets) that have a high utility (e.g. yield a high profit). HUIM is a generalization of the problem of Frequent Itemset Mining (FIM) [1], where items can appear more than once in each transaction and where each item has a weight (e.g. unit profit). HUIM has a wide range of applications such as website click stream analysis, cross-marketing in retail stores and biomedical applications [12, 15]. Although, numerous algorithms have been proposed for HUIM [3, 8, 11–13, 15], an important limitation of current algorithms is that utility is generally used as sole criterion for assessing 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 50 inch plasma 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 plasma television to people who buy pens because if one looks closely 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 plasma televisions, is that plasma televisions are very expensive, and thus almost any items combined with a plasma television may be a HUI. This limitation of current HUIM algorithm 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 strongly 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 bond measure [7, 6]. The contributions of this paper are threefold. First, we combine the concept of *bond* correlation used in FIM with the concept of HUIs to define a new type of patterns named *correlated high-utility itemsets* (CHIs), and study their properties. Second, we present a novel algorithm named FCHM (Fast Correlated high-utility itemset Miner) to efficiently discover the CHIs in transaction databases. The proposed algorithm integrates several strategies to discover correlated high-utility itemsets efficiently, namely (1) Directly Outputting Single items (DOS), (2) Pruning Supersets of Non correlated itemsets (PSN), (3) Pruning using the Bond Matrix (PBM), and (4) Abandoning Utility-List construction early (AUL). Third, an extensive experimental evaluation is carried to compare the efficiency of FCHM with the state-of-the-art FHM algorithm for HUIM. Experiments results show that FCHM can be more than two orders of magnitude faster than FHM, and in some case discover more than five orders of magnitude less patterns by only mining correlated HUIs, and 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

This section introduces preliminary definitions related to high-utility itemset mining and discussed related work. Let I be a set of items (symbols). 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. represent-

ing the purchase quantity of item i in transaction T_c). For example, 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. For example, 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

The utility of an item i in a transaction T_c is denoted as $u(i, T_c)$ and defined as $p(i) \times q(i, T_c)$. The utility of an itemset X (a group of items $X \subseteq I$) in a transaction T_c is denoted as $u(X, T_c)$ 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. For example, the utility of item a in T_4 is $u(a, T_4) = 5 \times 2 = 10$. The utility of the itemset $\{a, c\}$ in T_4 is $u(\{a, c\}, T_4) = u(a, T_4) + u(c, T_4) = 5 \times 2 + 1 \times 6 = 16$. The utility of the itemset $\{a, c\}$ is $u(\{a, c\}) = u(a) + u(c) = u(a, T_1) + u(a, T_3) + u(a, T_4) + u(c, T_1) + u(c, T_3) + u(c, T_4) = 5 + 5 + 10 + 1 + 1 + 6 = 28$. The *problem of high-utility itemset mining* is to discover all high-utility itemsets. An itemset X is a *high-utility itemset* if its utility $u(X)$ is no less than a user-specified minimum utility threshold *minutil* given by the user. Otherwise, X is a *low-utility itemset*. For instance, 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 a measure called the *Transaction-Weighted Utilization (TWU)* [13, 15] to prune the search space. This measure has the useful properties of being an upper-bound on the utility of itemsets and being anti-monotonic (unlike the utility measure). Thus, an itemset always has a TWU no less than the TWU of its supersets. The aforementioned algorithms operate in two phases. In the first phase, they identify itemsets that may be high-utility itemsets by considering their TWUs. Then, in the second phase, they scan the database to calculate the exact utility of all candidates found in the first phase those having a low utility. The TWU measure and its pruning property are defined as follows.

Definition 1 (Transaction weighted utilization). The *transaction utility* (TU) of a transaction T_c is the sum of the utility of all the items in T_c. i.e.

$TU(T_c) = \sum_{x \in T_c} u(x, T_c)$. 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 1. The TUs of T_1, T_2, T_3, T_4 and T_5 are respectively 30, 20, 8, 27 and 11. The TWU of single items a, b, c, d, e, f and g are respectively 65, 61, 96, 58, 88, 30 and 38. $TWU(\{c, d\}) = TU(T_1) + TU(T_2) + TU(T_3) = 30 + 20 + 8 = 58$.

