# Mining of Frequent Itemsets with JoinFI-Mine Algorithm

SUPATRA SAHAPHONG[1], GUMPON SRITANRATANA[2]
[1]Department of Computer Science, Faculty of Science,
Ramkhamhaeng University,
Ramkhamhaeng Road, Bangkapi District, Bangkok 10240,
THAILAND
supatra@ru.ac.th
[2]Department of Mathematics, Faculty of Science,
Mahidol University,
Rama VI Road, Ratchathewi District, Bangkok 10400,
THAILAND
scgst@mahidol.ac.th

*Abstract*: - Association rule mining among frequent items has been widely studied in data mining field. Many researches have improved the algorithm for generation of all the frequent itemsets. In this paper, we proposed a new algorithm to mine all frequents itemsets from a transaction database. The main features of this paper are: (1) the database is scanned only one time to mine frequent itemsets; (2) the new algorithm called the *JoinFI-Mine* algorithm which use mathematics properties to reduces huge of subsequence mining; (3) the proposed algorithm mines frequent itemsets without generation of candidate sets; and (4) when the minimum support threshold is changed, the database is not require to scan. We have provided definitions, algorithms, examples, theorem, and correctness proving of the algorithm.

*Key-Words*: - Algorithm, association rule mining, database, data mining, frequent itemsets mining, frequent pattern mining, knowledge discovery

## 1 Introduction

Nowadays there are tremendous amounts of data which have been collected and stored in large databases, and which cannot be analyzed using manual systems. For example, how can one find the items which occur together from large amounts of data?, how do the items in a massive data set relate to the others?, what items in the data set frequently appear?. Therefore, powerful data analysis is necessary to solve these requirements. One solution is to use frequent itemset mining. Frequent itemsets mining is an essential step in association rule mining. The association rule mining algorithm is to decompose into two major subtasks:

- The generation of all the frequent itemsets that satisfy the minimal support threshold.
- The extraction of all high confidence rules from frequent itemsets found in previous step.

Our work focuses on the mining of frequent itemsets. The first classic algorithm is *Apriori* which is proposed in [1]. The *Apriori* principle is “If an itemsets is frequent, then all of its subsets must also be frequent” [2]. The *Apriori* algorithm uses a level-wise and breadth-first search approach for generating association rule. It uses the support-based pruning to control the exponential growth of candidate itemsets. However, the algorithms based on generated and tested candidate itemsets have two major drawbacks:

- The database must be scanned multiple times to generate candidate sets. Multiple scans will increase the I/O load and is time-consuming.
- The generation of huge candidate sets and calculation of their support will consume a lot of CPU time.

The drawbacks which presented as above were overcome by using the next generation of algorithm, called the *FP-growth* algorithm [3]. The advantages of mining of frequent itemsets by using the *FP-growth* algorithm such as: First, the database is scanned only two times. Next, the generating of candidate sets is not required. The *FP-growth* algorithm performs depth-first search approach in the search space. It encodes the data set using a compact data structure called *FP-tree* and extracts frequent pattern directly from this prefix tree [4]. The following researches have improved this idea. In reference [5], the *H-mine* algorithm was introduced by using array-based and trie-based data structure. The *Patricia Mine* algorithm was proposed in [6] that compressed *Patricia trie* to store the data sets. The *FPgrowt** algorithm reduced the *FP-tree* traversal time by using array technique [7]. In reference [8], the *SFI-Mine* algorithm which constructs pattern-base by using a new method which is different from pattern-base in *FP-growth* and mines frequent itemsets with a new combination method without recursive construction of conditional *FP-tree*. However, most of the *FP-tree* algorithm base has the following drawbacks:

- Mining of frequent itemset from the *FP-tree*, it generates huge of conditional FP-tree and takes a lot of time and space.
- When the changing of minimum support, this algorithm may restart and scan database twice.

Many researchers have proposed ways to scan database once. The *Eclat* algorithm was proposed by using the join step from the *Apriori* property to generate frequent pattern [9]. In Reference [10], the new data structure, called *LIB-graph* is proposed to contain data when database is scanned and discovery of frequent patterns by using recursive conditional *FP-tree*. The *Sorted-List* structure which created from the *Vertical Index List* was proposed to contained data from scanning database once and mining of frequent itemsets by using *depth-first* search [11].

