An Efficient Method for Mining Frequent Weighted Closed Itemsets from Weighted Item Transaction Databases

Bay Vo[1,2]

[1]Division of Data Science, Ton Duc Thang University, Ho Chi Minh, Viet Nam
[2]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh, Viet Nam
vodinhbay@tdt.edu.vn, bayvodinh@gmail.com

Abstract: In this paper, a method for mining frequent weighed closed itemsets (FWCIs) from weighted item transaction databases is proposed. The motivation for FWCIs is that frequent weighted itemset mining, as frequent itemset (FI) mining, typically results in a substantial number of rules, which hinders simple interpretation or comprehension. Furthermore, in many applications, the generated rule set often contains many redundant rules. The inspiration for FWCIs is that one potential solution to the rule interpretation problem is to adopt frequent closed itemset. This study first proposes two theorems and a corollary. One theorem is used for checking non-closed itemsets while joining two itemsets to create a new itemset and the other theorem is used for checking whether a new itemset is non-closed itemset or not. The corollary is used for checking non-closed itemsets when using Diffsets. Based on these theorems and corollary, an algorithm for mining FWCIs is proposed. Finally, a Diffset-based strategy for the efficient computation of the weighted supports of itemsets is described. A complete evaluation of the proposed algorithm is presented.

Keywords: Frequent weighted closed itemset, Frequent weighted support, Weighted itemset-Tidset (WIT) trees

1. Introduction

Association rule mining (ARM) is used to identify relationships among items in transaction databases (Zaki 2004; Liu et al., 2012; Chen et al., 2013, 2014). 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-specified minimum support threshold (*minSup*). A traditional association rule is an expression of the form $\{X \rightarrow Y\ (sup, conf)\}$, where $X, Y \subseteq I$ and $X \cap Y = Ø$. 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*), the goal is to mine all association rules whose support and confidence values exceed *minSup* and *minConf*, respectively.

Traditional ARM does not take into consideration the relative benefit value of items. In some applications, the relative benefit (or weighted) value associated with each item is of interest. For example, the sale of bread may give a profit of 20 cents whereas that of a bottle of milk may give a profit of 40 cents. It is thus desirable to develop methods for applying ARM techniques to this kind of data. Ramkumar et al. (1998) (see also Cai et al., 1998) proposed a model for describing the concept of weighted association rules (WARs) and presented an Apriori-based algorithm for mining frequent weighted itemsets (FWIs). Since then, many WAR mining (WARM) techniques have been proposed (see for example Wang et al., 2000; Tao et al., 2003; Yun et al., 2012). Vo et al. (2013) proposed a number of WARM algorithms based on weighted itemset-Tidset (WIT) trees. The WIT tree data structure is adopted in the present study.

A major issue with FWI mining, as frequent itemset (FI) mining, is that a large number of rules are identified, many of which may be redundant. Frequent closed itemset (FCI) mining techniques have been proposed to solve this problem (Bastide et al., 2000; Duong, Truong, & Vo, 2014; Grahne & Zhu, 2005; Pasquier et al., 1999a,b; Pei, Han, & Mao, 2000; Singh, Singh, & Mahanta, 2005; Zaki, 2004; Zaki & Hsiao, 2005). Although many algorithms have been proposed for mining FCIs, there is only one algorithm for mining FWCIs (Vo et al., 2013). However, this algorithm is time-consuming when the minimum weighted support threshold is small. To overcome this issue, the present study develops two theorems for fast checking non-closed itemsets (Theorems 4.2 and 4.3). The Diffset strategy is used for reducing memory usage. A corollary is also developed to check non-closed itemsets when using Diffsets.

Our main contributions are as follows:

(1) Some theorems and corollary are proposed (Theorems 4.2 and 4.3, Corollary 4.1).

(2) Based on these theorems and corollary and WIT trees (Le et al., 2009; Le, Nguyen, & Vo, 2011; Vo, Coenen, & Le, 2013), an algorithm for fast mining FWCIs is proposed (Algorithm 1).

(3) The Diffset strategy (Zaki & Gouda, 2003) is extended for mining FWCIs (Algorithm 2).

(4) Some properties of WIT trees are exploited for fast mining FWCIs (Section 3).

The rest of this paper is organized as follows. Section 2 reviews work related to the mining of FWIs and WARs. Section 3 presents the proposed modified WIT tree data structure for compressing a database into a tree structure. Algorithms for mining FWCIs using WIT trees are described in Section 4. Some properties of WIT trees for fast mining FWCIs are also discussed. Experimental results are presented in Section 6, and conclusions are given in Section 7.

2. Related Work

This section briefly reviews some related works. A formal definition of weighted item transaction databases is given first. The Galois connection, used later in this paper to prove a number of theorems, is then introduced. Some definitions related to WARs are presented. Finally, some existing approaches for mining FCIs are discussed.

2.1. Weighted item transaction databases

A weighted item 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*.

Table 1. Transaction database

Transaction	**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. Item weights

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

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

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 database D. Let $\boldsymbol{P}(\boldsymbol{S})$ (the power set of S) include all subsets of $\boldsymbol{S}$. The following two mappings between $\boldsymbol{P}(\boldsymbol{I})$ and $\boldsymbol{P}(\boldsymbol{T})$ are called Galois connections (Zaki, 2004).

Let $X \subseteq I$ and $Y \subseteq T$. Then:

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 transactions Y.

Given $X, X_1, X_2 \in P(I)$, and $Y, Y_1, Y_2 \in P(T)$, the Galois connections satisfy the following properties (Zaki, 2004):

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 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 *X* 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 and 2 and Definition 2.1. $tw(t_1)$ can be computed as:

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

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

Table 3. Transaction weights for transactions in Table 1

Transaction	***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, *ws*(*BD*) can be computed as follows. Because *BD* appears in transactions {1, 3, 5, 6}, *ws*(*BD*) is computed as:

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

Mining FWIs requires the identification of all itemsets whose weighted supports satisfy the minimum weighted support threshold (*minws*), i.e., FWI = $\{X \subseteq I \mid ws(X) \geq minws\}$.

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. See Vo, Coenen, & Le (2013).

To mine WARs, all FWIs that satisfy the minimum weighted support threshold must be mined first. Mining FWIs is the most computationally expensive process of WARM. Ramkumar et al. (1998) proposed an Apriori-based algorithm for mining FWIs. This approach requires many scans of the whole database to determine the weighted supports of itemsets. Some other studies used this approach for generating WARs (Tao, Murtagh, & Farid, 2003; Wang, Yang, & Yu, 2000).