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 may generate a huge amount of candidates. To address this issue, a more efficient depth-first search algorithm named *HUI-Miner*[12] was proposed, which discovers HUIs directly using a single phase. A faster version of HUI-Miner was then proposed called FHM [3], by introducing a technique called co-occurrence pruning. It was shown that FHM can be up to 6 times faster than HUI-Miner. The FHM algorithm 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.

Definition 2 (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 2. 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 $\{d\}$ is $\{(T_1, 6, 3), (T_2, 6, 3), (T_3, 2, 0)\}$. 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 pairs of tuples $ex \in ul(\{x\})$ and $ey \in ul(\{y\})$ such that $ex.tid = ey.tid$. The join operation for two itemsets $P \cup \{x\}$ and $P \cup \{y\}$ such that $x \succ y$ is performed as follows. Let $ul(P)$, $ul(\{x\})$ and $ul(\{y\})$ be the utility-lists of P, $\{x\}$ and $\{y\}$. The utility-list of $P \cup \{x, y\}$ is obtained by creating a tuple $(ex.tid, ex.iutil + ey.iutil - ep.iutil, ey.rutil)$ for each set of tuples $ex \in ul(\{x\})$, $ey \in ul(\{y\})$, $ep \in ul(P)$ such that $ex.tid = ey.tid = ep.tid$. 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].

Although much work has been done on developing efficient HUIM algorithms, a key problem of current HUIM algorithms is that they may find a huge amount of itemsets containing items that are weakly correlated (as it will be shown in the experimental study). A promising solution to this problem, which is explored in this paper is to integrate the concept of correlation as an additional constraint in high-utility itemset mining. There have been several works on mining correlated patterns in the field of frequent pattern mining, using various correlation measures such as the all-confidence [14], frequence affinity [2], coherence [5], and bond [7, 6, 14]. To our knowledge, only the works of Ahmed et al. [2] and Lin et al. [10] have been applied in high-utility itemset mining by employing the measure of *frequency affinity*. In this paper, we instead rely on the *bond* measure, which has recently attracted more attention [7, 6, 14].

The concept of correlated itemsets based on the bond measure is defined as follows [7]: The *conjunctive support* of an itemset X in a database D is denoted as $consup(X)$ and defined as $|g(X)|$, where $|g(X)|$ is the number of transactions in $g(X)$). The *disjunctive support* of an itemset X in a database D is denoted as $dissup(X)$ and defined as $|\{T_c \in D | X \cap T_c \neq \emptyset\}|$. The *bond* of itemset X is defined as $bond(X) = consup(X)/dissup(X)$. An itemset X is said to be correlated if $bond(X) \geq minbond$, for a given user-specified *minbond* threshold ($0 \leq minbond \leq 1$). The bond measure is anti-monotonic.

Property 4 (Anti-monotonicity of the bond measure). Let X and Y be two itemsets such that $X \subseteq Y$. It follows that $bond(X) \geq bond(Y)$ [6].

Based on the above definitions, we define the problem of mining Correlated High-utility Itemsets (CHIs) using the bond measure.

Definition 3 (Correlated high-utility itemset mining). 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 $bond(X)$ is no less than a user-specified minimum bond threshold *minbond*, specified by the user.

For example, if $minutil = 30$ and $minbond = 0.5$, the complete set of CHIs is: $\{b, d\}$,{b,e}, and $\{b, c, e\}$, which respectively have a bond of 0.50, 0.75, and 0.60. Thus, three itemsets are CHIs among the seven HUIs for the running example.

3 The FCHM algorithm

This section presents the proposed FCHM algorithm. It first describes the main procedure, which is based on the FHM [3] algorithm, and discovers all HUIs. Then, it explains how this procedure is modified to find only CHIs.

The main procedure of FCHM (Algorithm 1) takes a transaction database with utility values as input, and the *minutil* 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 suggested in [12]). 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 very useful as it stores the TWU of all pairs of items, an information that will be later 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 [**?**] for more details about the construction of this structure and how it can be implemented using hashmaps. 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* and the EUCS structure.