In this paper, we proposed the algorithm, called *JoinFI-Mine* algorithm. The main advantages of our method are presented as follows:

- The database is scanned only one time to mine frequent itemsets.
- The *JoinFI-Mine* algorithm mines frequent itemsets without generation of candidate sets. The results of this method are still obtaining complete and correct frequent itemset.
- The rescanning of the database is not required, if decision maker want to change of the minimum support threshold.

This paper is organized as follows. The prior knowledge is presented in section 2, followed by the approach which is presented in section 3, the correctness proof is shown in section 4 and the finally, the conclusion is addressed in section 5.

# 2 Prior Knowledge

## 2.1 Basic Definition

This subsection introduces basic concepts for mining of frequent itemsets. All definitions in this subsection are proposed by J. Han et al in [4, 12], as follows.

**Definition 2.1**

Let $I=\{x_1,x_2,\ldots, x_m\}$ be a set of items and a *transaction database* $DB=\{T_1,T_2,\ldots,T_n\}$, where $T_i(i\in[1..n])$ is a transaction which contains items in $I$.

**Definition 2.2**

The *support or supp* (or occurrence frequency) of a pattern $A$, where $A$ is a set of items, is the number of transactions containing $A$ in $DB$. A pattern $A$ is *frequent* if $A$'s support is no less than a predefined minimum support threshold, *minsup*.

**Definition 2.3**

An item $x$ is called a *frequent item* if $supp(x)\geq minsup$, otherwise it is called an *infrequent item*.

Given a transaction database $DB$ and a minimum support threshold *minsup*, the problem of *finding the complete set of frequent itemsets* is called the *frequent itemset mining problem* and any element of the complete set is called a *frequent itemset*. For greater understanding, we provide an example to describe the above definitions.

**Example 1**. The Fig.1 is a $DB$. It consists of five transactions $T_1$, $T_2$, $T_3$, $T_4$, and $T_5$ labelled as *Transactions* in the Fig.1, and seventeen items $i_1$, $i_2$, $i_3$, $i_4$, $i_5$, $i_6$, $i_7$, $i_8$, $i_9$, $i_{10}$, $i_{11}$, $i_{12}$, $i_{13}$, $i_{14}$, $i_{15}$, $i_{16}$, and $i_{17}$, labelled as *Items* in the Fig.1. For example, the first transaction is $T_1$ containing $i_6$, $i_1$, $i_3$, $i_4$, $i_7$, $i_9$, $i_{13}$, and $i_{16}$.

| Transactions | Items |
|---|---|
| $T_1$ | $i_6, i_1, i_3, i_4, i_7, i_9, i_{13}, i_{16}$ |
| $T_2$ | $i_1, i_2, i_3, i_6, i_{12}, i_{13}, i_{15}$ |
| $T_3$ | $i_2, i_6, i_8, i_{10}, i_{15}$ |
| $T_4$ | $i_2, i_3, i_{11}, i_{17}, i_{10}$ |
| $T_5$ | $i_1, i_6, i_3, i_5, i_{12}, i_{16}, i_{13}, i_{14}$ |

Fig.1 A transaction database *DB*

## 2.2 Data Structure

In this subsection, we summarize background knowledge of the designing and construction of the *Vertical Index List* which introduced in [2, 9, 11].

**Definition 2.4**

Let $T_i=\{x_1,x_2,x_3,\ldots,x_m\}$ be a transaction in $DB$, where $i=1,2,\ldots,m$ *and* $x_j$ is an item for $j=1,2,\ldots,n$. A *Vertical Index List* (or *VIL*) is the structure constructed from a scan of each $T_i$ in $DB$ only once. Each row in *VIL* contains an item in $I$, support of item in $I$, and transactions in $DB$ which contain such an item. The set of transaction will be written in order according to the ascending of its identification number.

**Example 2.** We use an example *DB* in Fig.1. The *DB* is scanned once to create the *VIL*. We first create all the item set to the *VIL* and define all supports as zero.

