NOV-MFI: A Novel Algorithm for Maximal Frequent Itemset Mining

Huan Phan[#,*]
[#]Faculty of Mathematics and Computer Science
University of Science, VNU-HCM, Ho Chi Minh , 700000, Vietnam
[*]Division of IT
University of Social Sciences and Humanities, VNU-HCM, Ho Chi Minh, 700000, Vietnam
{huanphan@hcmussh.edu.vn}

ABSTRACT: *Since data explosion, data mining in transactional databases are increasingly important. There are many techniques for data mining such as mining association rule, the most important and well-researched. Moreover, maximal frequent itemset mining is one of the basic but time-consuming steps in the mines of association rules. Most algorithms used in the literature find maximal frequent itemset on search space items that have support at least minsup and not be used again for mining. In this paper, we propose a novel algorithm called NOV-MFI for mining maximal frequent itemsets in transactional databases. Advantages of NOV-MFI algorithms are reuse and easily expanded in distributed systems. Finally, experimental results show that the proposed algorithms are better than other existing algorithms on both real and synthetic datasets.*

Keywords: Association Rules, Maximal Frequent Itemset, NOV-MFI Algorithm

Received: 10 September 2019, Revised 20 December 2019, Accepted 19 January 2020

DOI: 10.6025/jcl/2020/11/2/60-72

1. Introduction

There are variously foundational and necessary problems in the applications of data mining (e.g. the discovery of association rules, strong rules, correlations, multidimensional pattern, and many other essential discovery tasks). Among of them, mining frequent itemsets (**FI**) is an important one. The formulation of this problem is being given a large transactional database, then finding all frequent itemsets, where a frequent item occurs in at least a user specified percentage of transaction database [1-3], [11]. When we are mining association rules in transaction database, a huge number of frequent itemsets will be generated. Authors around the world proposed mining for maximal frequent itemsets (**MFI**) [4-10]; **MFI** are nonredundant representations of all frequent itemsets.

Most mining algorithms for maximal frequent itemsets, proposed by authors form around the world, as algorithms **Depth Project** [4], **Mafia** [5], **GenMax** [6], **MaxMining** [7],.... Algorithms generated candidate using a breadth-first search, with tidset format. Besides, the algorithms improve upon a depth-first search, diffset format. The main limitation of algorithms based on **IT-Tree** a major advance developed using pattern-growth based on **FP-Tree**. In recent years, Wang H, proposed the **MaxMining** [7] algorithm based on depth-first search and vertical tidset format to mining maximal frequent itemsets from a transaction database and shows the better performance result. In this paper, we propose a **NOV-MFI** algorithm for maximal frequent itemsets mining, moreover easily expanded in distributed systems.

• ***Algorithm* 1:** Computing *Kernel_COOC* array of cooccurrences and occurrences of kernel item in at least one transaction;

• ***Algorithm* 2:** Detecting maximal frequent itemsets based on *Kernel_COOC* array;

This paper is organized as follows: section 2 represent the basic concepts for mining maximal frequent itemsets and data structure for transaction databases. Some theoretical aspects of our approach are based on section 3. Besides, we represent our **NOV-MFI** algorithm to detect maximal frequent itemsets based on *Algorithm 1* and *Algorithm 2*. Details on implementation and experimental results are debated in section 4. In section 5, we summarize our approach, perspectives and the extension of this future work.

2. Background

In this section, we represent the basic concepts for mining maximal frequent itemsets and data structure for transaction databases.

2.1 Maximal Frequent Itemset Mining

Let $I = \{i_1, i_2,..., i_m\}$ be a set of m distinct items. A set of items $X = \{i_1, i_2,..., i_k\}$, $\forall\, i_j \in I\ (1 \le j \le k)$ is called an itemset, an itemset with k items is called a *k-itemset*. D be a dataset containing n transaction, a set of transaction $T = \{t_1, t_2,..., t_n\}$ and each transaction $t_k = \{i_{k1}, i_{k2},..., i_{kl}\}$, $\forall\, i_{kj} \in I\ (1 \le kj \le k)$.

Definition 1: The support of an itemset X is the number of transaction in which occurs as a subset, denoted as $sup(X)$.

Definition 2: Let *minsup* be the threshold minimum support value specified by user. If $sup(X) \ge minsup$, itemset X is called a frequent itemset, denoted ***FI*** is the set of all the frequent itemset.

Definition 3: Itemset X is called a maximal frequent itemset: If $sup(X) \ge minsup$ and does not exist any itemset $Y \supset X$ then $sup(Y) \ge minsup$, denoted ***MFI*** is the set of all the maximal frequent itemset.

We use the transactional database D for examples in Table 1.