2.4. Mining frequent closed itemsets

FCIs are a variant of FIs that can be employed to reduce the overall number of generated rules. Formally, an itemset X is called an FCI if it is frequent, and there does not exist any FI Y such that $X \subset Y$ and $\sigma(X) = \sigma(Y)$. Many methods have been proposed for mining FCIs. They can be divided into the following four categories (Lee et al., 2008; Vo, Hong, & Le, 2013):

i) **Generate-and-test approaches**: These are methods based on the Apriori algorithm that use a level-wise approach to discover FCIs. Example algorithms include Close (Pasquier et al., 1999b) and A-Close (Pasquier et al., 1999a).

ii) **Divide-and-conquer approaches**: These are methods that adopt a divide-and-conquer strategy and use compact data structures extended from the frequent pattern (FP) tree to mine FCIs. Example algorithms include Closet (Pei, Han, & Mao, 2000), Closet+ (Wang, Han, & Pei, 2003), and FPClose (Grahne & Zhu, 2005).

iii) **Hybrid approaches**: These are methods that integrate the above two strategies to mine FCIs. These methods first transform the data into a vertical data format. Example hybrid methods include CHARM (Zaki & Hsiao, 2005) and CloseMiner (Singh, Singh, & Mahanta, 2005).

iv) **Hybrid approaches without duplication**: These methods differ from hybrid methods in that they do not use a subsumption-checking technique, and thus identified FCIs need not be stored in main memory. These methods also do not use a hash table. Example algorithms include DCI-Close (Lucchese, Orlando, & Perego, 2006), LCM (Uno et al., 2004), PGMiner (Moonestinghe, Fodeh, & Tan, 2006), and DBV-Miner (Vo, Hong, & Le, 2012).

2.5. Mining frequent (closed) high-utility itemsets

Mining high-utility itemsets (HUIs) is another important topic in data mining. It refers to discovering sets of items that not only co-occur but also carry high utilities (e.g., high profits). HUI mining has a variety of applications. Mining HUIs is not as easy as mining FIs due to the absence of the downward closure property (Liu et al., 2005; Wu et al., 2015). Several algorithms has been proposed for mining HUIs, such as Two-Phase (Liu et al., 2005), IHUP (Ahmed et al., 2009), UP-Growth (Tseng et al., 2010), and UP-Growth+ (Tseng et al., 2013). Existing methods for mining HUIs often present a large number of HUIs to users, causing mining tasks to suffer from long execution time and huge memory consumption. Moreover, a large number of HUIs is difficult to be utilized by users. To address this problem, closed HUIs were proposed as a compact and lossless representation of HUIs (Wu et al., 2015, Tseng et al., 2015).

2.6. Differences between FWI mining and (closed) HUI mining

Mining FWI/FWCI considers item weights and uses the average of weights to compute transaction weight (*ws*) while mining (closed) HUI considers total benefit of items. For example, when we are interested in weights of items (e.g., weight of a webpage in a website, weight of a word in a document, etc.), we use FWI/FWCI mining. When we consider benefit of selling products, we use (closed) HUI mining. These two research areas have different goals; therefore, their approaches and definitions are also different. Recently, several algorithms have been proposed for mining closed HUI (Wu et al., 2015, Tseng et al., 2015) but there is no method developed for mining FWCI.

3. WIT Tree Data Structure

Le et al. (2009) proposed the WIT tree data structure, an expansion of the IT tree proposed by Zaki et al. (1997), for mining HUIs. The WIT tree data structure represents the input data as itemset TID lists, which allows the fast computation of weighted support values. Each node in a WIT tree includes three fields:

i. *X*: an itemset.
ii. *t*(*X*): the set of transactions that contain *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*), to which their *tids* belong, and then dividing this by the sum of all *tw* values. Thus, computing *ws* is based on Tidsets. Links connect nodes at the k^{th} level (called *X*) with nodes at the $(k+1)^{th}$ level (called *Y*).

Definition 3.1 (Zaki & Hsiao, 2005) – Equivalence class

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

The set of all itemsets with a given prefix *X* is called an equivalence class, denoted as [*X*].

Example 3.1: Consider Tables 1 and 3 above. The associated WIT tree for mining FWIs is shown in Figure 1.

The root node of the WIT tree contains all 1-itemset nodes. All nodes in level 1 belong to a given equivalence class with prefix {} (or [∅]). Each node in level 1 will become a new equivalence class using its item as the prefix. Each node with the same prefix will join with all nodes following it to create a new equivalence class. The process proceeds recursively to create new equivalence classes in higher levels.

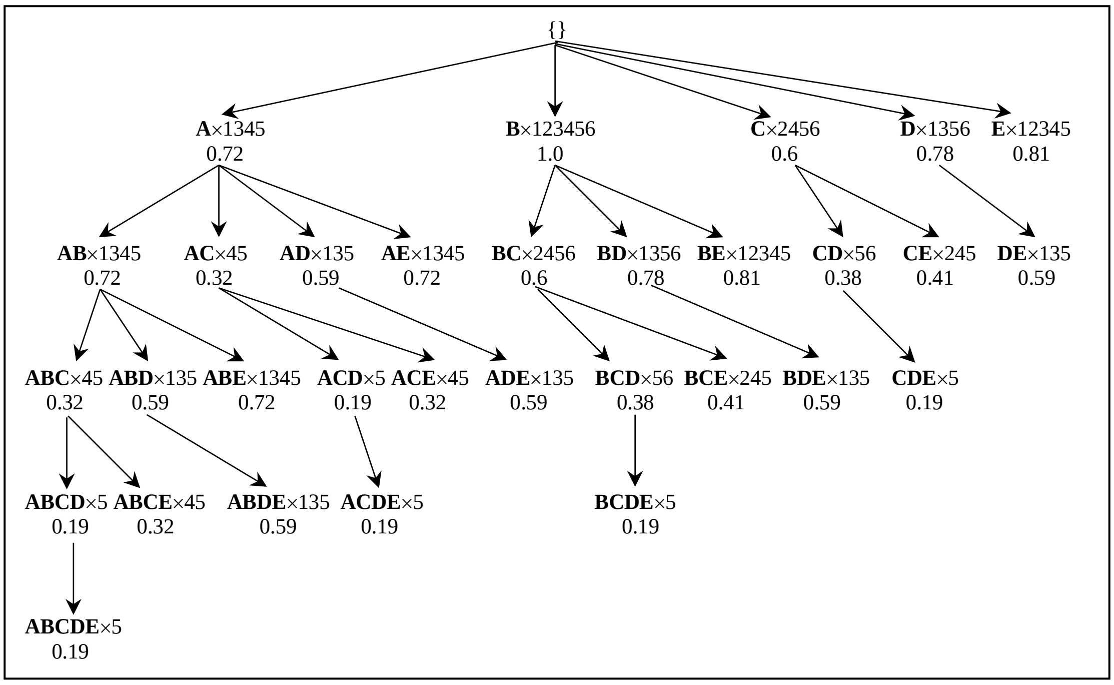

Figure 1. Search tree based on WIT tree structure.

For example, considering Figure 1, nodes $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, and $\{E\}$ belong to equivalence class [∅]. Consider node $\{A\}$. This node will join with all nodes following it ($\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$) to create new equivalence class $[A] = \{\{AB\}, \{AC\}, \{AD\}, \{AE\}\}$. $[AB]$ will become a new equivalence class by joining with all nodes following it ($\{AC\}$, $\{AD\}$, $\{AE\}$), and so on.

An inspection of Figure 1 indicates that all itemsets satisfy the downward closure property. Thus, an equivalence class in the WIT tree can be pruned if its *ws* value does not satisfy *minws*.