The first transcation is $T_1$ and consists of itemset <$i_6$, $i_1$, $i_3$, $i_4$, $i_7$, $i_9$, $i_{13}$, $i_{16}$>. The $T_1$ will be inserted into corresponding item name ordering by sequential of itemset. Therefore, the $T_1$ will be the first inserted at the transaction of item <$i_6$> and increase support of item <$i_6$> with 1. The second examined item is <$i_1$>, we insert $T_1$ at item <$i_1$> and increase support of item <$i_1$> with 1. Next, we examine item <$i_3$>, we subsequently insert $T_1$ at item <$i_3$> and increase support of item with 1, then, the remaining items ($i_4$, $i_7$, $i_9$, $i_{13}$, $i_{16}$) in $T_1$ can be done in the same way. The remaining transactions ($T_2, T_3, T_4$, and $T_5$) in *DB* can also be done in the same way. We present the insertion of all transaction in Fig.2.

| Items | Support | Transactions |
|---|---|---|
| $i_1$ | 3 | $T_1$, $T_2$, $T_5$ |
| $i_2$ | 3 | $T_2$, $T_3$, $T_4$ |
| $i_3$ | 4 | $T_1$, $T_2$, $T_4$, $T_5$ |
| $i_4$ | 1 | $T_1$ |
| $i_5$ | 1 | $T_5$ |
| $i_6$ | 4 | $T_1$, $T_2$, $T_3$, $T_5$ |
| $i_7$ | 1 | $T_1$ |
| $i_8$ | 1 | $T_3$ |
| $i_8$ | 1 | $T_1$ |
| $i_{10}$ | 1 | $T_3$ |
| $i_{11}$ | 1 | $T_4$ |
| $i_{12}$ | 2 | $T_2$, $T_5$ |
| $i_{13}$ | 3 | $T_1$, $T_2$, $T_5$ |
| $i_{14}$ | 1 | $T_5$ |
| $i_{15}$ | 2 | $T_2$, $T_3$ |
| $i_{16}$ | 3 | $T_1$, $T_4$, $T_5$ |
| $i_{17}$ | 1 | $T_4$ |

Fig.2 The structure of *VIL*

The construction of the *VIL* is presented in the Fig.3.

**Algorithm 1 (Vertical-Index-List Construction)**
**Input**: *DB*, *minsup*
**Output**: *VIL*
**Method**: The *VIL* is constructed as follows
Begin
  scan the transaction database *DB* once.
  create all items and define all $supp(x_i)$= 0 to *VIL*.
  For each transaction $T_i$ in DB
    For each $x_i$ in $T_i$ do
      insert *TID* of $x_i$ to *TID-set* which
        corresponding with $x_i$
      count $supp(x_i)$
    End //For
  End //For
End //Begin

Fig.3 The construction of vertical index list algorithm

**Definition 2.5**
The sorted-list (or *SL*) is the structure consisting of any item *x* and its support which selected from *VIL* if $supp(x) \geq minsup$.

**Example 3**. For this example, let the user defined *minsup* be 3. The construction of the *SL* is started from selection any item in the *VIL* which *supp*(*item*) ≥*minsup* and contain selected to the *SL* with ascending sort of support , as seen in Fig.4. All items in Fig.4 are frequent items.

| Items | Support |
|---|---|
| $i_1$ | 3 |
| $i_2$ | 3 |
| $i_{13}$ | 3 |
| $i_{16}$ | 3 |
| $i_3$ | 4 |
| $i_6$ | 4 |

Fig.4 The structure of *SL*

The construction of the *SL* is presented in the Fig.5.

**Algorithm 2** (Sorted-List Construction)
**Input**: *VIL*
**Output**: *SL*
**Method**: The *SL* is construction as follows.
Begin
  For each $x_i$ in the *VIL*
    select $supp(x_i) \geq minsup$ to the *SL* with
      ascending sort of support
  End //For
End //Begin

Fig.5 The algorithm of the construction of the *VIL*.

