A New Method for Mining Frequent Weighted Itemsets Based on WIT-trees

Bay Vo[1], Frans Coenen[2] and Bac Le[3,*]
[1]Department of Computer Science
Information Technology College
Ho Chi Minh, Viet Nam
vdbay@itc.edu.vn
[2] Department of Computer Science
University of Liverpool, UK
Coenen@liverpool.ac.uk
[3]Department of Computer Science
University of Science
Ho Chi Minh, Viet Nam
lhbac@fit.hcmus.edu.vn

Abstract: The mining frequent itemsets plays an important role in the mining of association rules. Frequent itemsets are typically mined from binary databases where each item in a transaction may have a different significance. Mining Frequent Weighed Itemsets (FWI) from weighted items transaction databases addresses this issue. This paper therefore proposes algorithms for the fast mining of FWI from weighted item transaction databases. Firstly, an algorithm for directly mining FWI using WIT-trees is presented. After that, some theorems are developed concerning the fast mining of FWI. Based on these theorems, an advanced algorithm for mining FWI is proposed. Finally, a Diffset strategy for the efficient computation of the weighted support for itemsets is described, and an algorithm for mining FWI using Diffsets presented. A complete evaluation of the proposed algorithms is also presented.

Keywords: Data Mining, Frequent Weighted Itemset, Frequent weighted support, WIT-trees, WIT-FWI.

[*] Corresponding author.

1. Introduction

Association Rule Mining (ARM) is an important element within the domain of Knowledge Discovery in Data (KDD) [1, 18]. ARM is used to identify relationships amongst items in transaction databases. Given a set of items $I = \{i_1, i_2, ..., i_n\}$, a transaction is defined as a subset of I. The input to an ARM algorithm is a dataset D comprising a set of transactions. Given an itemset $X \subseteq I$, the support of X in D, denoted as $\sigma(X)$, is the number of transactions in D which contain X. An itemset is described as being frequent if its support is larger than or equal to a user supplied minimum support threshold (*minSup*). A "classical" Association Rule (AR) is an expression of the form $\{X \square Y$ (*sup*, *conf*)$\}$, where X, $Y \subseteq I$ and $X \cap Y = \emptyset$. The support of this rule is $sup = \sigma(XY)$ and the confidence is $conf = \frac{\sigma(XY)}{\sigma(X)}$. Given a specific *minSup* and a minimum confidence threshold (*minConf*), we want to mine all association rules whose support and confidence exceeds *minSup* and *minConf* respectively.

However, "Classical" ARM does not take into consideration the relative benefit or significance of items. With respect to some applications, we are interested in the relative benefit (weighted value) associated with each item. For example the sale of bread may incur a profit of 20 cents while a bottle of milk might realize a profit of 40 cents. It is thus desirable to identify new methods for applying ARM techniques to this kind data so that such relative benefits are taken into account.

In 1998, Ramkumar *et al.* [13] (see also [2]) proposed a model for describing the concept of Weighted Association Rules (WAR) and presented an Apriori-based algorithm for mining Frequent Weighted Itemsets (FWI). Since then many Weighted Association Rule Mining (WARM) techniques have been proposed (see for example Wang *et al.* [19] and Tao *et al.* [14]).

The purpose of this paper is to develop algorithms for the fast mining of FWI. Firstly, some theorems and corollary are proposed. Based on these theorems, and using WIT-trees [10, 11], we present an algorithm for the fast mining of FWI. After that, we apply the Diffset strategy [20, 22] and extend the originally proposed FWI mining algorithm.

The rest of this paper is organized as follows. Section 2 presents some related work concerning the mining of FWI and WAR. Section 3 presents a proposed modification of the WIT-tree data structure [10, 11] for compressing the database into a tree structure. Algorithms for mining FWI using WIT-trees are discussed in section 4. Some experimental results are present in section 5, and some conclusions in section 6.

2. Related work

This section presents some related works. The section commences with a formal definition of weighted transaction databases. The Galois connection, used later in this paper to prove a number of theorems, is then reviewed in Sub-section 2.2. Next, in Sub-section 2.3, some definitions related to weighted association rules are presented.

2.1. Weighted items transaction databases

A weighted transaction database (*D*) is defined as follows: *D* comprises a set of transactions $T = \{t_1, t_2, ..., t_m\}$, a set of items $I = \{i_1, i_2, ..., i_n\}$ and a set of positive weights $W = \{w_1, w_2, ..., w_n\}$ corresponding to each item in *I*.

For example, consider the data presented in Table 1 and Table 2. Table 1 presents a data set comprising six transactions $T = \{t_1, \ldots, t_6\}$, and five items $I = \{A, B, C, D, E\}$. The weights of these items are presented in Table 2, $W = \{0.6, 0.1, 0.3, 0.9, 0.2\}$.

Table 1. The transaction database

Transactions	Bought items
1	*A, B, D, E*
2	*B, C, E*
3	*A, B, D, E*
4	*A, B, C, E*
5	*A, B, C, D, E*
6	*B, C, D*

Table 2. Items weight

Item	Weight
A	0.6
B	0.1
C	0.3
D	0.9
E	0.2

2.2. Galois connection

Let $\delta \subseteq I \times T$ be a binary relation, where *I* is a set of items and *T* is a set of transactions contained in a database *D*. Let ***P***(***S***) (the power set of S) include all subsets of ***S***. Two mappings between ***P***(***I***) and ***P***(***T***) are called Galois connections as follows [21].

Let $X \subseteq I$ and $Y \subseteq T$, we have:

i. $t : P(I) \mapsto P(T),\ t(X) = \{y \in T \mid \forall x \in X, x\delta\, y\}$

ii. $i : P(T) \mapsto P(I),\ i(Y) = \{x \in I \mid \forall y \in Y, x\delta\, y\}$

The mapping *t*(*X*) is the set of transactions in the database which contain *X*, and the mapping *i*(*Y*) is an itemset that is contained in all the transactions *Y*.