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 Bond Matrix

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* and (4) the EUCS. 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 [12] and is thus not detailed here. Then, a recursive call to

Algorithm 1: The FCHM algorithm

input : D: a transaction database, *minutil*: a user-specified threshold
output: the set of high-utility itemsets

Scan D to calculate the TWU of single items;
$I^* \leftarrow$ each item i such that TWU$(i) \geq minutil$;
Let $\succ$ be the total order of TWU ascending values on I^*;
Scan D to built the utility-list of each item $i \in I^*$ and build the $EUCS$;
Output each item $i \in I^*$ such that SUM$(\{i\}.utilitylist.iutils) \geq minutil$;
`Search` ($\emptyset$, I^*, *minutil*, $EUCS$);

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 Property 5. It can be easily seen based on Property 1, 2 and 3 that this procedure is correct and complete to discover all high-utility itemsets.

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
output: the set of high-utility itemsets

```
foreach itemset Px ∈ ExtensionsOfP do
    if SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.rutils) ≥ minutil then
        ExtensionsOfPx ← ∅;
        foreach itemset Py ∈ ExtensionsOfP such that y ≻ x do
            if ∃(x, y, c) ∈ EUCS such that c ≥ minutil then
                Pxy ← Px ∪ Py;
                Pxy.utilitylist ← Construct (P, Px, Py);
                ExtensionsOfPx ← ExtensionsOfPx ∪ Pxy;
                if SUM(Pxy.utilitylist.iutils) ≥ minutil then output Px;
            end
        end
        Search (Px, ExtensionsOfPx, minutil);
    end
end
```

We now explain how the algorithm is modified to mine only CHIs, rather than all HUIs. The modified algorithm takes *minbond* as an additional parameter.

Calculating the bond measure efficiently. To mine only CHIs, it is important to be able to calculate the bond of any itemset efficiently. A naive approach would be to scan the database for each HUI to calculate its disjunctive support and conjunctive support, to then calculate its bond. However, this

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

```
UtilityListOfPxy ← ∅;
foreach tuple ex ∈ Px.utilitylist do
    if ∃ey ∈ Py.utilitylist and ex.tid = exy.tid then
        if P.utilitylist ≠ ∅ then
            Search element e ∈ P.utilitylist such that e.tid = ex.tid.;
            exy ← (ex.tid, ex.iutil + ey.iutil − e.iutil, ey.rutil);
        end
        else exy ← (ex.tid, ex.iutil + ey.iutil, ey.rutil);
        UtilityListOfPxy ← UtilityListOfPxy ∪ {exy};
    end
end
return UtilityListPxy;
```

would be inefficient. A better approach, used in FCHM, is the following. A structure called *disjunctive bit vector* [7] is appended to each utility-list to efficiently calculate the disjunctive support of any itemset.

Definition 4 (Disjunctive bit vector). 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, $bv(a) = 10110$, $bv(b) = 11001$ and $bv(c) = 11111$. 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 easily obtained using its utility-list and disjunctive bit vector, as follows.

Property 5 (Calculating the bond of an itemset using its utility-list). Let 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 algorithm is modified to only output HUIs that are correlated itemsets (having a bond no less than *minbond*). The resulting algorithm is correct and complete

for mining CHIs. To improve the performance of FCHM, the next paragraphs describe four additional strategies incorporated in FCHM to more efficiently discover CHIs.

Strategy 1. Directly Outputting Single items (DOS). A first optimization is based on the observation that the bond of single items is always equal to 1. Thus, HUIs containing a single item can be directly output without calculating their bond.

Strategy 2. Pruning Supersets of Non correlated itemsets (PSN). Because the bond measure is anti-monotonic (Property 5), if the bond of an itemset Pxy is less than *minbond*, any extensions of Pxy should not be explored. Thus, Line 8 of Algorithm 1 is modified so that if $bond(Pxy) < minutil$ for an itemset Pxy, the utility-list of Pxy is not added to the set $ExtensionsOfPx$.

Strategy 3. Pruning using the Bond Matrix (PBM). 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 *Bond Matrix* is created to store the bond of all itemsets containing two items from I^*. The design of this structure is similar to the EUCS. The Bond Matrix is formally defined as follows. The Bond 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 $bond(\{a, b\}) = c$. Note that it only stores tuples of the form (a, b, c) such that $c \neq 0$. For example, the Bond Matrix for the running example is shown in Fig. 2.