## 3 The Approach

In this section, we present a new algorithm, called the *JoinFI-Mine* algorithm. The main features of this proposed algorithm are: (1) the frequent itemsets are found without generation of candidate itemsets; (2) the algorithm uses the efficiency of searching technique and mathematic properties to reduce subsequent of mining; and (3) the decision maker can change minimum support threshold without rescanning of database. From the above features, we obtained all frequent itemsets very quickly. We give the definitions, the examples and the algorithms for illustrative of how to mine frequent itemsets. We prove that the *JoinFI-Mine* algorithm can mine frequent itemsets completely and correctly.

**Definition 3.1**
Let *I* be the set of all items in *DB*, *S* be the set of all supports of items in *I*, $<_l$ the lexicographic order in *I* and < the usual less than order in *S*. We define the

order relation < in $I \times S$ by $(a, m) < (b, n)$ if $m=n$ and $a <_l b$ or if $m < n$, shortly we use $a < b$ for $(a,m) < (b,n)$. It is to note that in the table of *SL*, each row is of the form $(a,s) \in I \times S$ and clearly if $(a, m)$ and $(b, n)$ are the $i^{th}$ row and $j^{th}$ row of *SL* respecttively, then $(a, m) < (b, n)$ iff $i < j$, that is $a<b$ iff $i < j$.

**Example 3**. In this example, we describe the *SL*'s property which is shown in Fig.4 and definition 3.1. The set *I* consists of six items, and the set *S* consists of six corresponding supports. The relation of $I \times S$ are ordered by $<_l$ and $<$. Therefore, the order in the *SL* are $(i_1,3) < (i_2,3) < (i_{13},3) < (i_{16},3) < (i_3,4) < (i_6,4)$. The $i_1$'s support is equal to $i_2$'s support but $i_1 <_l i_2$, so $i_1$ appears before $i_2$. The $i_3$'s support is greater than the support of $i_1$, $i_2$, $i_{13}$, $i_{16}$, so the order of $i_3$ in the *SL* appears after $i_1$, $i_2$, $i_{13}$, and $i_{16}$.

**Definition 3.2**

Let $a_1$ and $a_2$ be two items in *SL* with $a_1<a_2$ and $supp(a_1a_2) \geq minsup$. Then we define $JF_2(a_1a_2):=\{a_1a_2\}$ and $JF_k(a_1a_2):=\{a_1a_2 \ldots a_k | a_3,\ldots,a_k$ are in *SL* where $a_1<\ldots<a_k$ with $supp(a_1\ldots a_k) \geq minsup\}$ for $k \geq 3$.

**Example 4**. In Fig.4, $i_1$, $i_{13}$, $i_3$, $i_6$ are items in *SL* such that $i_1 < i_{13} < i_3 < i_6$ and in Fig.2, $supp(i_1i_{13}i_3i_6)=3$ therefore $JF_2(i_1i_{13})=\{i_1i_{13}\}$, $JF_3=\{i_1i_{13}i_3\}$ and $JF_4=\{i_1i_{13}i_3\, i_6\}$.

**Definition 3.3**

Let $k \geq 2$ and $a_1,\ldots,a_k$ be items in *SL*. Then $a_1 \ldots a_k$ is called the *terminal frequent k-itemset* generated by $a_1a_2$ (denoted by $TI_k(a_1a_2)$) iff $a_1\ldots a_k \in JF_k(a_1a_2)$, $supp(a_1\ldots a_kb)<minsup$, $supp(a_1\ldots a_iba_{i+1}\ldots a_k)<minsup$ and $supp(ba_1\ldots a_k) < minsup$ if $b$ is in *SL* with $b \neq a_i$ for each $i=1,\ldots,k$.

**Definition 3.4**

Let $b_1\ldots b_i$ be $TI_i(b_1b_2)$. Then $b_1\ldots b_i$ is *repeated-itemset* (denoted by *RI*) if there exists $k > i$ such that $\{b_1,\ldots,b_i\} \subset \{a_1,\ldots,a_k\}$ and $a_1\ldots a_k$ is $TI_k(a_1a_2)$.

**Definition 3.5**

Let $a_1$ and $a_2$ be items in *SL* with $a_1<a_2$. Let $a_1\ldots a_k$ be $TI_k(a_1a_2)$ with $k \geq 3$. Then the set of all subsets *A* of $\{a_1,\ldots,a_k\}$ such that $|A| \geq 3$ is called an *extendable-itemset* generated by $a_1a_2$ and is denoted by $EI(a_1a_2)$.