For example, suppose that *minwus* = 0.4. Because *ws*(*ABC*) = 0.32 < *minws*, the equivalence class with the prefix *ABC* can be pruned, i.e., all child nodes of *ABC* can be pruned.

4. Mining Frequent Weighted Closed Itemsets

Definition 4.1: Let $X \subseteq I$ be an FWI. *X* is called an FWCI if and only if there does not exist FWI *Y* such that $X \subset Y$ and *ws*(*X*) = *ws*(*Y*).

From Definition 4.1, there are a lot of FWIs that are not closed. For example, *A*, *AB*, and *AE* are not closed because *ABE* has the same *ws* value. The mining of FWCIs from weighted item transaction databases is described below.

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

Proof. See Vo et al. (2013).

Theorem 4.2. Let $\underset{ws(X)}{X \times t(X)}$ and $\underset{ws(Y)}{Y \times t(Y)}$ be two nodes in the equivalence class [*P*]. Then:

i) If $t(X) = t(Y)$, then *X* and *Y* are not closed.

ii) If $t(X) \subset t(Y)$, then *X* is not closed.

iii) If $t(X) \supset t(Y)$, then *Y* is not closed.

Proof:

i) We have $t(X \cup Y) = t(X) \cap t(Y) = t(X) = t(Y)$ (because $t(X) = t(Y)$) $\Rightarrow$ according to Theorem 4.1, we have $ws(X) = ws(Y) = ws(X \cup Y) \Rightarrow$ *X* and *Y* are not closed.

ii) We have $t(X \cup Y) = t(X) \cap t(Y) = t(X)$ (because $t(X) \subset t(Y)$) $\Rightarrow$ according to Theorem 4.1, we have $ws(X) = ws(X \cup Y) \Rightarrow X$ is not closed.

iii) We have $t(X \cup Y) = t(X) \cap t(Y) = t(Y)$ (because $t(X) \supset t(Y)$) $\Rightarrow$ according to Theorem 4.1, we have $ws(Y) = ws(X \cup Y) \Rightarrow Y$ is not closed.

When the nodes in equivalence class P are sorted in increasing order according to the cardinality of Tidsets, condition *iii*) (of Theorem 4.2) will not occur, so only conditions *i*) and *ii*) are considered below.

In the process of mining FWCIs, considering nodes in a given equivalence class consumes a lot of time. Thus, nodes that satisfy condition *i*) at level 1 of the WIT tree are grouped. In the process of creating a new equivalence class, nodes that satisfy condition *i*) are also grouped. This reduces the cardinality of the equivalence class, and thus significantly decreases mining time. This approach differs from Zaki's approach (Zaki & Hsiao, 2005) in that it significantly decreases the number of nodes that need to be considered, and it need not remove the nodes that satisfy condition i in an equivalence class.

Theorem 4.3. Suppose that itemset $l_i \cup l_j$ is created from nodes l_i and l_j in equivalence class $[P]$ ($i < j$). If one of the two following conditions occurs, then $l_i \cup l_j$ is not a closed itemset:

i) There exists node $\underset{ws(l_k)}{l_k \times t(l_k)}$ (k < i) in [P], so that [l_k] contains the child node $\underset{ws(Z)}{Z \times t(Z)}$ that satisfies $t(l_i \cup l_j) = t(Z)$ or

ii) There exists node $\underset{ws(l_k)}{l_k \times t(l_k)}$ (k < i) in [P], so that [l_k] contains the grandchild node $\underset{ws(Z)}{Z \times t(Z)}$ that satisfies $t(l_i \cup l_j) = t(Z)$

Proof: Consider the process of generating node $\underset{ws(l_k)}{l_k \times t(l_k)}$ in the WIT-FWCI algorithm (Figure 2). $\underset{ws(l_k)}{l_k \times t(l_k)}$ will join $\underset{ws(l_i)}{l_i \times t(l_i)}$:

i) if $t(l_k \cup l_i) \subset t(l_k \cup l_j)$, according to Theorem 4.1 (ii), $l_k \cup l_i$ will not be closed and the algorithm will replace $l_k \cup l_i$ by $l_k \cup l_i \cup l_j$. Therefore, if $Z = l_k \cup l_i \cup l_j$ and $t(l_i \cup l_j) = t(Z)$, $l_i \cup l_j$ is not closed.

ii) if $t(l_k \cup l_i) \not\subset t(l_k \cup l_j)$, when $l_k \cup l_i$ joins $l_k \cup l_j$ to create a new itemset $l_k \cup l_i \cup l_j$, if $Z = l_k \cup l_i \cup l_j$ and $t(l_i \cup l_j) = t(Z)$, then $l_i \cup l_j$ is not closed.

Node $l_i \cup l_j$ is called a subsumed node.

4.1. Algorithm for mining FWCIs

```
Algorithm 1: WIT-FWCI algorithm for mining FWCIs
Input: Database $D$ and $minws$
Output: Set FWCI that contains all FWCIs that satisfy $minws$ from $D$
Method:
WIT-FWCI()
1. $[\varnothing] = \{i \in I: ws(i) \geq minws\}$
2. FWCI = $\varnothing$
3. SORT($[\varnothing]$)
4. GROUP($[\varnothing]$)
5. FWCI-EXTEND ($[\varnothing]$)

FWCI-EXTEND($[P]$)
6. for all $l_i \in [P]$ do
7.     $[P_i] = \varnothing$
8.     for all $l_j \in [P]$, with j > i do
9.         if $t(l_i) \subset t(l_j)$ then
10.            $l_i = l_i \cup l_j$
11.        else
12.         $X = l_i \cup l_j$
13.            $Y = t(l_i) \cap t(l_j)$
14.            $ws(X)$ = COMPUTE-WS ($Y$)          // use Eq. (2.2)
15.            if $ws(X) \geq minws$ then
16.                if $X \times Y$ is not subsumed then  // use Theorem 4.3.
17.                    Add $\{ \underset{ws(X)}{X \times Y} \}$ to $[P_i]$          // sort in increasing order by $|Y|$
```

Figure 2. WIT-FWCI algorithm for mining FWCIs.

The WIT-FWCI algorithm (see Figure 2) commences with an empty equivalence class that contains single items with their *ws* values satisfying *minws* (line 1). The algorithm then sorts nodes in equivalence class $[\varnothing]$ in increasing order according to the cardinality of Tidsets (line 3). It then groups all nodes that have the same tids into a unique node (line 4), and calls the procedure **FWCI-EXTEND** with parameter [∅] (line 5). Procedure **FWCI-EXTEND** uses equivalence class [*P*] as an input value. It considers each node in equivalence class [*P*] with equivalence classes following it (lines 6 and 8). With each pair l_i and l_j, the algorithm considers condition ii of Theorem 4.2. If it is satisfied (line 9), the algorithm replaces equivalence class [l_i] by [$l_i \cup l_j$] (line 10); otherwise, the algorithm creates a new node and adds it into equivalence class [P_i] (initially it is assigned as an empty value, line 7). According to Theorem 4.3, when the Tidset of *X* (i.e., *Y*) is identified, it is checked whether it is subsumed by another node (line 16); if not, then it is added into [*P*]. Adding node $\underset{ws(X)}{X \times Y}$ into [P_i] is performed similarly to level 1 (i.e., it is considered with nodes in [P_i]; nodes with the same Tidset are grouped, line 17). After l_i is considered with all nodes following it, the algorithm adds l_i and its *ws* into **FWCI** (line 18). Finally, the algorithm is called recursively to generate equivalence classes after [l_i] (line 19).