Given X, X_1, $X_2 \in P(I)$ and Y, Y_1, $Y_2 \in P(T)$. The Galois connection satisfies the following properties [21]:

i) $X_1 \subset X_2 \Rightarrow t(X_1) \supseteq t(X_2)$

ii) $Y_1 \subset Y_2 \Rightarrow i(Y_1) \supseteq i(Y_2)$

iii) $X \subseteq i(t(X))$ and $Y \subseteq t(i(Y))$

2.3. Mining frequent weighted itemsets

Definition 2.1. The transaction weight (*tw*) of a transaction t_k is defined as follows:

$$tw(t_k) = \frac{\sum_{i_j \in t_k} w_j}{|t_k|} \tag{2.1}$$

Definition 2.2. The weighted support of an itemset is defined as follows:

$$ws(X) = \frac{\sum_{t_k \in t(X)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} \tag{2.2}$$

where T is the list of transactions in the database.

Example 2.1. Consider tables 1, 2, and definition 2.1, we can compute the $tw(t_1)$ value as follow:

$$tw(t_1) = \frac{0.6 + 0.1 + 0.9 + 0.2}{4} = 0.45$$

Table 3 shows all *tw* values of transactions in Table 1.

Table 3. Transaction weights for transactions in Table 1

Transactions	*tw*
1	*0.45*
2	*0.2*
3	*0.45*
4	*0.3*
5	*0.42*
6	*0.43*
Sum	*2.25*

From Tables 1 and 3, and definition 2.2, we can compute the *ws*(*BD*) value as follows: Because *BD* appears in transactions {1, 3, 5, 6}, *ws*(*BD*) is computed:

$$ws(BD) = \frac{0.45 + 0.45 + 0.42 + 0.43}{2.25} \approx 0.78$$

The mining of FWI requires the identification of all itemsets whose weighted support satisfies an user specified minimum weighted support threshold (*minws*), i.e. FWI = $\{X \subseteq I | ws(X) \geq minws\}$.

Theorem 2.1. The use of the weighted support metric described above satisfies the *downward closure property*. i.e., if $X \subset Y$ then $ws(X) \geq ws(Y)$.

Proof: Because $X \subset Y$, according to the property i) of Galois connection, we have $t(X) \supseteq t(Y)$

$$\Rightarrow \sum_{t_k \in t(X)} tw(t_k) \geq \sum_{t_k \in t(Y)} tw(t_k) \Rightarrow \frac{\sum_{t_k \in t(X)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} \geq \frac{\sum_{t_k \in t(Y)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} \Rightarrow ws(X) \geq ws(Y).$$

To mine WAR, we must first mine all FWI that satisfy the minimum weighted support threshold. The mining of FWI is the most computationally expensive element of WAR mining. In 1998, Ramkumar *et al.* [13] proposed an Apriori-based algorithm for mining FWI. This approach requires many scans of the whole database to determine the weighted support of itemsets. Some other studies used this approach for generating WAR [14, 19].

3. WIT-tree data structure

We proposed the WIT-tree (Weighted Itemset-Tidset tree) data structure, an expansion of the IT-tree proposed in [22], to support the mining of high utility itemsets. The WIT-tree data structure provides for a representation of the input data (so that we only need scan the database once), comprising of itemset TID lists, that supports the fast computation of weighted support values. Each node in a WIT-tree includes 3 fields:

i. X: an itemset.

ii. $t(X)$: the set of transactions contains X.

iii. ws: the weighted support of X.

The node is denoted using a tuple of the form $\langle X, t(X), ws \rangle$.

The value for *ws* is computed by summing all *tw* values of transactions, *t*(*X*), which their *tids* belong to and then dividing this by the sum of all *tw* values. Thus, computing of *ws* is founded on Tidset. The links connect nodes at k^{th} level (called *X*) with nodes at the $(k+1)^{th}$ level (called *Y*).

Definition 3.1[22] – The equivalence class

Let *I* be a set of items and $X \subseteq I$, a function ***p*(*X*,k) = *X*[1:k]** as the *k* length prefix of *X* and a prefix-based equivalence relation θ_K on itemsets as follows: $\forall X, Y \subseteq I, X \equiv_{\theta_k} Y \Leftrightarrow p(X,k) = p(Y,k)$.

The set of all itemsets having the same prefix *X* is called an equivalence class, and is denoted as the equivalence class with prefix *X* is [*X*].

Example 3.1: Consider Tables 1 and 3 above, the associated WIT-tree for mining frequent weighted itemsets is as presented in Figure 1.