Then, Line 5 of Algorithm 1 is modified to add the condition that the utility-list of Pxy should only be built if $\exists(x, y, c) \in BondMatrix$ such that $c \geq minbond$. It can be easily seen that this pruning condition preserve the correctness and completeness of FCHM, because any superset of an itemset $\{x, y\}$ such that $bond(\{x, y\}) < minbond$ is not a CHIs by Property 5.

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 6 (Required conjunctive support for an extension Pxy). An itemset Pxy is a correlated itemset only if its conjunctive support is no less than the value $lowerBound(Pxy) = \lceil |consup(Pxy)| * minBond) \rceil$ transactions.

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 $consup(Pxy) = |bv(Pxy)|$. A second modification is to create a variable *maxSupport* that is initialized to the conjunctive support of Px ($consup(Px) = |bv(Px)|$, 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 conjunctive support of Pxy will not be higher than $lowerBound(Pxy)$. Thus Pxy is not a CHI by Property 6, and its extensions can also be ignored by Property 5. 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 [8], and it is thus not described here.

4 Experimental Study

We performed an experiment to assess the performance of FCHM on a computer having a third generation 64 bit Core i5 processor running Windows 7, and 5 GB of free RAM. We compared the performance of the proposed FCHM algorithm with the state-of-the-art FHM algorithm for mining HUIs. The experiment was carried on four real-life datasets commonly used in the HUIM litterature: *mushroom*, *retail*, *kosarak* and *foodmart*. These datasets have varied characteristics and 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. *mushroom* is a dense dataset ($|I|$ = 16,470, $|D|$ = 88,162, A = 23). *kosarak* is a dataset that contains many long transactions ($|I|$ = 41,270, $|D|$ = 990,000, A = 8.09). *retail* is a sparse dataset with many different items ($|I|$ = 16,470, $|D|$ = 88,162, A = 10,30). *foodmart* is a sparse dataset ($|I|$ = 1,559, $|D|$ = 4,141, A = 4.4). *foodmart* contains real external and internal utility values. For the other datasets, external utilities for items are generated between 1 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, as the settings of [12, 15]. 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. For the experiment, FCHM was run with five different *minbond* threshold values (0.1, 0.3, 0.5, 0.7 and 0.9). Thereafter, the notation FCHMx represents FCHM with $minbond = x$. Algorithms were first 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.

It can first be observed that mining CHIs using FCHM can be much faster than mining HUIs. The reason for the excellent performance of FCHM is that it prunes a large part of the search space using its designed strategies for pruning using the bond measure. For datasets containing a huge amount of weakly correlated HUIs such as *mushroom*, *foodmart*, and *kosarak*, this leads to a massive performance improvement. For example, for the lowest *minutil* values and $minbond = 0.5$ on these datasets, FCHM is respectively 103, 10 and 21 times