Two nodes in the same equivalence class do not satisfy condition i of Theorem 4.2 because the algorithm groups these nodes into one node whenever they are added into [*P*]. Similarly, condition iii does not occur because the nodes in equivalence class [*P*] are sorted in increasing order of the cardinality of Tidsets.

4.2. Illustration of WIT-FWCI

Using the example data presented in Tables 1 and 3, the WIT-FWCI algorithm with *minws* = 0.4 is illustrated. First of all, [∅] = {*A*, *B*, *C*, *D*, *E*}. After sorting and grouping, the result is [∅] = {*C*, *D*, *A*, *E*, *B*}. The algorithm then calls the function **FWCI-EXTEND** with input nodes {*C*, *D*, *A*, *E*, *B*}.

With equivalence class [*C*]:

Consider *C* with *D*: we have a new itemset *CD*×56 with $ws(CD) = 0.38 < minws$.

Consider *C* with *A*: we have a new itemset $CA \times 45$ with $ws(CA) = 0.32 < minws$.

Consider *C* with *E*: we have a new itemset $CE \times 245$ with $ws(CE) = 0.41 \Rightarrow [C] = \{CE\}$.

Consider *C* with *B*: we have $t(C) \subset t(B)$ (satisfies condition ii of Theorem 4.2) $\Rightarrow$ Replace [*C*] by [*CB*]. This means that all equivalence classes following [*C*] replace *C* with *CB*. Therefore, [*CE*] is replaced by [*CBE*].

After making equivalence class [*C*] (becomes [*CB*]), *CB* is added to **FWCI** $\Rightarrow$ **FWCI** = $\{CB\}$. The algorithm is called recursively to create all equivalence classes following it.

Consider equivalence class $[CBE] \in [CB]$: add *CBE* to **FWCI** $\Rightarrow$ **FWCI** = $\{CB, CBE\}$.

With equivalence class [*D*]:

Consider *D* with *A*: we have a new itemset $DA \times 135$ with $ws(DA) = 0.59 \Rightarrow [D] = \{DA\}$.

Consider *D* with *E*: we have a new itemset $DE \times 135 \Rightarrow$ Group *DA* with *DE* into $DAE \Rightarrow [D] = \{DAE\}$.

Consider *D* with *B*: we have $t(D) \subset t(B)$ (satisfies condition *ii*) of Theorem 4.2) $\Rightarrow$ Replace [*D*] by [*DB*]. This means that all equivalence classes following [*D*] replace *D* with *DB*. Therefore, [*DAE*] is replaced by [*DBAE*].

After making equivalence class [*D*] (becomes [*DB*]), *DB* is added to **FWCI** $\Rightarrow$ **FWCI** = $\{CB, CBE, DB\}$. The algorithm is called recursively to create all equivalence classes following it.

Consider equivalence class $[DBAE] \in [DB]$: add *DBAE* to **FWCI** $\Rightarrow$ **FWCI** = $\{CB, CBE, DB, DBAE\}$.

This is similar to equivalence classes [*A*], [*E*], and [*B*]. **FWCI** = $\{CB, CBE, DB, DBAE, AEB, EB, B\}$.

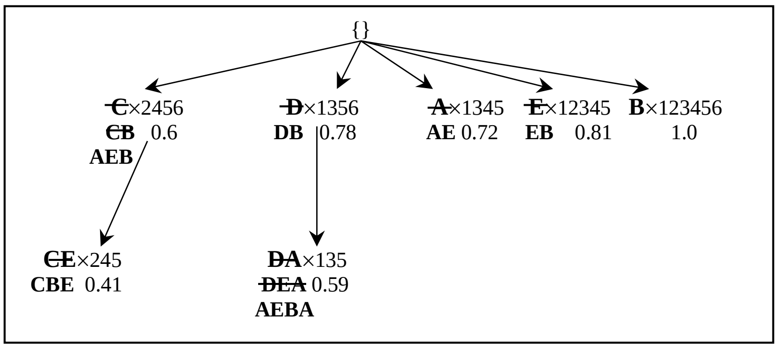

Figure 3. Results of WIT-FWCI for data in Tables 1 and 3 with *minws* = 0.4.

Figure 3 shows that there are fewer FWCIs than FWIs (7 vs. 19), and that there are fewer search levels in the tree created using WIT-FWCI than in that created using WIT-FWI (2 vs. 4). Thus, FWCI mining is more efficient than FWI mining.

4.3. Discussions

The WIT-FWCI algorithm has some improvements compared with the algorithm proposed by Zaki & Hsiao (2005). First, itemsets with the same tids are grouped, which reduces the number of nodes that need to be considered in an equivalence. Second, because of sorting according to increasing order of Tidset cardinality, we need not check condition iii of Theorem 4.2. Thus, the number of cases considered by the WIT-FWCI algorithm is lower than that considered by CHARM (Zaki & Hsiao, 2005).

When a new itemset is created from two nodes in a given equivalence class [*P*], it is checked whether it is closed using Theorem 4.3.

To illustrate the impact of Theorem 4.3, the database in Tables 1 and 3 with *minws* = 0.4 is considered. Assume that items in [∅] are sorted as follows: [∅] = {*A*, *C*, *D*, *E*, *B*}.

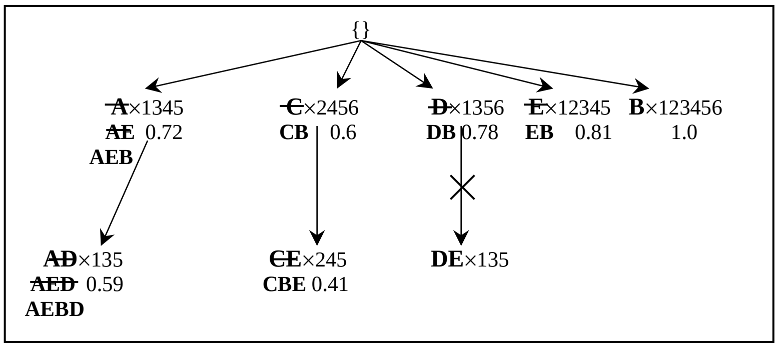

Figure 4. Effect of Theorem 4.3.

In Figure 4, when *A* is joined with all nodes following it, the result is node $\underset{0.59}{AEBD}\times 135$. *D* joins *E* to create $DE\times 135$. $t(DE) \subseteq t(AEBD) \Rightarrow DE$ is not closed and is consumed by *AEBD*, and thus not added to equivalence class [*D*].

4.4. Using Diffsets

Diffset is applied for fast computing *ws* and reducing memory when mining FWCI using the following corollary:

Corollary 4.1. If $d(PXY) = \varnothing$, then *PX* is not closed.