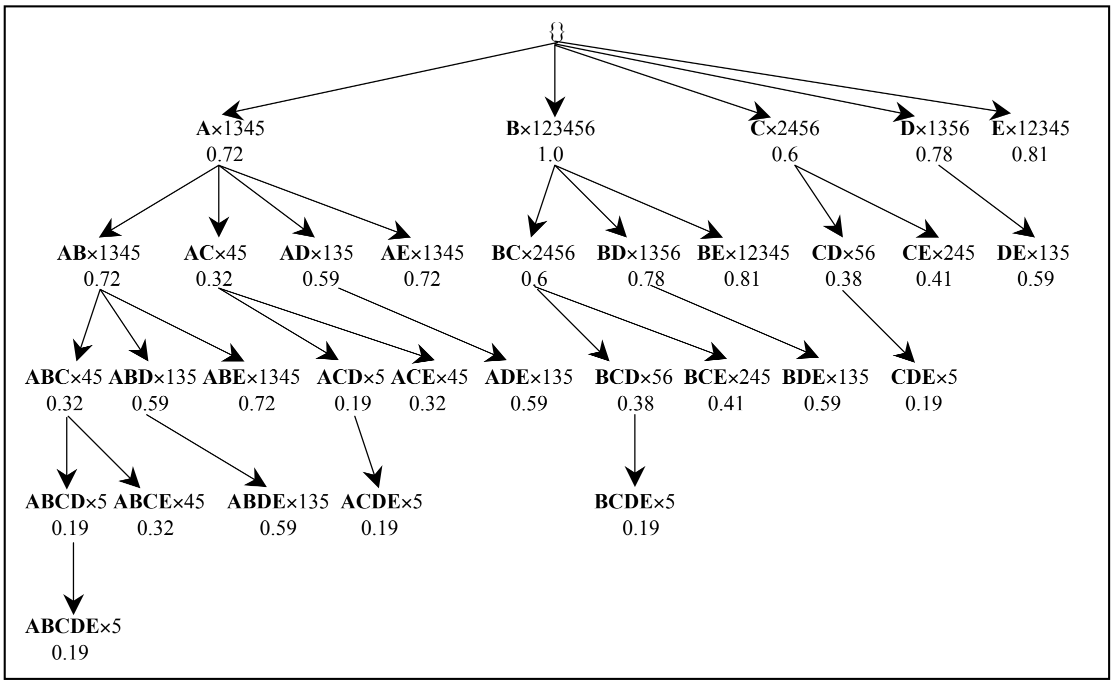

Figure 1. Search tree using WIT-tree

The root node of the WIT-tree contains all 1-itemset nodes. All nodes in level 1 belong to the same equivalence class with prefix {} (or [∅]). Each node in level 1 will become a new equivalence class using its item as the prefix. With each node in the same prefix, it will join with all nodes following it to create a new equivalence class. The process will be done recursively to create new equivalence classes in higher levels. For example, considering Figure 1, nodes $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$ belong to the equivalence class $[\varnothing]$. Consider node $\{A\}$, this node will join with all nodes following it ($\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$) to create a new equivalence class $[A] = \{\{AB\}, \{AC\}, \{AD\}, \{AE\}\}$. $[AB]$ will become a new equivalence class by also joining with all nodes following it ($\{AC\}$, $\{AD\}$, $\{AE\}$); and so on.

Inspection of Figure 1 indicates that all itemsets satisfy the *downward closure property*. Thus, we can prune an equivalence class in the WIT-tree if its *ws* value does not satisfy the *minws*. For example, suppose that *minwus* = 0.4, because $ws(ABC) = 0.32 < minws$ we can prune the equivalence class with the prefix *ABC*, i.e., all child nodes of *ABC* can be pruned.

4. Mining frequent weighted itemsets

In this section, we propose algorithms for mining FWI from weighted transaction databases. Firstly, an algorithm for directly mining FWI from WIT-trees is presented. It uses a *minws* threshold and the *downward closure property* to prune nodes that are not frequent. Some theorems are then derived and based on these theorems, an improved algorithm is proposed. Finally, the algorithm is further developed, by adopting a Diffset strategy to allow for the fast computation of the weighted support of itemsets in a memory efficient manner.

4.1. WIT-FWI algorithm

In this sub-section, an algorithm for mining FWI using WIT-trees will be present. It is founded on the *downward closure property* to prune nodes that are not frequent.

In function **WIT-FWI** (Figure 2), let L_r contains all single items that their weighted supports satisfy minimum weighted support (line 1). Nodes in L_r are stored in increasing order according to their weighted support (line 2). After that, the set of FWI is set to null (line 3). Finally, the **FWI-EXTEND** function will be called to mine all FWI (line 4).

Consider function **FWI-EXTEND**: This function considers each node l_i in L_r with all nodes following it to create a set of nodes L_i (lines 5 and 7). The way to create L_i as follows: Firstly, let $X = l_i$.itemset ∪ l_j.itemset, it computes $Y = t(X) = t(l_i) \cap t(l_j)$ (line 8), if $ws(X)$ (computed through $t(X)$ using equation 2.2, line 9) satisfies *minws* (line 10), we add new node $<X, Y, ws(X)>$ into L_i (line 11). After creating L_i, function **FWI-EXTEND** is called recursively to process with the input is L_i (line 13) if number of nodes in L_i greater than 1. Function **COMPUTE-WS**(Y) uses the eq. (2.2) to compute the *ws* of itemset X based on the values in Table 3 with $Y = t(X)$ (line 12).

```
Input: Database $D$ and minimum weighted support threshold minws
Output: FWI contains all frequent weighted itemsets that satisfy minws from $D$
Method:
  WIT-FWI( )
  1. $L_r$ = all items that their ws satisfy minws
  2. Sort nodes in $L_r$ increasing by their ws
  3. FWI = $\varnothing$
  4. Call function FWI_EXTEND with the parameter is $L_r$

  FWI-EXTEND($L_r$)
  5. Consider each node $l_i$ in $L_r$ do
  6.     Add ($l_i$.itemset, $l_i$.ws) to FWI
  7.     Create a new set $L_i$ by join $l_i$ with all $l_j$ following it in $L_r$ by:
  8.          Set $X = l_i$.itemset ∪ $l_j$.itemset and  $Y = t(l_i) \cap t(l_j)$
  9.          $ws(X)$ = COMPUTE-WS($Y$)   // using the eq. (2.2)
  10.         if $ws(X)$ satisfies minws then
  11.             Add new node $<X, Y, ws(X)>$ into $L_i$
  12.    if number of nodes in $L_i \geq 2$ then
  13.        Call recursive the function FWI-EXTEND with the parameter is $L_i$
```

Figure 2. **WIT-FWI** algorithm for mining frequent weighted itemsets

Example 4.1: Using the data presented in Tables 1 and 3, and with reference to Figure 2, an illustration of the **WIT-FWI** algorithm with *minws* = 0.4 is as follows:

We commence by computing the *ws* values of the single items. We have $ws(A) = 0.72$, $ws(B) = 1.0$, $ws(C) = 0.60$, $ws(D) = 0.78$, $ws(E) = 0.81$. All *ws* values of single items satisfy *minws* $\Rightarrow L_r = \{<A,1345,0.72>, <B,123456,1.0>, <C,2456,0.6>, <D,1356,0.78>, <E,12345,0.81>\}$. After sorting them according to their weighted supports, we have $L_r = \{<C,2456,0.6>, <A,1345,0.72>, <D,1356,0.78>, <E,12345,0.81>, <B,123456,1.0>\}$.

Consider node <*A*, 1345, 0.72>: First of all, *A* is added to **FWI** = {*C*, *CE*, *CEB*, *CB*} ⇒ **FWI** = {*C*, *CE*, *CEB*, *CB*, *A*}.

- *A* joins *C*, we have a new itemset *AC*. Because $ws(AC) = 0.32 < minws$, *AC* is not added into L_A.
- *A* joins *D*, we have a new itemset *AD* with $t(AD) = 135$ and $ws(AD) = 0.59$, so add *AD* into L_A $\Rightarrow L_A = \{<AD,135,0.59>\}$.
- *A* joins *E*, we have a new itemset *AE* with $t(AE) = 1345$ and $ws(AE) = 0.72$ ⇒ Add node *AE* into $L_A \Rightarrow L_A = \{<AD,135,0.59>, <AE,1345,0.72>\}$.
- *A* joins *B*, we have a new itemset *AB* with $t(AB) = 1345$ and $ws(AB) = 0.72 \geq minws \Rightarrow$ Add node *AB* into $L_A \Rightarrow [A] = \{<AD,135,0.59>, <AE,1345,0.72>,<AB,1345,0.72>\}$.