TID	**Items**							
t1	A		C		E	F		
t2	A		C				G	
t3					E			H
t4	A		C	D		F	G	
t5	A		C		E		G	
t6					E			
t7	A	B	C		E			
t8	A		C	D				
t9	A	B	C		E		G	
t10	A		C		E	F	G	

Table 1. The Transaction Database D used as Our Running Example

k- itemset	FI (minsup = 3) # FI = 19	MFI (minsup = 3) # MFI = 2	FI (minsup = 5) # FI = 11	MFI (minsup = 3) # MFI = 2
1	F, G, E, A, C		G, E, A, C	
2	FA, FC, GE, GA, GC, EA, EC, AC		GA, GC, EA, EC, AC	
3	FAC, GEA, GEC, GAC, EAC	FAC	GAC, EAC	GAC, EAC
4	GEAC	GACE		

Table 2. FI, MFI of D with minsup = 3 and minsup = 5

Example 1: See Table 1. There are eight different items items I = {*A, B, C, D, E, F, G, H*} and ten transactions $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10},\}$. Table 2 shows the frequent itemsets and maximal frequent itemsets at *two minsup values* – 3 (30%) and 5 (50%) correspondingly (**MFI** $\subseteq$ **FI**).

2.2 Data Structure presentation for Transaction Database

Binary matrix is an efficient data structure for the mine of frequent items [2], [3], [12]. The process starts with a transactional database transformed into a binary matrix BiM, in which each row corresponds to a transaction and each column corresponds to an item. Each element in the binary matrix BiM contains 1 if the item is presented in the current transaction; otherwise it contains 0, as shown in Figure 1.

TID	A	B	C	D	E	F	G	H
t1	1	0	1	0	1	1	0	0
t2	1	0	1	0	0	0	1	0
t3	0	0	0	0	1	0	0	1
t4	1	0	1	1	0	1	1	0
t5	1	0	1	0	1	0	1	0
t6	0	0	0	0	1	0	0	0
t7	1	1	1	0	1	0	0	0
t8	1	0	1	1	0	0	0	0
t9	1	1	1	0	1	0	1	0
t10	1	0	1	0	1	1	1	0

Figure 1. Binary matrix BiM presentation of D'

3. The Proposed Algorithms

3.1 Generating Array of Co-occurrence Items of Kernel Item

In this section, we represent the framework of the algorithm generating array of co-occurrence items of items in dataset [12].

Definition 4: Project set of item i_k on database D **:** $\pi(i_k) = \{t_j \in D \mid i_k \subseteq t_j\}$ is set of transaction contain item i_k (π – ***decreasing monotonic***) .According to *Definition* 1:

$$sup(i_k) = |\pi(i_k)| \quad (1)$$

Example 2: In Table 1. Consider item F, we detect project set of item F on D: $\pi(F) = \{t_1, t_4, t_{10}\}$ then $sup\,(F) = |\pi(F)| = 3$.

Definition 5: Project set of itemset $X = \{i_1, i_2, ..., i_k\}$, $\forall i_j \in I (1 \le j \le k) : \pi(X) = \{\pi(i_1) \cap \pi(i_2) \ldots \pi(i_k)\}$.

$$sup(X) = |\pi(X)| \quad (2)$$

Example 3: In Table 1. Consider item G, we detect project set of item G on database D: $\pi(G) = \{t_2, t_4, t_5, t_9, t_{10}\}$ then $sup(FG) = |\pi(FG)| = |\pi(F) \cap \pi(G)| = |\{t_1, t_4, t_{10}\} \cap \{t_2, t_4, t_5, t_9, t_{10}\}| = 2$.

Definition 6: Let $i_k \in I$ is called a kernel item. Itemset $X_{cooc} \subseteq I$ is called co-occurrence items with kernel item i_k, so that satisfy $\pi(i_k) \equiv \pi(i_k \cup X_{cooc})$. Denoted as $cooc(i_k) = X_{cooc}$.

Example 4: In Table 1. Consider item F as kernel item, we detect co-occurrence items with item F as $cooc(F) = \{A, C\}$ and $sup(F) = sup(FAC) = 3$.

Definition 7: Let $i_k \in I$ is called a kernel item. Itemset $Y_{looc} \subseteq I$ is called occurrence items with kernel item i_k in as least one transaction, but not co-occurrence items, so that satisfy $1 \le |\pi(i_k \cup i_{looc})| < |\pi(i_k)|$, $\forall i_{looc} \in Y_{looc}$. Denoted as $looc(i_k) = Y_{looc}$.

Example 5: In Table 1. Consider item F as kernel item, we detect occurrence items with item F in as least one transaction $looc(F) = \{G, E\}$ and $\pi(FG) = \{t_4, t_{10}\} \subset \pi(F) = \{t_1, t_4, t_{10}\}$.