**Definition 3.6**

We define $JF_1 := \{a \in I | a$ is an item in $SL\}$, $JF_2 := \{a_1a_2 | a_1a_2$ is $TI_2(a_1a_2)\}$, $JF_k := \{a_1a_2\ldots a_k | \{a_1,\ldots, a_k\} \subset \{b_1,\ldots,b_i\}$ where $b_1\ldots b_i$ is $TI_i(b_1b_2)$ for some $i\}$ for $k \geq 3$.

**Definition 3.7**.

The whole frequent itemsets are given by $WFI := \cup_{k \geq 1} JF_k$.

For the examples of definition 3.3 to 3.7, we can see more details in example 5. Based on the above definitions and examples, the *JoinFI-Mine* algorithm consists of the following steps as shown in Fig.6.

**Algorithm 3** (*JoinFI-Mine*: Mining of frequent itemsets by using *VIL* with control order of the frequent items in *SL*)
**Input**: *VIL*, *SL*, user define support: *ms*, number of transaction: *nt*
**Output**: The Complete set of frequent itemsets
**Procedure** FindMinsup(*ms*, *nt*)
Begin
  $minsup =$ ceil($(ms/100)*nt$)
End // Begin
**Procedure** JoinFI-Mine(*SL*,*VIL*, *minsup*, *x*)
Begin
  For $i$ = 1 to $n$
    c = $i$+1, $k = c$
    While $c <> n$ Do
      find $JF_k(x_ix_c)$
      If $supp(JF_k(x_ix_c)) \geq minsup$ Then
        call CkMostDepth
        call CkRI
      End // If
        $c = c+1$
    End // While
  End // For
  Result $WFI \coloneqq \cup_{k \geq 2} JF_k$
End // Begin
**Procedure** CkMostDepth ($JF_k(x_ix_c), i, n$)
Begin
  $\alpha = minsup$, $c = i+2$, $k=c$, $f = JF_k(x_ix_c)$
  For $c <= n$
    While (($c \leq n$) and ($\alpha \geq minsup$)) Do
      $c = c+1$
      If $\alpha = supp(f\,x_c) \geq minsup$ Then
        $f = f\,x_c$
      End // If
      $c = c+1$
    End // While
    $\alpha = minsup$
  End // For
  $\alpha = minsup$
  $TI_k(f)=f$
End// Begin
**Procedure** CkRI ($f$)
If $TI_k(f) \notin JF_k$ Then // insert new answer
  store $TI_k(f)$ to $JF_k$ //where $k$ is the size of $TI_k$
  call ExpandItemset($TI_k(f)$)
End //If
**Procedure** ExpandItemset ($TI_k(f)$)
If $|TI_k(f)|>2$ then //Expand itemset
  find all subset of $TI_k(f)$ (or EI) except $|EI| \leq 2$
  If $EI \notin JF_k$ then
    store $EI$ to $JF_k$ // where $k$ is the size of $TI_k$
  End //If
End // If

Fig.6 The proposed algorithm

ISBN: 978-960-474-273-8

An example 5 illustrates the details of the mining of frequent itemsets process based on definitions and algorithms in section 2 and section 3. The processing of reducing subsequent mining process which can be seen in this example and the result of all the answer sets is shown in Table 1. All frequent itemsets which separate by sequential of the *k*-level is presented in Table 2.

**Example 5.** Let the user want to make the decision at minimum support be 45% and according to Fig.2, Fig.4, Fig.6 and all definitions. First, we compute $minsup = ceil((45/100)*5) = 3$ when the number of transaction in Fig.1 is 5. The processing steps of mining are presented as follows:

Step (1) Examining the first frequent item: $<i_1>$.

Step (1.1) Starting the first item in Fig.4, it is an item $<i_1>$ and the following item is item $<i_2>$. Seeking out at item $<i_1>$ and $<i_2>$ at Fig.2, we get $JF_2(i_1i_2):=\{i_1i_2\}$. The checking of this step is terminated because $supp(JF_2(i_1i_2)) < minsup$.