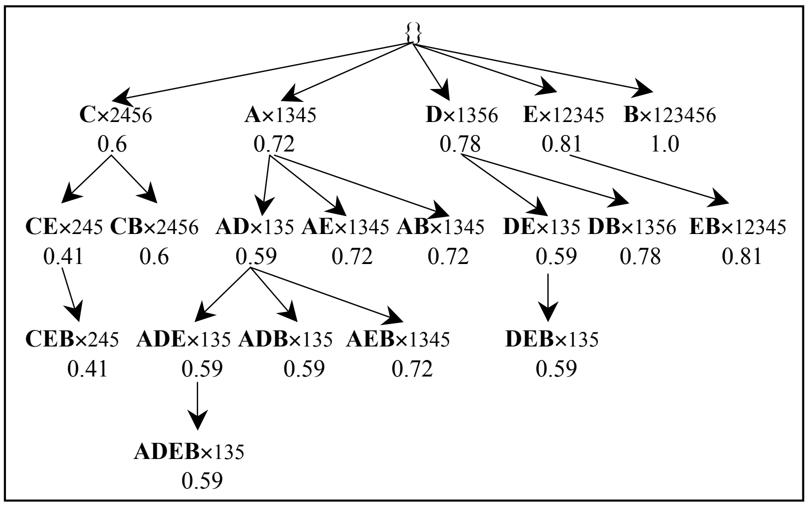

Figure 3. WIT-tree for mining FWI from the databases in Tables 1 and 3 with *minws* = 0.4

After creating the set L_A, and because the number of nodes in L_A is larger than 1, the function will be called (in a recursive manner) to create all child nodes of L_A.

- Consider node <*AD*,135, 0.59>:
  - Add *AD* to **FWI** ⇒ **FWI** = { *C*, *CE*, *CEB*, *CB*, *A*, *AD*}.
  - *AD* joins *AE*, we have a new itemset *ADE*×135 with *ws*(*ADE*) = 0.59, so add *ADE* to [*AD*] ⇒ [*AD*] = {*ADE*}.
  - *AD* joins *AB*, we have a new itemset *ADB*×135 with *wus*(*ADB*) = 0.59, so add *ADB* to [*AD*] ⇒ [*AD*] = {*ADE*, *ADB*}.
    - Consider node <*ADE*,1345,0.72>: Add *ADE* to **FWI** ⇒ **FWI** = { *C*, *CE*, *CEB*, *CB*, *A*, *AD*, *ADE*}.
    - *ADE* joins *ADB* into a new itemset *ADEB*×135 with *wus*(*ADEB*) = 0.59 ⇒ [*ADE*] = {*ADEB*}. Because the equivalence class [*ADE*] has only one element ⇒ there is no any equivalence class created in following it.

Similar to nodes <*C*,2456,0.6>, <*D*,1356,0.78>, <*E*,12345,0.81>, <*B*,123456,1.0>.

Finally, we have the set of all FWI that satisfy *minwus* = 0.4 is **FWI** = { *C, CE, CEB, CB, A*, *AD*, *ADE, ADEB, ADB, AE, AEB, AB, D, DE, DEB, DB, E, EB, B*} as in Figure 3.

4.2. An improved algorithm

From Figure 2, we can see that with respect to some nodes we need not compute the weighted support because this can be obtained from the parent nodes. For example, nodes *AE*, *AB* and *AEB* have the same weighted support as node *A*; node *ADE*, *ADB* and *ADEB* have the same weighted support as node *AD*; and so on.

Theorem 4.1: Given two itemsets *X* and *Y*, if *t*(*X*) = *t*(*Y*) then *ws*(*X*) = *ws*(*Y*)

Proof: Because *t*(*X*) = *t*(*Y*) ⇒ $\sum_{t_k \in t(X)} tw(t_k) = \sum_{t_k \in t(Y)} tw(t_k) \Rightarrow \frac{\sum_{t_k \in t(X)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} = \frac{\sum_{t_k \in t(Y)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)}$ or $ws(X) = ws(Y)$.

Corollary 4.1: If $X \subset Y$ and $|t(X)| = |t(Y)|$ then $ws(X) = ws(Y)$

Proof: If $X \subset Y$, we have $t(X) \supseteq t(Y)$ (according to the property i) of the Galois connection). Besides, because $|t(X)| = |t(Y)| \Rightarrow t(X) = t(Y) \Rightarrow ws(X) = ws(Y)$ according to the theorem 4.1.

Based on Corollary 4.1, we developed an algorithm for mining FWI by introducing some modifications to the algorithm presented in the section 4.1. When we join two nodes l_i, l_j of L_r to create a new node $l_i \cup l_j$, if $|t(l_i)| = |t(l_i \cup l_j)|$ then $ws(l_i \cup l_j) = ws(l_i\text{.itemset})$, we need not compute the ws value of l_i.itemset $\cup$ l_j.itemset. Similarly, if $|t(l_j)| = |t(l_i \cup l_j)|$ then $ws(l_i \cup l_j) = ws(l_i)$, we need not also compute the ws value of $l_i \cup l_j$.

```
Input: Database D and minimum weighted support threshold minws
Output: FWI contains all frequent weighted itemsets that their ws satisfy minws from D
Method:
  WIT-FWI-MODIFY()
  1. $L_r$ = all items that their ws satisfy minws
  2. Sort nodes in $L_r$ increasing by their ws
  3. FWI = $\varnothing$
  4. Call function FWI_EXTEND with the parameter is $L_r$

  FWI-EXTEND-MODIFY($L_r$)
  5. Consider each node $l_i$ in $L_r$ do
  6.    Add ($l_i$.itemset, $l_i$.ws) to FWI
  7.    Create a new set $L_i$ by join $l_i$ with all $l_j$ following it in $L_r$ by:
  8.         Set $X = l_i\text{.itemset} \cup l_j\text{.itemset}$ and $Y = t(l_i) \cap t(l_j)$
  9.         if $|t(l_i)| = |Y|$ then $ws(X) = ws(l_i)$        // using corollary 4.1
  10.        elseif $|t(l_j)| = |Y|$ then $ws(X) = ws(l_j)$   // using corollary 4.1
  11.        else $ws(X)$ = COMPUTE-WS($Y$)    // using the eq. (2.2)
  12.        if $ws(X)$ satisfies minws then
  13.             Add new node <$X$, $Y$, $ws(X)$> to $L_i$.
  14.   if number of nodes in $L_i \geq 2$ then
  15.       Call recursive the function FWI-EXTEND-MODIFY with the parameter is $L_i$
```