Algorithm Generating Array of Co-occurrence Items

This algorithm is generating co-occurrence items of items in transaction database and archive into the *Kernel_COOC* array. Each element within the *Kernel_COOC*, 4 fields:

- *Kernel_COOC*[*k*].item: *kernel item k*;
- *Kernel_COOC*[*k*].sup: support of *kernel item k*;
- *Kernel_COOC*[*k*].cooc: co-occurrence items with *kernel item k*;
- *Kernel_COOC*[*k*].looc: occurrence items *kernel item k* in least one transaction.

The framework of **Algorithm 1** is as follows:

Algorithm 1. Generating Array of Co-occurrence Items

Input : Dataset *D*

Output : *Kernel_COOC array, matrix BiM*

```
1:          foreach Kernel_COOC[k] do
2:                Kernel_COOC[k].item = i_k
3:                Kernel_COOC[k].sup = 0
4:                Kernel_COOC[k].cooc = 2^m - 1
5:                Kernel_COOC[k].looc = 0
6:          foreach t_j ∈ T do
7:          foreach i_k ∈ t_j do
8:                      Kernel_COOC[k].sup ++
9:                      Kernel_COOC[k].cooc AND=vectorbit(t_j)
10:                     Kernel_COOC[k].looc OR= vectorbit(t_j)
11: sort Kernel_COOC array in ascending by support
```

We illustrate **Algorithm 1** on database in Table 1.

Initialization of the Kernel_COOC array, number items in database m = 8;

item	A	B	C	D	E	F	G	H
sup	0	0	0	0	0	0	0	0
cooc	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
looc	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Read once of each transaction from $t1$ to $t10$

Transaction $t1$ = {A, C, E, F} has vector bit representation **10101100**;

item	A	B	C	D	E	F	G	H
sup	1	0	1	0	1	1	0	0
cooc	10101100	11111111	10101100	11111111	10101100	10101100	11111111	11111111
looc	10101100	00000000	10101100	00000000	10101100	10101100	00000000	00000000

Trans $t2$ ={A, C, G} has vector bit representation **10100010**;

item	A	B	C	D	E	F	G	H
sup	2	0	2	0	1	1	1	0
cooc	**10100000**	11111111	**10100000**	11111111	10101100	10101100	**10100010**	11111111
looc	**10101110**	00000000	**10101110**	00000000	10101100	10101100	**10100010**	00000000

Trans $t3$ = {E, H} has vector bit representation **00001001**;

item	A	B	C	D	E	F	G	H
sup	2	0	2	0	2	1	1	1
cooc	10100000	11111111	10100000	11111111	**00001000**	10101100	10100010	**00001001**
looc	10101110	00000000	10101110	00000000	**10101101**	10101100	10100010	**00001001**

Trans $t4$ = {A, C, D, F, G} has vector bit representation **10110110**;

item	A	B	C	D	E	F	G	H
sup	3	0	3	1	2	2	2	1
cooc	**10100000**	11111111	**10100000**	**10110110**	00001000	**10100100**	**10100010**	00001001
looc	**10111110**	00000000	**10111110**	**10110110**	10101101	**10111110**	**10110110**	00001001

Trans $t5$ = {A, C, E, G} has vector bit representation **10101010**;

item	A	B	C	D	E	F	G	H
sup	4	0	4	1	3	2	3	1
cooc	**10100000**	11111111	**10100000**	10110110	**00001000**	10100100	**10100010**	00001001
looc	**10111110**	00000000	**10111110**	10110110	**10101111**	10111110	**10111110**	00001001

Transaction $t6$ = {E} has vector bit representation **00001000**;

item	A	B	C	D	E	F	G	H
sup	4	0	4	1	4	2	3	1
cooc	10100000	11111111	10100000	10110110	**00001000**	10100100	10100010	00001001
looc	10111110	00000000	10111110	10110110	**10101111**	10111110	10111110	00001001

Trans $t7$ = {A, B, C, E} has vector bit representation **11101000**;

item	A	B	C	D	E	F	G	H
sup	5	1	5	1	5	2	3	1
cooc	**10100000**	**11101000**	**10100000**	10110110	**00001000**	10100100	10100010	00001001
looc	**11111110**	**11101000**	**11111110**	10110110	**11101111**	10111110	10111110	00001001

Trans $t8 = \{A, C, D\}$ has vector bit representation **10110000**;

item	A	B	C	D	E	F	G	H
sup	6	1	6	2	5	2	3	1
cooc	**10100000**	11101000	**10100000**	**10110000**	00001000	10100100	10100010	00001001
looc	**11111110**	11101000	**11111110**	**10110110**	11101111	10111110	10111110	00001001