Step (1.2) Seeking item $<i_1>$ and $<i_{13}>$ in Fig.4. Seeking out at item $<i_1>$ and $<i_{13}>$ at Fig.2, so $JF_2(i_1i_{13}):=\{i_1i_{13}\}$ which has support not less than *minsup*. Therefore, we test next sublevel ($k$=3), as seen in the following deep step of $JF_2(i_1i_2)$.

Step (1.2.1) In this step, we do CkMostDepth by seeking out at item $<i_{16}>$ at Fig.2. Therefore, $JF_3(i_1i_{13}i_{16}):=\{i_1i_{13}i_{16}\}$ which has support less than *minsup*. Therefore, we terminate this step and examine next step.

Step (1.2.2) In this step, we do CkMostDepth by seeking out at item $<i_3>$ at Fig.2. $JF_3(i_1i_{13}i_3):=\{i_1i_{13}i_3\}$ with support is 3, so we examine next deep step of step (1.2.2).

Step (1.2.2.1) We do CkMostDepth by seeking out at item $<i_6>$ at Fig.2. $JF_4(i_1i_{13}i_3i_6):=\{i_1i_{13}i_3i_6\}$ which has support not less than *minsup* and the CkMostDepth is terminated because we process until meet the last item in *SL*. We get $TI_4(i_1i_{13}i_3i_6)$ which its support not less than *minsup.* Next, we do CkRI and we get $TI_4(i_1i_{13}i_3i_6)$ is the new answer, so we save $\{i_1i_{13}i_3i_6\}$ to $JF_4$ or we can say $JF_4:=\{i_1i_{13}i_3i_6\}$. Next, we do ExpandItemset by examining at $|TI_4(i_1i_{13}i_3i_6)| \geq 3$, so we can use the *extendable-itemset* property to obtain frequent itemsets. $EI_4(i_1i_{13}i_3i_6) := \{<i_1i_{13}>, <i_1i_3>, <i_1i_6>, <i_{13}i_3>, <i_{13}i_6>, <i_3i_6>, <i_1i_{13}i_3>, <i_1i_{13}i_6>, <i_1i_3i_6>\}$. We get $JF_2 := \{<i_1i_{13}>, <i_1i_3>, <i_1i_6>, <i_{13}i_3>, <i_{13}i_6>, <i_3i_6>\}$, $JF_3 := \{<i_1i_{13}i_3>, <i_1i_{13}i_6>, <i_1i_3i_6>\}$. We then terminate this searching path.

Step (1.3) Seeking out at item $<i_{16}>$ at Fig.2. $JF_2(i_1i_{16}) := \{i_1i_{16}\}$ which has support less than *minsup*. Therefore, we terminate this step and examine next step.

Step (1.4) Seeking out at item $<i_3>$ at Fig.2. When we do similar above, we get $TI_3(i_1i_3i_6)$. We find that it is *RI*, so we terminate this step.

Step (1.5) Comparing operation of $JF_2(i_1i_6)$ is not required because an item $<i_6>$ is the last item of *SL* and is member of $JF_2$, so we terminate this step.

Step (2.1) Examine at item $<i_2>$ at Fig.4. We do similar with the above process. We get all steps of this item having support less than *minsup*, so we terminate this step.

Step (3.1) Examine at item $<i_{13}>$ and the next item is $<i_{16}>$ at Fig.4. Seeking out at item $<i_{13}>$ and $< i_{16}>$ at Fig.2 and getting its support not less than *minsup*, so we terminate this step.

Step (3.2) Examine at item $<i_{13}>$ and the next item is $< i_3>$ at Fig.4. Seeking out at item $<i_{13}>$ and $< i_3>$ at Fig.2. When we do similar above, we get $TI_3(i_{13}i_3i_6)$. We find that it is *RI*, so we terminate this step.

Step (3.3) Comparing operation of $JF_2(i_{13}i_6)$ is not required because an item $<i_6>$ is the last item of *SL* and is a member of $JF_2$, so we terminate this step.