Figure 4. The modification of **WIT-FWI** algorithm for mining frequent weighted itemsets

The algorithm in Figure 2 is modified as follow: Line 9 (in Figure 2) is changed by 3 lines (from lines 9 to 11 in Figure 4). In line 9, we have l_i.itemset $\subset X$, according to corollary 4.1, if $|t(l_i)| = |Y|$ then $ws(X) = ws(l_i)$. Similarly, if $|t(l_j)| = |Y|$ then $ws(X) = ws(l_j)$ (line 11).

Example 4.2: Consider the WIT-tree presented in Figure 3, when we join node <*A*,1345, 0.72> with node <*B*,123456,1.0> to create new node <*AB*,1345,?>, because of $|t(A)| = |1345| = |t(AB)| \Rightarrow ws(AB) = ws(A) = 0.72$. Similarly, $|t(C)| = |2456| = |t(BC)| \Rightarrow ws(BC) = ws(C) = 0.6$. Besides, because $t(A)| \neq |t(AD)|$ and $|t(D)| \neq |t(AD)| \Rightarrow$ we must compute $ws(AD)$ (based on $t(AD)$). According to Corollary 4.1, we need not compute *ws* values of 11 itemsets, namely {*AE, AB, CB, DE, BD, EB, ADE, ADB, AEB, CEB, DEB, ABDE*} (see Figure 3 for more details).

4.4. Diffset for computing *ws* values fast and saving memory

Zaki and Gouda [20] proposed the Diffset strategy for fast computing the support of itemsets and saving memory to store Tidsets. We recognize that it can be used for fast computing the *ws* values of itemsets. Diffset computes the difference set between two Tidsets in the same equivalence class. In a dense database, the size of Diffset is smaller than the Tidset [20, 22]. Therefore, using Diffset will consume less storage and allow for the fast computing of weighted support values.

Let $d(PXY)$ be the difference set between PX and PY. We have:

$$d(PXY) = t(PX) \setminus t(PY) \text{ [20]} \tag{4.1}$$

where PX and PY are in equivalence class $[P]$.

Assume that we have $d(PX)$ and $d(PY)$, and need get $d(PXY)$: According to the results in [20], we can get it easily by computing the difference set between $d(PY)$ and $d(PX)$:

$$d(PXY) = d(PY) \setminus d(PX) \text{ [20]} \tag{4.2}$$

Based on eq. (4.1) and eq. (4.2), we can compute the *ws* value of PXY by using the $d(PXY)$ as follows:

$$ws(PXY) = ws(PX) - \frac{\sum_{t \in d(PXY)} tw(t)}{\sum_{t \in T} tw(t)} \quad (4.3)$$

Proof: We have $t(PXY) = t(PX) \cap t(PY) = t(PX) \setminus [t(PX) \setminus t(PY)] = t(PX) \setminus d(PXY) \Rightarrow$

$$ws(PXY) = \frac{\sum_{t_k \in t(PXY)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} = \frac{\sum_{t_k \in [t(PX) \setminus d(PXY)]} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} = \frac{\sum_{t_k \in t(PX)} tw(t_k) - \sum_{t_k \in d(PXY)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} = \frac{\sum_{t_k \in t(PX)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} - \frac{\sum_{t_k \in d(PXY)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)}$$

$$= ws(PX) - \frac{\sum_{t_k \in d(PXY)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)}.$$

Based on eq. (4.1), eq. (4.2) and eq. (4.3), we can use the Diffset strategy instead of using Tidsets for computing the *ws* values of itemsets in the process of mining FWI.

Theorem 4.2. If $d(PXY) = \varnothing$ then $ws(PXY) = ws(PX)$.

Proof: Because $d(PXY) = \varnothing \Rightarrow ws(PXY) = ws(PX) - \frac{\sum_{t \in d(PXY)} tw(t)}{\sum_{t \in T} tw(t)} = ws(PX)$.

To save the memory for storing Diffset and the time for computing Diffset, we sort itemsets in the same equivalence class in increasing order by their *ws*.

4.4.1. WIT-FWI-DIFF – An algorithm based on Diffset

The **WIT-FWI-DIFF** algorithm presented in Figure 5 differs from the **WIT-FWI-MODIFY** algorithm presented in Figure 4 in that it uses Diffset to compute the *ws* values. Because the first level stores the Tidsets, if L_r belongs to the first level (line 9), means that l_i and l_j belong to the first level, we use the eq. (4.1) to compute $Y = d(X) = d(l_i \cup l_j) = t(l_i) \setminus t(l_j)$ (line 11, using eq. (4.1)). From level 2 stores Diffset, we use the eq. (4.2) to compute $Y = d(X) = d(l_i \cup l_j) = d(l_j) \setminus d(l_i)$ (line 13). Line 14 uses the theorem 4.2 to fast compute $ws(X)$.

```
Input: Database $D$ and minimum weighted support threshold $minws$
Output: FWI contains all frequent weighted itemsets that satisfy $minws$ from $D$
Method:
  WIT-FWI-DIFF( )
  1. $L_r$ = all items that their $ws$ satisfy $minws$
  2. Sort nodes in $L_r$ increasing by their $ws$
  3. FWI = $\varnothing$
  4. Call function FWI-EXTEND-DIFF with the parameter is $L_r$

  FWI-EXTEND-DIFF($L_r$)
  5. Consider each node $l_i$ in $L_r$ do
  6.     Add ($l_i$.itemset, $l_i.ws$) to FWI
  7.     Create a new set $L_i$ by join $l_i$ with all $l_j$ following it in $L_r$ by:
  8.         Set $X = l_i$.itemset $\cup$ $l_j$.itemset
  9.         if $L_r$ is the first level then  $Y = t(l_i) \setminus t(l_j)$ // using eq. (4.1)
  10.        else $Y = d(l_j) \setminus d(l_i)$                // using eq. (4.2)
  11.        if $Y = \varnothing$ then $ws(X) = ws(l_i)$           // using theorem 4.2
  12.        else $ws(X)$ = COMPUTE-WS-DIFF($Y$) // using the eq. (4.3)
  13.        if $ws(X)$ satisfies $minws$ then
  14.            Add new node <$X$, $Y$, $ws(X)$> into $L_i$
  15.     if number of nodes in $L_i \geq 2$ then
  16.         Call recursive the function FWI-EXTEND-DIFF with the parameter is $L_i$
```