Trans $t9 = \{A, B, C, E, G\}$ has vector bit representation **11101010**;

item	A	B	C	D	E	F	G	H
sup	7	2	7	2	6	2	4	1
cooc	**10100000**	**11101000**	**10100000**	10110000	**00001000**	10100100	**10100010**	00001001
looc	**11111110**	**11101010**	**11111110**	10110110	**11101111**	10111110	**11111110**	00001001

The last, transaction $t10 = \{A, C, E, F, G\}$ has vector bit representation **10101110**;

item	A	B	C	D	E	F	G	H
sup	8	2	8	2	7	3	5	1
cooc	10100000	11101000	10100000	10110000	00001000	10100100	10100010	00001001
looc	11111110	11101010	11111110	10110110	11101111	10111110	11111110	00001001

After the processing of **Algorithm 1**, the Kernel_COOC array as shown in Table 3:

Item	**H**	**B**	**D**	**F**	**G**	**E**	**A**	**C**
Sup	1	2	2	3	5	7	8	8
cooc	E	A, C, E	A, C	A, C	A, C	∅	C	A
Iocc	∅	G	F, G	D, E, G	B, D, E, F	A, B, C, F, G, H	B, D, E, F, G	B, D, E, F, G

Table 3. Kernel_Cooc array are ordered in support ascending order

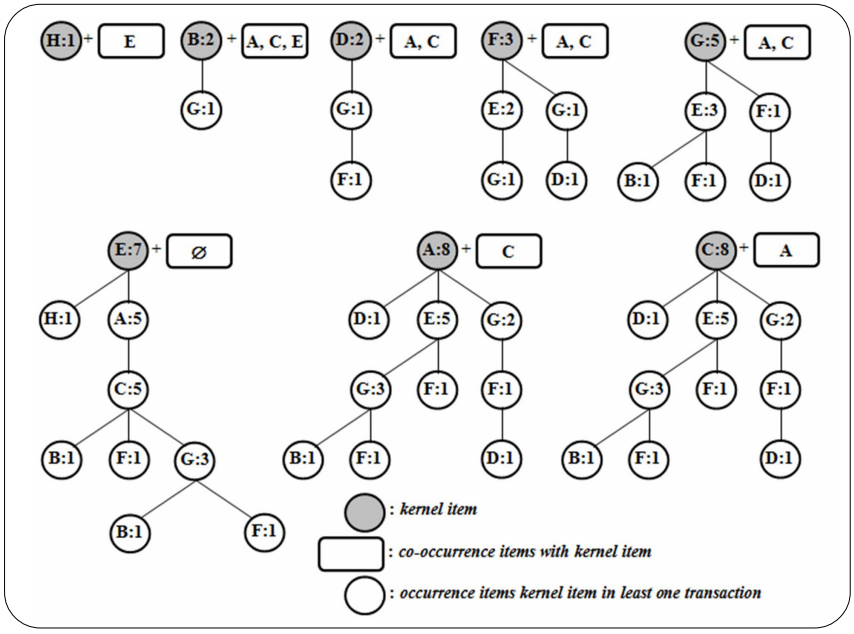

Figure 2. The pattern-space of occurrence items with *kernel item* in as least one transaction

Figure 2, show pattern-space of co-occurrence items and occurrence in as least one transaction with *kernel item*. We have *cooc* $(A) = \{C\}$ and *cooc* $(C) = \{A\}$. In this case, the frequent itemset generated from A and C items will be duplicated. We provide a *Definition* 8, 9 to eliminate the duplication when generating frequent itemsets.

Definition 8: Let $i_k \in I (i_1 \prec i_2 \prec ... \prec i_m)$ items are ordered in support ascending order, i_k is called a kernel item. Itemset $X_{lexcooc} \subseteq I$ is called co-occurrence items with the kernel item i_k, so that satisfy $\pi(i_k) \equiv \pi(i_k \cup i_j), i_k \prec i_j, \forall\, i_j \in X_{lexcooc}$. Denoted as $lexcooc(i_k) = X_{lexcooc}$.

Definition 9: Let $i_k \in I (i_1 \prec i_2 \prec ... \prec i_m)$ items are ordered in support ascending order, i_k is called a kernel item. Itemset $Y_{lexlooc} \subseteq I$ is called occurrence items with kernel item i_k in as least one transaction, but not co-occurrence items, so that satisfy $1 \leq |\pi(i_k \cup i_{lexlooc})| < |\pi(i_k)|, \forall\, i_{lexlooc} \in Y_{lexlooc}$. Denoted as $lexlooc(i_k) = Y_{lexlooc}$.