Step (4.1) Examining the fourth item of *SL*: $<i_{16}>$; and the following item is $<i_3>$. Seeking out at item $<i_{16}>$ and $<i_3>$ at Fig.2, we get $JF_2(i_{16}i_3):=\{i_{16}i_3\}$. When we do similar above, we get $TI_3(i_{16}i_3i_6)$ which has support less than *minsup*. Therefore, $JF_2(i_{16}i_3)$ is $TI_2(i_{16}i_3)$, and is not *RI*. We save $JF_2(i_{16}i_3)$ to $JF_2$.

Step (4.2) Comparing operation of $JF_2(i_{16}i_6)$ is not required because an item $<i_6>$ is the last item of *SL* and is a member of $JF_2$, so we terminate this step.

Step (5.1) Examining the fifth item of *SL*: $< i_3>$; and the last item is $< i_6>$. Comparing operation of $JF_2(i_3i_6)$ is not required because an item $<i_6>$ is the last item of *SL* and is a member of $JF_2$, so we terminate this step.

After all of items are done, we present all frequent itemsets which separate by sequential of generation in Table 1 and by sequential of the frequent *k*-itemsets in Table 2.

Table 1
All frequent itemsets which separate by sequential of processing

| Items | Frequent Itemsets |
|---|---|
| $i_1$ | $<i_1i_3i_6i_{13}>,\{<i_1i_3>,<i_1i_6>,<i_1i_{13}>,<i_3i_6>, <i_3i_{13}>,<i_6i_{13}>,<i_1i_3i_6>,<i_1i_3i_{13}>,<i_1i_6i_{13}>, <i_3i_6i_{13}>\}$ |
| $i_2$ | ∅ |
| $i_{13}$ | ∅ |
| $i_{16}$ | $<i_3i_{16}>$ |
| $i_3$ | ∅ |

The Table 1 shows that the *JoinFI-Mine* algorithm is able to expand frequent itemsets and reduce subsequence of mining. The mining step of item $<i_1>$

can show the expansion of frequent itemsets which is $JF_4$:={$i_1i_{13}i_3i_6$} and the result of expansion are {<$i_1i_3$>, <$i_1i_6$>, <$i_1i_{13}$>, <$i_3i_6$>, <$i_3i_{13}$>, <$i_6i_{13}$>, <$i_1i_3i_6$>, <$i_1i_3i_{13}$>, <$i_1i_6i_{13}$>, <$i_3i_6i_{13}$>}. Moreover, the operation of *EI* reduces the operation of the following items such as <$i_{13}$> and <$i_3$>. Because we can get some answer in previous operation so we do not operate for some item in *SL*.

Table 2

All frequent itemsets which separate by *k*-level

| *k* | frequent *k*-itemsets |
|---|---|
| 2 | <$i_1i_3$>,<$i_1i_6$>,<$i_1i_{13}$>,<$i_3i_6$>,<$i_3i_{13}$>,<$i_3i_{16}$>,<$i_6i_{13}$> |
| 3 | <$i_1i_3i_6$>,<$i_1i_3i_{13}$>,<$i_1i_6i_{13}$>,<$i_3i_6i_{13}$> |
| 4 | <$i_1i_3i_6i_{13}$> |

The Table 2 shows all frequent itemsets which separate by *k*-level such frequent 2-itemsets, frequent 3-itemsets, and frequent 4-itemsets or $JF_2$ = {<$i_1i_3$>, <$i_1i_6$>, <$i_1i_{13}$>, <$i_3i_6$>, <$i_3i_{13}$>, <$i_3i_{16}$>, <$i_6i_{13}$>}, $JF_3$ = {<$i_1i_3i_6$>, <$i_1i_3i_{13}$>, <$i_1i_6i_{13}$>, <$i_3i_6i_{13}$>}, and $JF_4$ = {<$i_1i_3i_6i_{13}$>}. Therefore, all frequent itemsets which appears in Table 2 are *WFI*.

## 4 The Correctness

In the following theorem, we present that the proposed algorithm can mine all frequent itemsets completely and correctly.

**Theorem**.

The *WFI* is the answer set.