Figure 5. **WIT-FWI-DIFF** algorithm for mining frequent weighted itemsets

4.4.2. An example of Diffset

Using the example data presented in Tables 1 and 3, and the algorithm in Figure 6, we illustrate the **WIT-FWI-DIFF** algorithm with *minws* = 0.4 as follows. Level 1 of the WIT-tree contains single items, their *tids*, and their *ws*. They are sorted in increasing order by their |*tids*|. The purpose of this work is to compute Diffset faster. For example, consider nodes *B* and *D*, if they are not sorted, we must compute $d(BD) = t(B) \setminus t(D) = 123456 \setminus 1356 = 14$; otherwise, $d(DB) = t(D) \setminus t(B) = 1356 \setminus 123456 = \varnothing$.

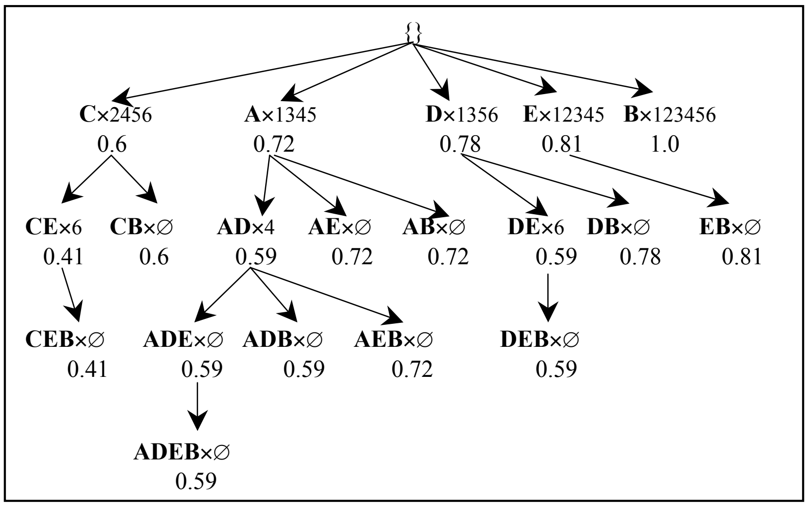

Figure 6. Results of the algorithm **WIT-FWI-DIFF** from the databases in Tables 1 and 3 with $minws = 0.4$

Consider node <*A*,1345,0.72>:

A joins *C*: $d(AC) = t(A) \setminus t(C) = 1345 \setminus 2456 = 13 \Rightarrow ws(AC) = ws(A) - \frac{\sum_{t\in d(AC)} tw(t)}{\sum_{t\in T} tw(t)} = 0.72 - \frac{0.45+0.45}{2.25} = 0.32 < minws$.

A joins *D*: $d(AD) = t(A) \setminus t(D) = 1345 \setminus 1356 = 4 \Rightarrow ws(AD) = ws(A) - \frac{\sum_{t\in d(AD)} tw(t)}{\sum_{t\in T} tw(t)} = 0.72 - \frac{0.3}{2.25} = 0.59 \geq minws$.

A joins *E*: $d(AE) = t(A) \setminus t(E) = 1345 \setminus 12345 = \varnothing \Rightarrow ws(AD) = ws(A) = 0.72$.

A joins *B*: $d(AB) = t(A) \setminus t(B) = 1345 \setminus 123456 = \varnothing \Rightarrow ws(AB) = ws(A) = 0.72$.

5. Experimental results

All experiments described below were performed on a Centrino core 2 duo (2×2.53 GHz), 4GBs RAM memory, Windows 7, using C# 2008. The experimental datasets used for the experimentation were downloaded from http://fimi.cs.helsinki.fi/data/. Some statistical information regarding these data sets is given in Table 4. We modified these datasets by creating one table to store weighted values of items (values in the range of 1 to 10) for each database.

Table 4. Experimental databases

Database (DB)	**#Trans**	**#Items**	**Remark**
BMS-POS	515597	1656	Modified
Connect	67557	130	Modified
Accidents	340183	468	Modified
Chess	3196	76	Modified
Mushroom	8124	120	Modified

Table 5. Numbers of FWI from databases

Database	***minws*(%)**	**#FWI**
BMS-POS	10	12
	8	21
	6	32
	4	85
Chess	85	2624
	80	8088
	75	20298
	70	47181
Mushroom	35	1257
	30	2937
	25	5751
	20	53853
Connect	96	1015
	94	4131
	92	11315
	90	28991
Accidents	95	15
	85	65

	75	289
	65	1035

The results presented in Table 5 show that number of FWI of BMS-POS is small, for Accidents it may be described as medium, and for Chess, Mushroom and Connect are large. It should be noted that the number of FWI found in the Connect database changes rapidly; when we change *minws* from 96% down to 90%, the number of FWI changes from 1015 up to 28991.

Experiments were also conducted to compare the processing time of our three proposed algorithms (WIT-FWI, WIT-FWI-MODIFY, WIT-FWI-DIFF) with an Apriori-based algorithm [13] (Apriori). Figures 7 to 11 show the recorded run times with respect to each of the selected above test data sets.