Additional command line 12, 13 and 14 into **Algorithm 1:**

```
12: foreach i_k ∈ t_j do
13:    Kernel_COOC[k].cooc = lexcooc(i_k)
14:    Kernel_COOC[k].looc = lexlooc(i_k)
```

According to *Definition 8*, we have *cooc* $(C) = \{A\}$, where $A \prec C$ so *lexcooc* $(C) = \{\varnothing\}$. Similarly, according to *Definition* 9, we have *looc* $(G) = \{B, D, E, F\}$, where $B, D \prec F \prec G \prec E$, so *lexlooc* $(G) = \{E\}$. Execute command line 12, 13 and 14 has result on Table 4.

Item	H	B	D	F	G	E	A	C
Sup	1	2	2	3	5	7	8	8
Cooc	E	A, C, E	A, C	A, C	A, C	∅	C	∅
Iooc	∅	G	F,G	G,E	E	A, C	∅	∅

Table 4. Kernel_Cooc are co-occurrence items ordered in support ascending order and reduced

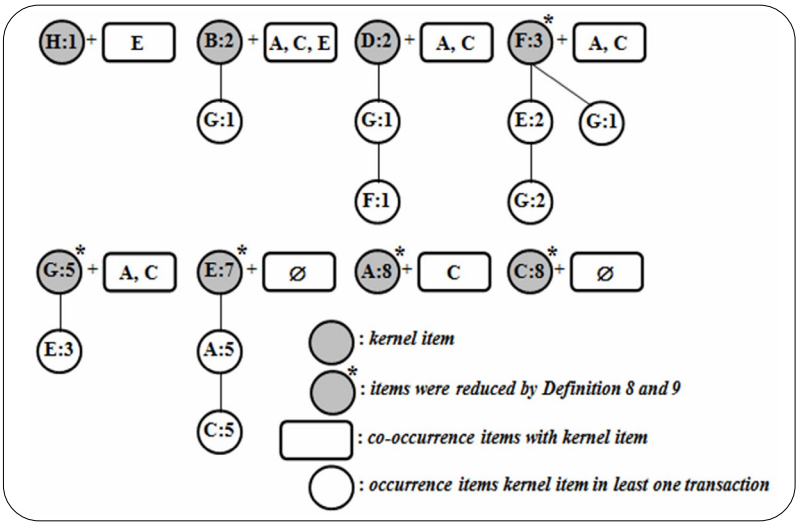

Figure 3. The pattern-tree was reduced by Definition 8 and 9

Figure 3, show pattern-space was reduced by Definition 8 and 9. We have *kernel items* changed as being *items F, G, E, A and C*.

3.2 Algorithm Generating All Maximal Frequent Itemsets

In this section, we represent the framework of the algorithm generating LOCAL maximal frequent itemsets of *Kernel items* bases on the Kernel_COOC array.

Lemma 1: $\forall i_k \in I$, if $sup(i_k) \geq minsup$ and itemset $\overline{X}_{lexcooc}$ is set of for all element of $lexcooc(i_k)$ then $sup(i_k \cup \overline{X}_{lexcooc}) \geq minsup$ and itemset $\{i_k \cup \overline{X}\, lexcooc\}$ is local maximal frequent itemset of *kernel item* i_k.

Proof. According to Definition 8, equation (1) and (2): $\pi(i_k) \equiv \pi(i_k \cup X_{lexcooc})$ and $sup(i_k) \geq minsup$. Therefore, we have $sup(i_k \cup X_{lexcooc}) \geq minsup$. According to Definition 9, $\forall y_{lexlooc} \in Y_{lexlooc}$, *Ylexlooc* is powerset of $lexlooc(i_k)$, we have $sup(i_k \cup y_{lexlooc} \cup X_{lexcooc}) < sup(i_k \cup X_{lexcooc})$.

Example 6: In Table 4. Consider the item *D* as kernel item (*minsup* = 2), we detect co-occurrence items with the item *D* as $lexcooc(D) = \{A, C\}$ a nd $\overline{lexcooc\ (D)} = AC$ then $sup(DAC) = 2 \geq minsup$ and itemset *DAC* is **MFI**.

Lemma 2: $\forall i_k \in I$, $Y_{lexlooc}$ is powerset of $lexlooc(i_k)$, $\forall y_{lexlooc} \in Y_{lexlooc}$, if $sup(i_k \cup y_{lexlooc}) \geq minsup$, $!\exists z_{lexlooc} \supset y_{lexlooc}$ so that $sup(i_k \cup z_{lexlooc}) < sup(i_k \cup y_{lexlooc})$ and $\overline{X}_{lexcooc}$ is set of all element of $lexcooc(i_k)$ then $sup(i_k \cup y_{lexlooc} \cup \overline{X}_{lexcooc}) \geq minsup$ and itemset $\{i_k \cup y_{lexlooc} \cup \overline{X}_{lexcooc}\}$ is local maximal frequent itemset.