Proof: Because $d(PXY) = \varnothing$, according to Theorem 4.2, $ws(PX) = ws(PXY) \Rightarrow PX$ is not closed according to Definition 4.1.

a) Algorithm

The algorithm in Figure 5 differs from that in Figure 3 in that it substitutes Tidsets with Diffsets. It uses Corollary 4.1 to check condition *ii*) of Theorem 4.2. If $d(l_i \cup l_j) = \varnothing$, then l_i is not closed (line 13).

Algorithm 2: WIT-FWCI with Diffsets

Input: Database *D* and *minws*

Output: Set **FWCI** that contains all FWCIs that satisfy *minws* from *D*

Method:

```
WIT-FWCI-DIFF()
    1. $[\varnothing] = \{i \in I: ws(i) \geq minws\}$
    2. FWCI = $\varnothing$
    3. SORT($[\varnothing]$)
    4. GROUP($[\varnothing]$)
    5. FWCI-EXTEND-DIFF($[\varnothing]$)

FWCI-EXTEND-DIFF($[P]$)
    6. for all $l_i \in [P]$ do
    7.     $[P_i] = \varnothing$
    8.     for all $l_j \in [P]$, with j > i do
    9.     if $P = \varnothing$ then
    10.        $Y = t(l_i) \setminus t(l_j)$
    11.    else
    12.        $Y = d(l_j) \setminus d(l_i)$
    13.    if $Y = \varnothing$ then
    14.        $l_i = l_i \cup l_j$
    15.    else
    16.        $X = l_i \cup l_j$
    17.        $ws(X)$ = COMPUTE-WS($Y$)
    16.        if $ws(X) \geq minws$ then
    17.            if $X \times Y$ is not subsumed then  // use Theorem 4.3 and Corollary 4.1
    18.                Add $\{ \underset{ws(X)}{X \times Y} \}$ to $[P_i]$          // sort in increasing order by $|Y|$
```

Figure 5. Algorithm that uses Diffsets for mining FWCIs.

b) Illustration

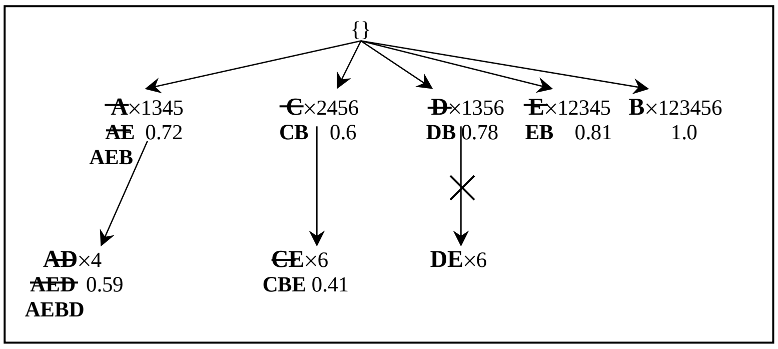

Figure 6. Mining FWCIs using Diffsets.

Figure 6 uses Diffsets for fast computing the *ws* values of itemsets. The results are the same as those obtained using Tidsets, but the memory consumed and mining time are lower.

5. Experimental Results

All experiments described below were performed on a computer with a Centrino Core 2 Duo (2 × 2.53 GHz) CPU and 4 GB of RAM running Windows 7. The algorithms were implemented using C# 2008. The datasets used in the experiments were downloaded from http://fimi.cs.helsinki.fi/data/. Some statistical information regarding these datasets is given in Table 4.

A table was added to each database to store the weighted values of items (in the range of 1 to 10).

Table 4. Experimental databases

Database	# of trans	# of items	Average length
BMS-POS	515597	1656	6.53
Connect	67557	130	43
Accidents	340183	468	33.81
Chess	3196	76	37
Mushroom	8124	120	23

Table 5. Numbers of FWCIs obtained from experimental databases

Database	***minws* (%)**	**# of FWCIs**
BMS-POS	10	12
	8	21
	6	32
	4	85
Chess	85	1894
	80	5125
	75	11724
	70	24604
Mushroom	35	252
	30	423
	25	696
	20	1202
Connect	96	513
	94	1284
	92	2286
	90	3443
Accidents	95	15
	85	65
	75	289
	65	1035
	0.01	14398

Table 5 shows the numbers of FWCIs obtained from the experimental databases. The number of FWCIs obtained from a database is often smaller than the number of FWIs. For example, consider the Chess database with *minws* = 70%. The number of FWCIs is 24604 and the number of FWIs is

47181. The ratio is $\frac{24604}{47181}\times 100\% = 52.15\%$. Therefore, mining rules from FWCIs is more efficient than from FWIs.

Currently, there are no other approaches (of other authors) for mining FWCIs. Therefore, in this paper, the mining times obtained using the Tidsets and Diffsets methods are compared. Figures 7-11 show the results. In general, the WIT-FWCI-Diff algorithm (using Diffsets concept) is more efficient than the WIT-FWCI algorithm (using Tidsets concept) in terms of mining time.

For the BMS-POS database (Figure 7), the number of FWCIs is small (see Table 5). Therefore, there is no significant difference between WIT-FWCI-Diff and WIT-FWCI.

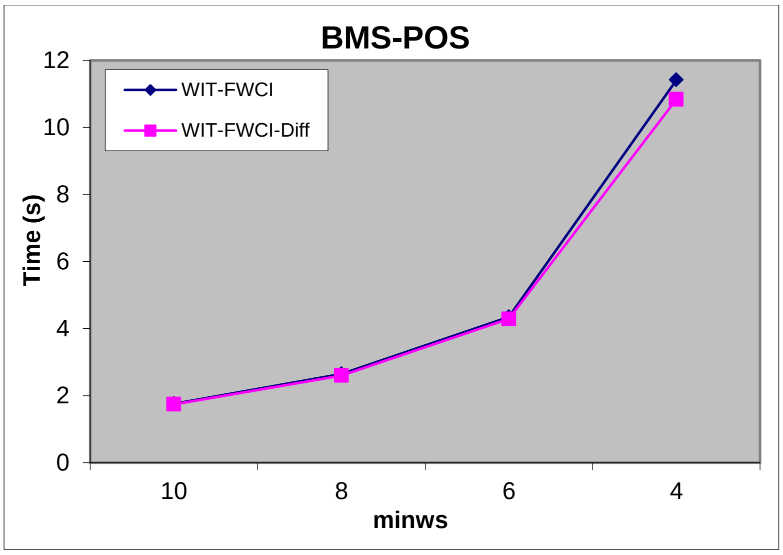

Figure 7. Mining times of WIT-FWCI with Tidsets and Diffsets for BMS-POS.

For the Chess database (Figure 8), the time gap between WIT-FWCI-Diff and WIT-FWCI (Δt) increased sharply from 0.6 to 6.69 s when *minws* was decreased from 85% to 70%.