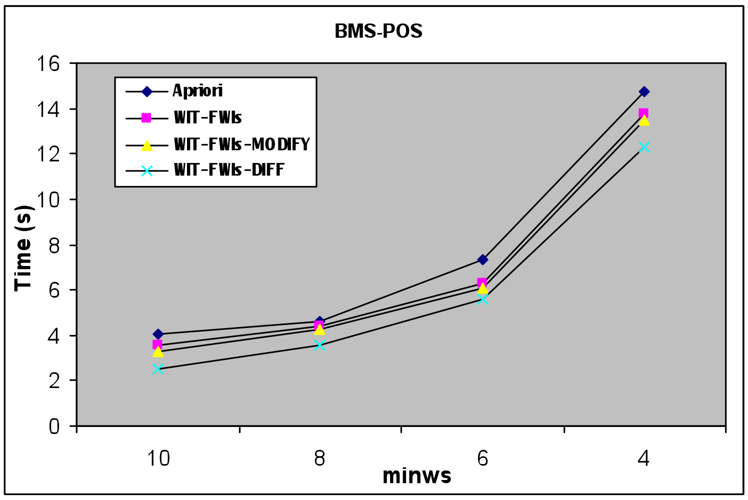

Figure 7. Run time for the four algorithms when mining FWI in the BMS-POS database

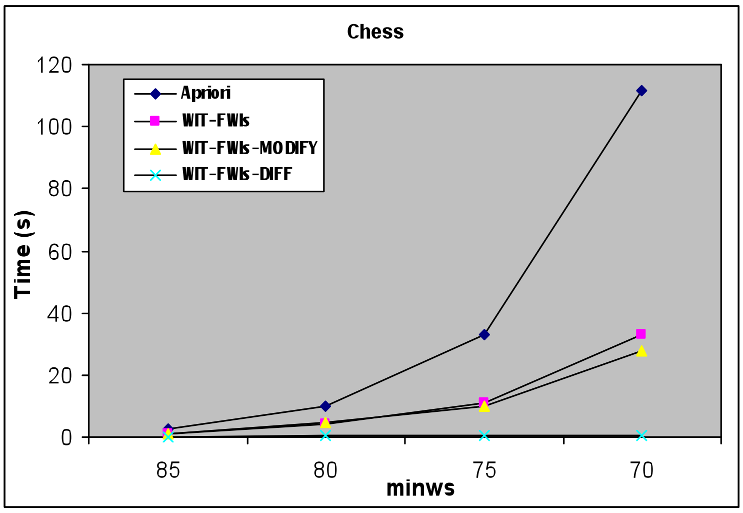

Figure 8. Run time for the four algorithms when mining FWI in the Chess database

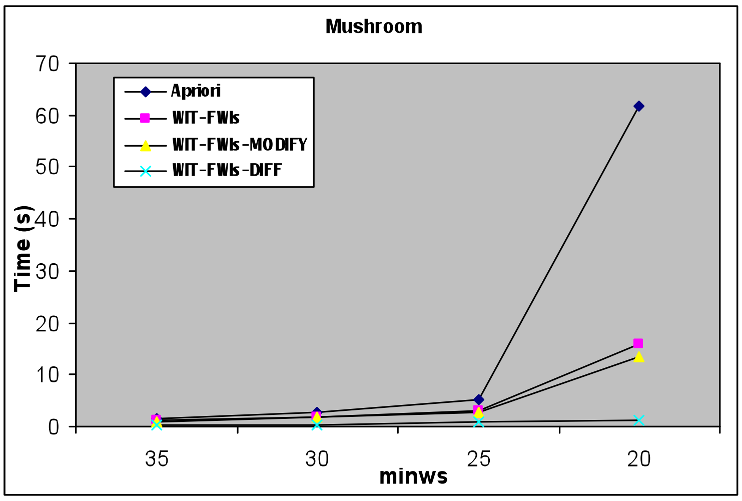

Figure 9. Run time for the four algorithms when mining FWI in the Mushroom database

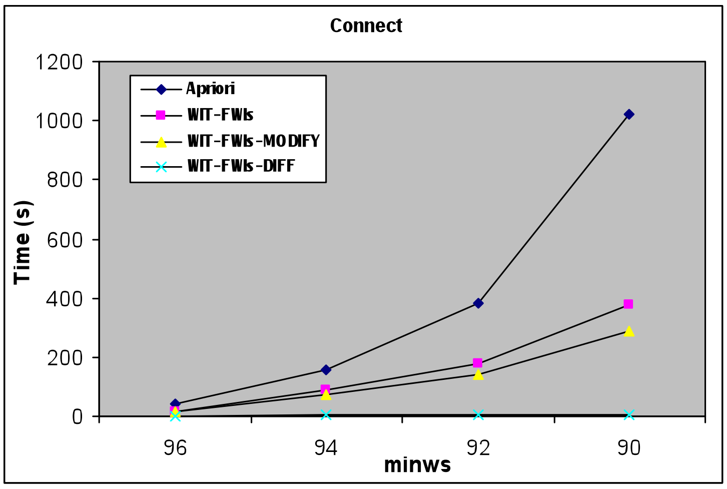

Figure 10. Run time for the four algorithms when mining FWI in the Connect database

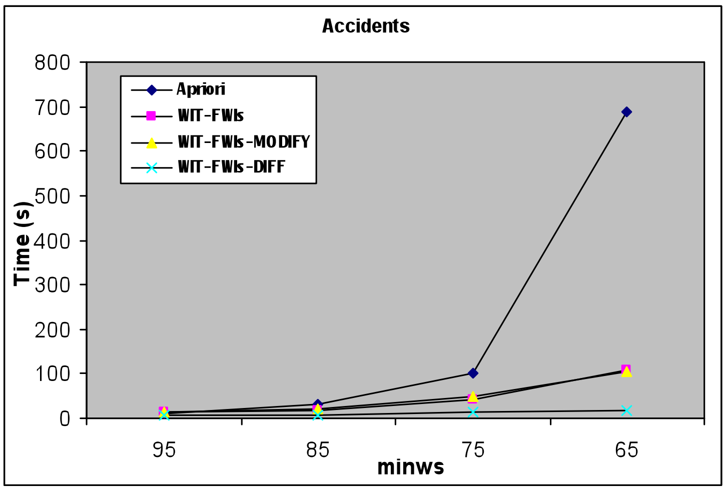

Figure 11. Run time for the four algorithms when mining FWI in the Accidents database

The experimental results presented in Figures 7 to 11 demonstrate that our proposed algorithms are more efficient than an Apriori-based algorithm when mining FWI. When the number of FWI is small, the running time of algorithms using WIT-trees is slight faster than Apriori-based algorithm. For example, consider the BMS-POS database with *minws* = 4%, the mining time of Apriori is 14.74 (s), of WIT-FWI is 13.78 (s), of WIT-FWI-MODIFY is 13.49 (s), and of WIT-FWI-DIFF is 12.27 (s). The scale run time difference between Apriori and WIT-FWI-DIFF is $\frac{12.27}{14.74}\times 100\% = 83.24\%$. However, when we compare the operation of these algorithms using the Chess and Mushroom databases (which contain a large number of FWIs) the WIT-tree based algorithms were found to be more efficient than that of the Apriori-based algorithm. For example: Consider the Chess database with *minws* = 70%, the mining time of Apriori is 111.82(s), of WIT-FWI is 33.21 (s), of WIT-FWI-MODIFY is 27.85 (s), and of WIT-FWI-DIFF is 0.7 (s). The scale difference between Apriori-based and WIT-FWI-DIFF is $\frac{0.7}{111.82}\times 100\% = 0.63\%$.