Proof. According to Definition 8, 9 and lemma 1: we have, $|\pi(i_k \cup y_{lexlooc})| < |\pi(i_k)| = |\pi(i_k \cup x_{lexcooc})|$ and $sup(i_k \cup y_{lexlooc}) \geq minsup$. Therefore, we have $sup(i_k \cup y_{lexlooc} \cup \overline{X}_{lexcooc}) \geq minsup$ and is local maximal frequent itemset.

Example 7: In Table 4. Consider the item *G* as kernel item (*minsup* = 2), we detect co-occurrence items with item *G* as $lexcooc(G) = \{A, C\}$, $\overline{X}\, lexcooc = AC$, $lexlooc(G) = \{E\}$, $sup(GE) = 3 \geq minsup$ then sup(GEAC) = 3 ≥ *minsup* and itemset *GEAC* is local maximal frequent itemset.

Property 1: If $sup(i_{k-1}) = sup(i_k)$ and $i_k \in lexcooc(i_{k-1})$ then $i_{k-1} \equiv i_k$ (generating maximal frequent itemset from item i_{k-1} then not consider item i_k).

The pseudocode of **Algorithm 2** is presented as follows:

```
Input : minsup, Kernel_COOC array, Dataset ´
Output : MFI consists all maximal frequent itemsets
1: foreach Kernel_COOC[k].sup ≥ minsup do
2:    if (NOT satisfy Property 1) then
3:          if (Kernel_COOC[k].sup = minsup) then
4:                X_co = GenFull(Kernel_COOC[k].cooc)// X̄ lexcooc
5:                LocalMFI[k] = LocalMFI[k]∪{i_k ∪ X_co}//lem 1
6:          else
7:                X_co = GenFull(Kernel_COOC[k].cooc)// X̄ lexcooc
8:                LocalMFI[k] = LocalMFI[k] ]∪{i_k ∪ X_co}
9:                Lo ← GenSub(Kernel_COOC[k].looc)//noempty
10:               foreach is_j ∈ Lo do
11:                     F_t = F_t ∪ {i_k ∪ is_j}//long frequent itemset
```

```
12:                foreach f_i ∈ F_t do
13:                        LocalMFI[k] = LocalMFI[k]∪{X_co ∪ f_i}//lem 2
14: MFI = FilterMFI(LocalMFI)//remove NOT MFI
```

3.3 The NOV-MFI Algorithm

In this section, we represent the diagram of **NOV-MFI** algorithm for mining maximal frequent itemsets, as follows:

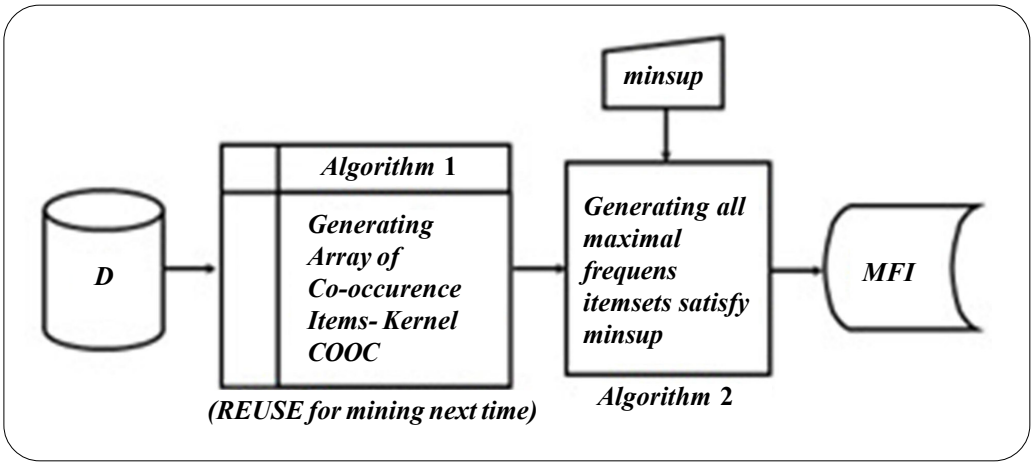

Figure 4. The diagram NOV-MFI for maximal frequent itemsets mining

We illustrate **NOV-MFI** algorithm on example database in Table 1, and *minsup* = 2. After the processing **Algorithm 1,** the Kernel_COOC array in Table 4 is showed.

• Consider kernel items satisfying *minsup* as items {*B, D, F, G, E, A, C*}//line 1;

• Consider *kernel item B*, *sup*(*B*) = 2 = *minsup* (*Lemma 1- line 3 to 5*) generating maximal frequent itemset with *kernel item B* as $\text{LocalMFI}_{[B]}$ = {(**B**EAC, 2)}.

• Consider *kernel item D*, *sup*(*B*) = 2 = *minsup* (similary *kernel item B*) generating maximal frequent itemset with *kernel item D* as $\text{LocalMFI}_{[D]}$ = {(**D**AC, 2)}.