**Proof**.

Let $a_1a_2…a_k$ be in *WFI*. Then $a_1a_2 …a_k \in JF_k$. If $k$=1, then $a_1$ is in *SL* and thus $a_1$ is a frequent item. If $k$=2, then $a_1a_2$ is $TI_2(a_1a_2)$ and therefore clearly from definition 2 and definition 3, $a_1a_2$ is a frequent itemset. For $k \geq 3$, from definition 6 there exist $i \geq k$ and $b_1,...,b_i$ such that $\{a_1,…,a_k\} \subset \{b_1,…,b_i\}$ and $b_1,…,b_i$ is $TI_i(b_1b_2)$, i.e., $a_1...a_k \in EI(a_1a_2)$. Hence by definition 2 and definition 3, we see that $a_1… a_k$ is a frequent $k$-itemset.

Conversely, let $a_1a_2…a_k$ be a frequent $k$-itemset. It is easy to see that if $k$=1,2, then $a_1a_2…a_k \in WFI$ and for $k \geq 3$, if $a_1...a_k$ is not $TI_k(a_1a_2)$, then it is in some $JF_k(b_1b_2)$ and therefore $a_1a_2...a_k \in JF_k$. Hence $a_1...a_k \in WFI$. The proof is complete. □

## 5 Conclusion

We have presented a new algorithm to mine all frequent itemsets, named *JoinFI-Mine* algorithm. This algorithm reads transaction database by scanning only one time and does not generate candidate sets. Our method reduces huge of subsequence mining by using mathematics properties so we can find all frequent itemsets very quickly and also correctly. In case that the decision maker wants to change the minimum support threshold, our algorithm is performed without rescanning of database. We presented our method by giving definitions, algorithms, examples, and concluded by proving correctness of the proposed algorithm. The proof shows that our algorithm can mine all frequent itemsets completely and correctly.

*References:*

[1] R. Agrawal and R. Srikant: Fast Algorithm for Mining Association Rules, *Proceedings of the 20th International Conference on Very Large Data Bases*, Chile, September 1994, 487-499.

[2] P-N. Tan, M. Steinbach, & V. Kumar, *Introduction to Data Mining* (Pearson Education Inc., 2006).

[3] J. Han, J. Pei, & Y. Yin: Mining Frequent Pattern without Candidate Generation, *Proceedings of the 2000 ACM SIGMOD international conference on Management of Data*, Texas, May 2000, 1-12.

[4] J. Han, J. Pei, Y. Yin, & R. Mao: Mining Frequent Pattern without Candidate Generation: a Frequent Pattern Tree, *Springer, vol.* 8, 2004, no 1, 53-87.

[5] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, & D. Yang: Hmine: Hyper-Structure Mining of Frequent Patterns in Large Databases, *Proceedings of the 2001 IEEE International Conference on Data Mining*, USA, November 2001, 441-448.

[6] A. Pietracaprina, & D.Zandolin: Mining Frequent Itemsets Using Patricia Tries, *Proceedings of the 3rd IEEE International Conference on Data Mining*, Florida, USA, November 2003.

[7] G. Grahne, & J. Zhu: Efficiently Using Prefix-Trees in Mining Frequent Itemsets," *Proceedings of the 3rd IEEE International Conference on Data Mining,* Florida, USA, November 2003.

[8] S. Sahaphong, & V. Boonjing: The Combination Approach to Frequent Itemsets Mining, *Proceedings of the 2008 International Conference on Convergence and hybrid Information Technology*, Korea, November 2008, 565-570.

[9] M.J. Zaki, Scalable Algorithms for Association Mining, *IEEE Transaction on Knowledge and Data Engineering, vol.* 12, no. 3, 2000, 372-390.

[10] D. J. Chai, L. Jin, B. Hwang, & K. H. Ryu: Frequent Pattern Mining Using Bipartite Graph, *Proceedings of the 18th International Conference on Database and Expert Systems Applications*, Germany, August 2007, 182-186.

[11] S. Sahaphong, Frequent Itemsets Mining Using Vertical Index List, *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*, China, August 2009, 480-484.

[12] J. Han, & M. Kamber, *Data Mining: Concepts and Techniques*, Elsevier, Maryland Heights MO, 2006.