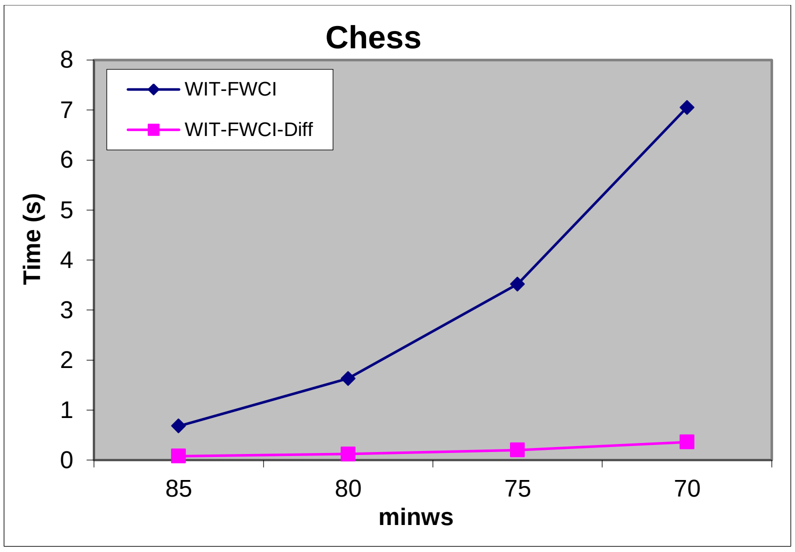

Figure 8. Mining times of WIT-FWCI with Tidsets and Diffsets for Chess.

For the Mushroom database (Figure 9), Δt increased slowly from 0.14 to 0.51 s when *minws* was decreased from 35% to 20%.

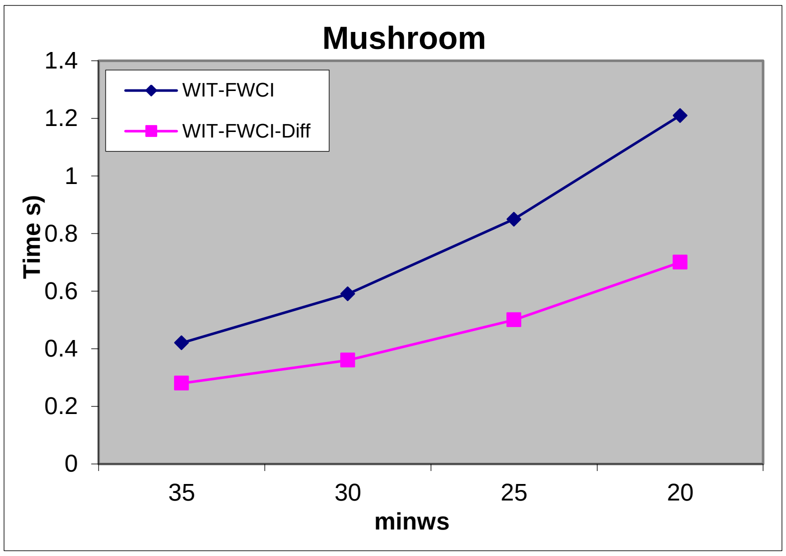

Figure 9. Mining times of WIT-FWCI with Tidsets and Diffsets for Mushroom.

For the Connect and Accidents databases (Figures 10 and 11), Δt was large and increased sharply. In conclusion, using the Diffset concept is better than using the Tidset concept in terms of mining time in most cases.

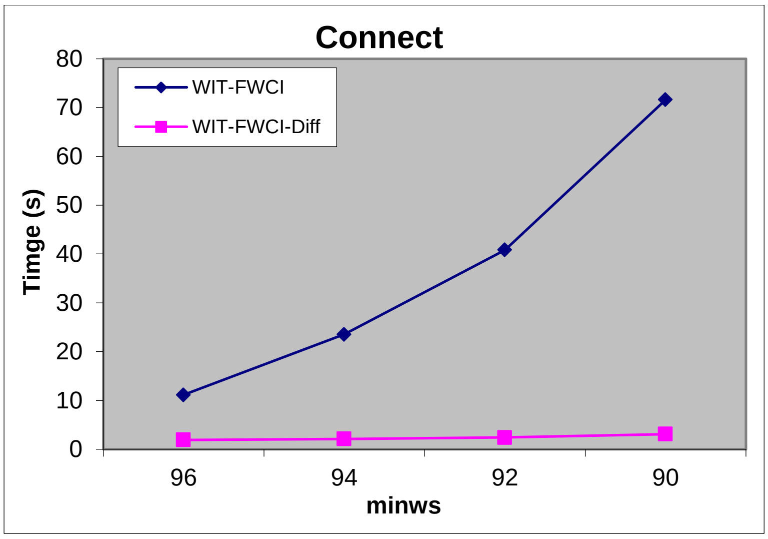

Figure 10. Mining times of WIT-FWCI with Tidsets and Diffsets for Connect.

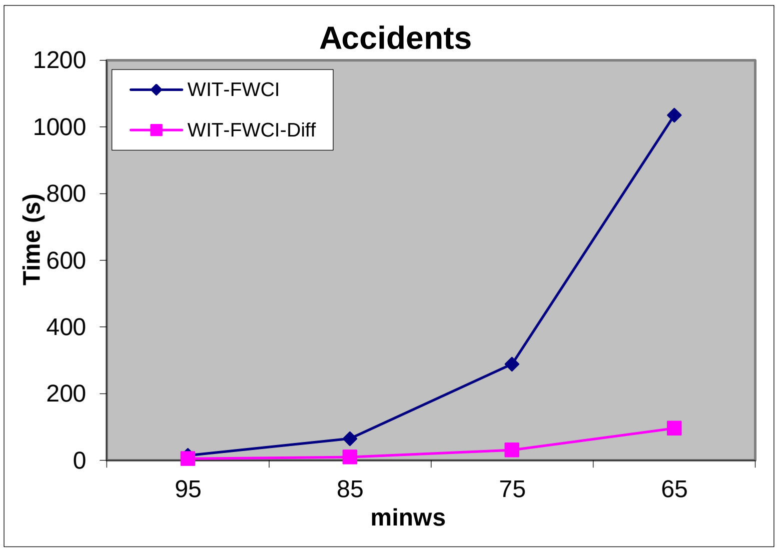

Figure 11. Mining times of WIT-FWCI with Tidsets and Diffsets for Accidents.

Figures 12 to 16 show a comparison of memory usage for WIT-FWCI and WIT-FWCI-Diff. The memory usage of WIT-FWCI-Diff is more efficient than that of WIT-FWCI in most cases. For BMS-POS, because the number of FWCIs is small, the difference between the two algorithms is small; however, when the number of FWCIs is large, the gap becomes large. For example, consider the Chess database with *minws* = 70%. The number of FWCIs is 24604, the memory usage of WIT-FWCI is 216.13 MB, and that of WIT-FWCI-Diff is 1.3 MB.

The average length of the database affects the mining time and memory usage. The average length of BMS-POS is small so the gap between the two algorithms is small. With Mushroom, the average length is medium, so the gap is wider; with Chess, Connect, and Accidents, the gap is very large.