• Consider the *kernel item F*, *sup*(*F*) = 3 > *minsup* (*from line 7 to 13*): generating frequent itemsets $\text{LocalMFI}_{[F]}$ = {(**F**AC, 3)}; *line 9* – generating noempty subsets of *looc* field *Lo* = {*G, E*}, F_k = {(**F**G, 2),(**F**E, 2)} – generating maximal frequent itemsets $\text{LocalMFI}_{[F]}$ = {(**F**ACG, 2), (**F**ACE, 2)}}.

• Consider the *kernel item G*, *sup*(*G*) = 5 > *minsup* (similary *kernel item F*): generating frequent itemsets LocalMFI[G] = {(**G**AC, 5)}; *line 9* – generating noempty subsets of *looc* field *Lo* = {*E*}, F_k = {(**G**E, 3)} – generating maximal frequent itemsets $\text{LocalMFI}_{[G]}$ = {(**G**ACE, 3)}.

• Consider the *kernel item* E, *sup*(E) = 7 > *minsup* (similary *kernel item F*): generating frequent itemsets LocalMFI[E] = {(**E**, 7)}; *line 9* – generating noempty subsets of *looc* field *Lo* = {*A, C*}, F_k = {(**E**AC, 5)} – generating maximal frequent itemsets $\text{LocalMFI}_{[E]}$ = {(**E**AC, 5)}.

• Consider the *kernel item* A, *sup*(A) = 8 > *minsup* (similary *kernel item F*): generating frequent itemsets $\text{LocalMFI}_{[A]}$ = {(**A**C, 8)}; *line 9* – generating noempty subsets of *looc* field *Lo* = {∅} – generating maximal frequent itemsets $\text{LocalMFI}_{[E]}$ = {(**A**C, 8)}.

• The *kernel item* C is not consider – NOT satisfy *line 2* (*Pro 1*).

• *Line 14*, remove NOT maximal frequent itemset from LocalMFI are {(**E**AC, 5), (**A**C, 8)}.

Kernel item	Maximal frequent itemsets - MFI
B	(**B**EAC, 2)
D	(**D**AC, 2)
F	(**F**ACG, 2) (**F**ACE, 2)
G	(**G**ACE, 3)

Table 5. MFI of *D* with minsup = 2

4. Experiment Results

In the experiment, we were conducted on a computer with a CPU 2.0 GHz, 4Gb main memory, running Microsoft Windows 7 Ultimate. All codes were compiled using C#, MVS 2010, .Net Framework 4.

We experimented on two instance types of datasets:

• Two real datasets are both dense form of UCI Machine Learning Repository [http://archive.ics.uci.edu/ml] as **Chess** and **Mushroom** datasets.

• Two synthetic sparse datasets are generated by software of IBM Almaden Research Center [http://www.almaden.ibm.com] as **T10I4D100K** and **T40I10D100K** datasets.

Name	#Transcation	#Items	#Avg. Length	Density (%)	Type
Chess	3,196	75	37	49.3	Dense
Mushroom	8,142	119	23	19.3	Dense
T10I4D100K	100,000	870	10	1.1	Sparse
T40I10D100K	100,000	942	40	4.2	Sparse

Table 6. Datasets used in experiment

We have compared the **NOV-MFI** algorithm with two algorithms: the first algorithm is **GenMax** [6] based on ITTree structure; the second algorithm is **MaxMining** [7] constructive a *FP-tree-like*.

Figure 5, 6 show the running time of the compared algorithms on real datasets **Chess** and **Mushroom**. **NOVMFI** runs faster among two algorithms **GenMax** and **MaxMining** under all minimum supports.

Figure 7, 8 show the running time of the compared algorithms on synthetic sparse datasets **T10I4D100K** and **T40I10D100K**. **NOV-MFI** runs faster among two algorithms **GenMax** and **MaxMining** under all minimum supports. However, figure 7, 8 shows **NOV-MFI** efficient with sparse datasets.

In the experiment, results suggest the following ordering of these algorithms as running times is involved: **NOV-MFI** runs faster among two algorithms **MaxMining** and **GenMax** under all minimum supports on sparse and dense datasets.