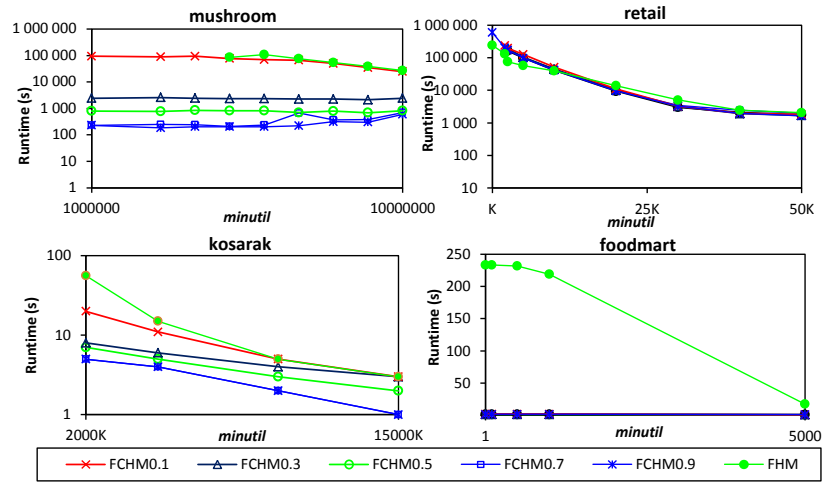

Fig. 3: Execution times

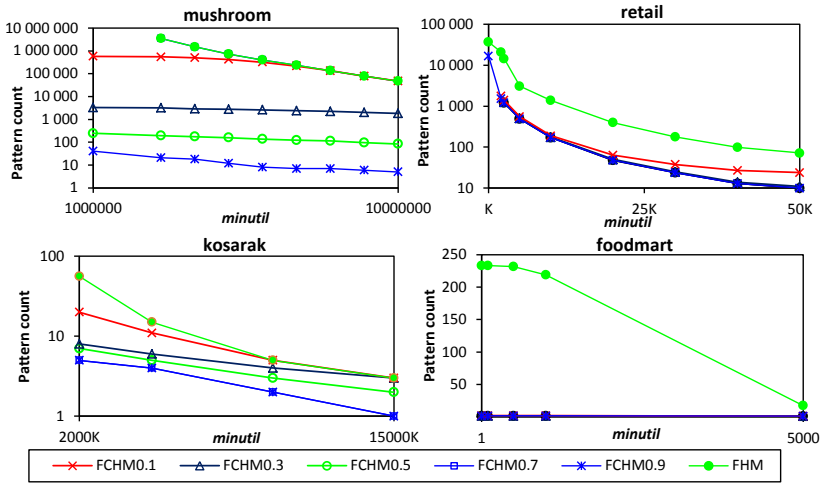

Fig. 4: Number of patterns found

faster than FHM. In general, when *minbond* increases, the gap between the runtime of FHM and FCHM increases. For example, for *minbond* = 0.9 on *mushroom*, FCHM is 417 times faster than FHM.

A second observation is that the number of CHIs can be much less than the number of HUIs, even when using low *minbond* threshold values (see Fig. 4). For example, on *mushroom*, 3,538,181 HUIs are found for $minutil = 3,000,000$. But only 3,186 HUIs are CHIs for $minbond = 0.3$ and only 196 for $minbond = 0.5$ (about 1 CHIs for 18,000 HUIs). Huge reduction in the number of patterns are

also observed on the *foodmart* and *kosarak* datasets. On the *retail* dataset, the reduction is less because patterns are more correlated. However, there is still from 2 to 10 times less CHIs than HUIs on this dataset, depending on the *minutil* values. These overall results show that the proposed FCHM algorithm is useful as it can filter a huge amount of weakly correlated itemsets encountered in real datasets, and can run faster.

Memory consumption was also compared, although detailed results are not shown as a figure due to space limitation. It was observed that FCHM can use up to 30 times less memory than FHM depending on how the *minbond* threshold is set. For example, on *mushroom* and $minutil = 5,000,000$, FHM, FCHM0.9, FCHM0.7, FCHM0.5, FCHM0.3, and FCHM0.1, respectively consume 666 MB, 424 MB, 136 MB, 59 MB, 26 MB and 20 MB.

Lastly, the efficiency of the proposed PBM and AUL strategies was also assessed. Detailed results are not shown due to space limitation. But these strategies were shown to prune a huge amount of candidates for most datasets, and it greatly increases when *minbond* is increased. For example, on *mushroom* and $minutil = 5,000,000$, FHM, FCHM0.9, FCHM0.7, FCHM0.5, FCHM0.3, and FCHM0.1, visit 1,282,377, 536,136, 5,254, 1,160, 883, and 848 candidates.

5 Conclusion

This paper proposed an algorithm named FCHM (Fast Correlated high-utility itemset Miner) to efficiently discover correlated high-utility itemsets using the bond measure. An extensive experimental study shows that FCHM is up to two orders of magnitude faster than FHM, and can discover more than five orders of magnitude less patterns by only mining correlated HUIs. 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/`. For future work, ideas introduced in FCHM could be considered in incremental high-utility itemset mining [9], and high-utility sequential rule mining [16]. Besides, we will consider developping an algorithm for correlated high-utility itemset mining based on the EFIM algorithm [17], since EFIM was shown to outperform FHM for HUIM.

Acknowledgement. This research was partially supported by National Natural Science Foundation of China (NSFC) under grant No.61503092.

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. 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)
6. 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)
7. 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.
8. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Systems with Applications, 42(5), 2371-2381 (2015)
9. Lin, J. C. W., Gan, W., Hong, T. P., Pan, J. S.: Incrementally Updating High-Utility Itemsets with Transaction Insertion. Proc. 10th Int. Conf. on Advanced Data Mining and Applications, pp. 44–56 (2014)
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.