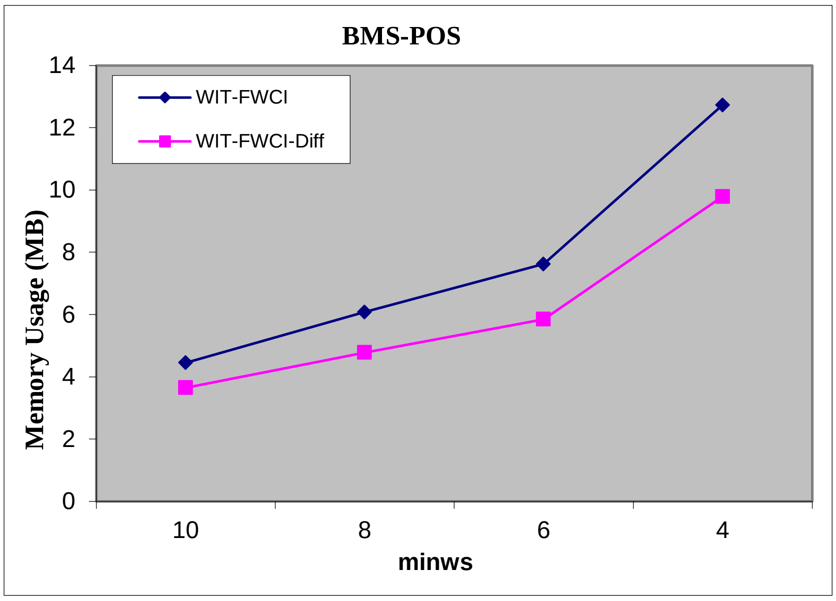

Figure 12. Memory usage of WIT-FWCI with Tidsets and Diffsets for BMS-POS.

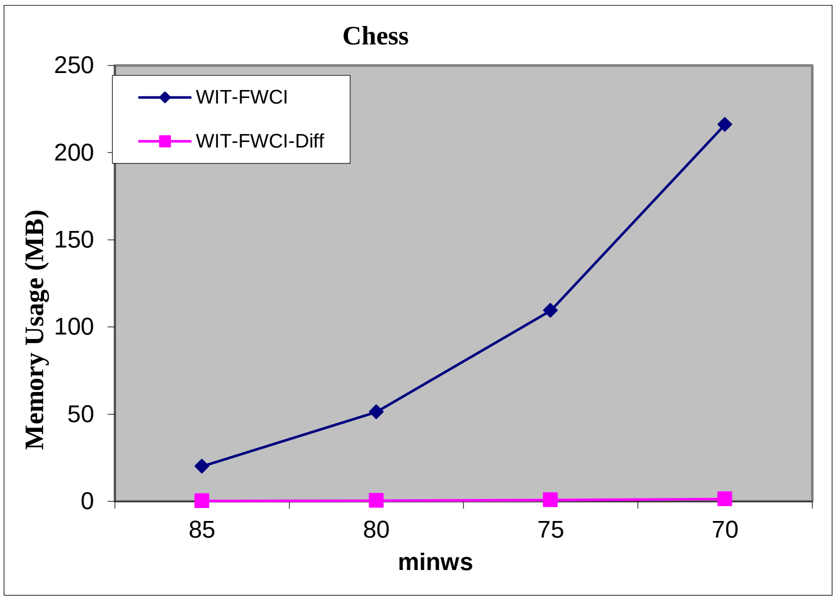

Figure 13. Memory usage of WIT-FWCI with Tidsets and Diffsets for Chess.

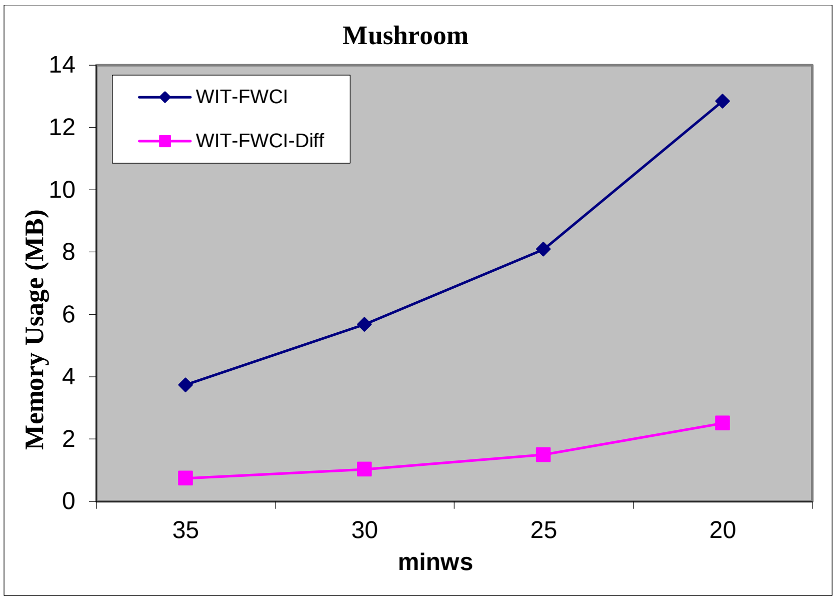

Figure 14. Memory usage of WIT-FWCI with Tidsets and Diffsets for Mushroom.

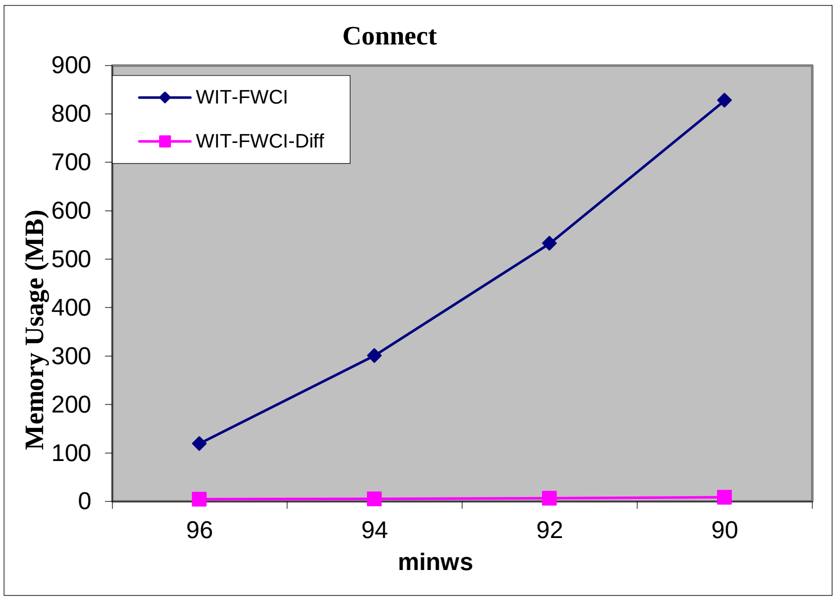

Figure 15. Memory usage of WIT-FWCI with Tidsets and Diffsets for Connect.