It should also be that the WIT-FWI-MODIFY algorithm is not as efficient as WIT-FWI when the number of FWI in the input database is small (for example in the case of the Accidents database).

6. Conclusions and future work

This paper has presented a method for mining frequent weighted itemsets from weighted item transaction databases, and a number of efficient algorithms have been proposed. From the reported evaluation the mining (run) time to identify FWIs using the proposed WIT-tree-based algorithms is significantly less than the time required using alternatives such as Apriori-based FWI mining algorithms. This is because using the proposed WIT-tree data structure, the algorithms only scan the database once. The evaluation also indicated that use of the Diffset strategy allows for further efficiency gains.

In this paper, we have concentrated only on the mining of FWIs (using the proposed WIT-tree data structure). In reecent years some methods for the fast mining of association rules have been discussed [15-17]. In future, we will study how to apply these methods to efficiently mine weighted association rules from discovered FWIs using our method. Besides, we will apply our method for mining weighted utility association rules [9]. The mining of association rules in incremental databases has also been considered in recent years [3-6, 12]. The intention is thus to also consider the concept of mining weighted association rules from such incremental databases.

Acknowledgement

This work was supported by Vietnam's National Foundation for Science and Technology Development (NAFOSTED).

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB'94, pp. 487 – 499 (1994).

2. Cai, C.H., Fu, A.W., Cheng, C.H., Kwong, W.W.: Mining association rules with weighted items. In: Proceedings of International Database Engineering and Applications Symposium (IDEAS 98), pp. 68 – 77 (1998).

3. Hong, T.P., Wang, C.J.: An efficient and effective association-rule maintenance algorithm for record modification. In: Expert Systems with Applications 37, pp. 618–626 (2010).

4. Hong, T.P., Wu, Y.Y., Wang, S.L.: An effective mining approach for up-to-date patterns. In: Expert Systems with Applications 36, pp. 9747–9752 (2009).

5. Hong, T.P., Lin, C.W., Wu, Y.L.: Maintenance of fast updated frequent pattern trees for record deletion. In: Computational Statistics and Data Analysis 53, pp. 2485–2499 (2009).

6. Hong, T.P., Lin, C.W., Wu, Y.L.: Incrementally fast updated frequent pattern trees. In: Expert Systems with Applications 34, pp. 2424–2435 (2008).

7. http://fimi.cs.helsinki.fi/data/ (download on April 2005).

8. Khan, M.S., Muyeba, M., Coenen, F.: Fuzzy weighted association rule mining with weighted support and confidence framework. In: Proceedings of 1st Int Workshop on Algorithms for Large-Scale Information Processing in Knowledge Discovery (ALSIP 2008), held in conjunction with PAKDD 2008 (Japan), pp. 52 – 64 (2008).

9. Khan, M.S., Muyeba, M., Coenen, F.: A weighted utility framework for mining association rules. In: Proceedings of Second UKSIM European Symposium on Computer Modeling and Simulation Second UKSIM European Symposium on Computer Modeling and Simulation, pp. 87 – 92 (2008).

10. Le, B., Nguyen, H., Cao, T.A., Vo, B.: A novel algorithm for mining high utility itemsets. In: The first Asian Conference on Intelligent Information and Database Systems, published by IEEE, pp. 13 – 16 (2009).

11. Le, B., Nguyen, H., Vo, B.: An efficient strategy for mining high utility itemsets. In: International Journal of Intelligent Information and Database Systems, 5(2), pp. 164 – 176 (2011).

12. Lin, C.W., Hong, T.P., Lu, W.H.: The Pre-FUFP algorithm for incremental mining. In: Expert Systems with Applications, 36(5), 9498–9505 (2009).

13. Ramkumar, G. D., Ranka, S., Tsur, S.: Weighted association rules: Model and algorithm. In: SIGKDD'98, pp. 661 – 666 (1998).

14. Tao, F., Murtagh, F., Farid, M.: Weighted association rule mining using weighted support and significance framework. In: SIGKDD'03, pp. 661 – 666 (2003).

15. Vo, B., Le, B.: Mining traditional association rules using frequent itemsets lattice. In: The 39th International Conference on Computers & Industrial Engineering, July 6 – 8, Troyes, France, IEEE, pp. 1401 - 1406 (2009).

16. Vo, B., Le, B.: Mining minimal non-redundant association rules using frequent itemsets lattice. In: Journal of Intelligent Systems Technology and Applications, 10(1), pp. 92 – 106 (2011).

17. Vo, B., Le, B.: Interestingness measures for association rules: Combination between lattice and hash tables. In: Expert Systems with Applications, 38(9), pp. 11630 – 11640 (2011).

18. Vo, B., Hong, T.P., Le, B.: DBV-Miner: A dynamic bit-vector approach for fast mining frequent closed itemsets. In: Expert Systems with Applications, 39(8), pp. 7196 – 7206 (2012).

19. Wang, W., Yang, J., Yu, P. S.: Efficient mining of weighted association rules. In: SIGKDD 2000, pp. 270 – 274 (2000).

20. Zaki, M. J., Gouda, K.: Fast vertical mining using diffsets. In: SIGKDD'03, pp. 326 - 335 (2003)

21. Zaki, M. J.: Mining non-redundant association rules. In: Data Mining and Knowledge Discovery, 9(3), pp. 223–248 (2004)

22. Zaki, M. J., Hsiao, C.J.: Efficient algorithms for mining closed itemsets and their lattice structure. In: IEEE Transactions on Knowledge and Data Engineering, 17(4), pp. 462-478 (2005)