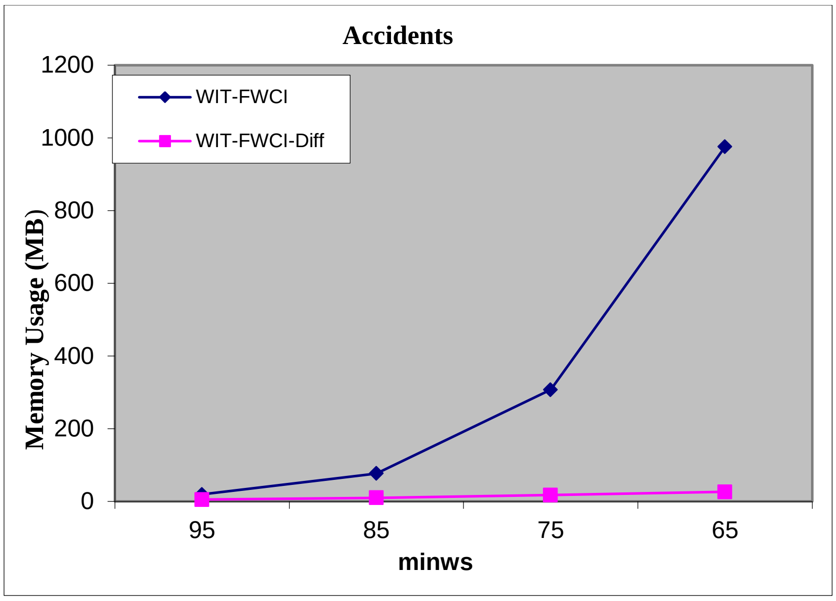

Figure 16. Memory usage of WIT-FWCI with Tidsets and Diffsets for Accidents.

6. Conclusion and future work

This paper proposed a method for mining FWCIs from weighted item transaction databases. Using the WIT tree data structure, the algorithm only scans the database once. The proposed algorithm is faster when using Diffsets than when using Tidsets because the former reduces memory use, and thus computation time.

In this paper, we only improve the phase of mining FWCIs using the WIT tree structure. In the future, we will study how to efficiently mine association rules from FWCIs. We will apply the proposed method to the mining of weighted utility association rules. How to mine FWIs/FWCIs from incremental databases will also be considered.

Acknowledgment

This work was supported by Vietnam's National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.05-2015.10.

REFERENCES

Ahmed C. F., Tanbeer S. K., Jeong B.-S., and Lee Y.-K. (2009). Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering, 21(12), pp. 1708-172

Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., Lakhal, L. (2000). Mining minimal non-redundant association rules using frequent closed itemsets. In: 1st International Conference on Computational Logic, pp. 972 – 986

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

Chen C.H., Li A.F., Lee Y.C. (2014). Actionable high-coherent-utility fuzzy itemset mining. Soft Computing, 18(12), 2413-2424

Chen C.H., Li A.F., Lee Y.C. (2013). A fuzzy coherent rule mining algorithm. Applied Soft Computing, 13(7), 3422-3428Cheng-Wei W., Philippe F.-V., Jia-Yuan G., Vincent S. T. (2015). Mining Closed+ High Utility Itemsets without Candidate Generation. In: Proceedings of Conference on Technologies and Applications of Artificial Intelligence (TAAI 2015)

Duong, H., Truong, T., Vo, B. (2014). An efficient method for mining frequent itemsets with double constraints. In: Engineering Applications of Artificial Intelligence 27, pp. 148-154

Grahne, G., Zhu, J. (2005). Fast algorithms for frequent itemset mining using FP-trees. In: IEEE Transaction on Knowledge and Data Engineering 17(10), pp. 1347–1362

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

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

Lee, A. J. T., Wang, C. S., Weng, W. Y., Chen, J. A., Wu, H. W. (2008). An efficient algorithm for mining closed inter-transaction itemsets. In: Data & Knowledge Engineering 66(1), pp. 68 – 91

Liu X.B., Zhai K., Pedrycz W. (2012). An improved association rules mining method. Expert Systems with Applications 39(1), 1362-1374

Lucchese, B., Orlando, S., Perego, R. (2006). Fast and memory efficient mining of frequent closed itemsets. In: IEEE Transaction on Knowledge and data Engineering 18(1), pp. 21 – 36

Moonestinghe, H.D.K., Fodeh, S., Tan, P.N. (2006). Frequent closed itemsets mining using prefix graphs with an efficient flow-based pruning strategy. In: Proceedings of 6th ICDM, Hong Kong, pp. 426 – 435

Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L. (1999a). Discovering frequent closed itemsets for association rules. In: Proc. of the 5th International Conference on Database Theory, LNCS, Springer-Verlag, Jerusalem, Israel, pp. 398 – 416

Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L. (1999b). Efficient mining of association rules using closed itemset lattices. In: Information Systems 24(1), pp. 25 – 46

Pei, J., Han, J., Mao, R. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. In: Proc. of the 5th ACM-SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, USA, pp. 11 – 20

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

Singh, N. G., Singh, S. R., Mahanta, A.K. (2005). CloseMiner: Discovering frequent closed itemsets using frequent closed tidsets. In: Proc. of the 5th ICDM, Washington DC, USA, pp. 633 – 636

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

Tseng V. S., Wu C.-W., Shie B.-E., and Yu P. S. (2010). UP-Growth: an efficient algorithm for high utility itemset mining. In: Proceeding of ACM SIGKDD, pp. 253–262.

Tseng V. S., Shie B.-E., Wu C.-W., and Yu P. S. (2013). Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Transactions on Knowledge and Data Engineering 25(8), pp. 1772-1786

Tseng V. S., Wu C.-W., Philippe F.-V., and Yu P. S. (2015). Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets. IEEE Transactions on Knowledge and Data Engineering 27(3), pp. 726-739

Uno, T., Asai, T., Uchida, Y., Arimura, H. (2004). An efficient algorithm for enumerating closed patterns in transaction databases. In: Proc. of the 7th International Conference on Discovery Science, LNCS, Springer, Padova, Italy, pp. 16 – 31

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

Vo, B., Coenen, F., Le, B. (2013). A new method for mining frequent weighted itemsets based on WIT-trees. In: Expert Systems with Applications 40(4), pp. 1256 – 1264

Vo, B., Tran, N.Y., Ngo, D.H. (2013). Mining frequent weighted closed itemsets. In: Advanced Computational Methods for Knowledge Engineering, pp. 379-390

Vo, B., Hong, T.P., Le, B. (2013). A lattice-based approach for mining most generalization association rules. In: Knowledge-Based Systems 45, pp. 20–30

Yun, U., Shin, H., Ryu, K.H., Yoon, E. (2012). An efficient mining algorithm for maximal weighted frequent patterns in transactional databases. In: Knowledge-Based Systems 33, pp. 53–64.

Y. Liu, W. Liao, and A. Choudhary (2005). A fast high utility itemsets mining algorithm. In: Procceeding of the Utility-Based Data Mining Workshop, pp. 90-99

Wang, J., Han, J. Pei, J. (2003). CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 236 – 245

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

Zaki, M. J., Parthasarathy, S., Ogihara, M., Li, W. (1997). New algorithms for fast discovery of association rules. In: 3rd International Conference on Knowledge Discovery and Data Mining (KDD), pp. 283–286

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

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

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