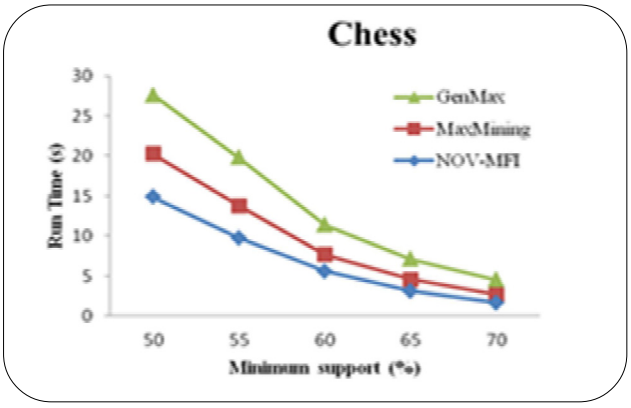

Figure 5. Run times of the three algorithms on **Chess** datasets

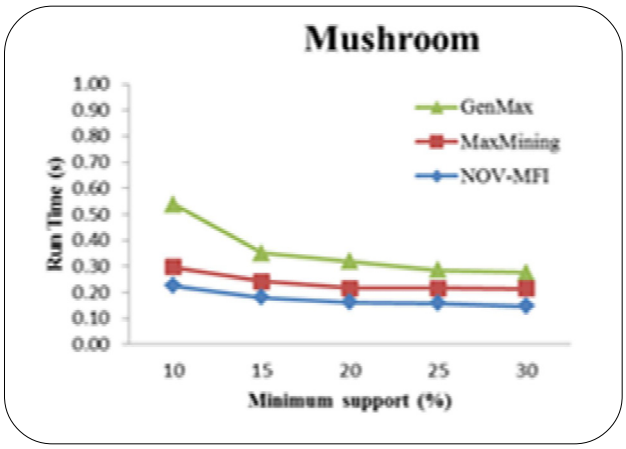

Figure 6. Run times of the three algorithms on **Mushroom** datasets

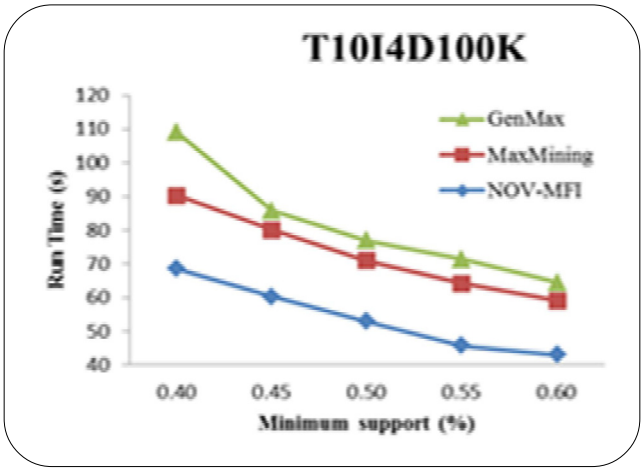

Figure 7. Run times of the three algorithms on **T10I4D100K** datasets

5. Conclusions and Future Work

In this research, we have proposed a novel algorithm for mining maximal frequent itemsets in transactional databases, consisting

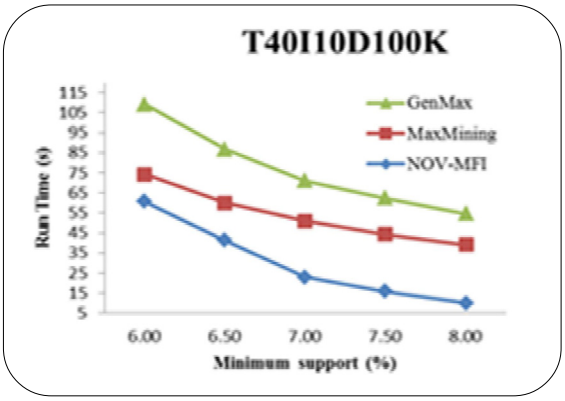

Figure 8. Run times of the three algorithms on **T40I10D100K** datasets

of two stages: the first stage, quickly compute a the *Kernel_COOC* array of co-occurrences and occurrences of kernel item in at least one transaction; the second stage, the algorithm is proposed for fast mining maximal frequent itemsets based on *Kernel_COOC* array. Besides, when using mining maximal frequent itemsets with other *minsup* value then the proposed algorithm only performs mining **MFI** based on the *Kernel_COOC* array that is calculated previously, which reduces significant processing time. Experimental results show that **NOV-MFI** algorithms have better results than other existing algorithms on both real and synthetic datasets.

We suggest the following future expansion to our work: mining maximal frequent itemsets on weighted transactional databases, as well as to develop the **NOV-MFI** algorithm on distributed computing systems.

Acknowledgment

This research is supported by University of Science; University of Social Sciences and Humanities, VNU-HCM, Vietnam.

References

[1] Agrawal, R., Imilienski, T., Swami, A. (1993). Mining association rules between sets of large databases, *In*: Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, DC, p. 207-216.

[2] Dong, J., Han, M. (2007). BitTableFI: An efficient mining frequent itemsets algorithm, *Knowledge-Based Systems*, 20 (4) 329–335.

[3] Song, W., Yang, B. (2008). Index-BitTableFI: An improved algorithm for mining frequent itemsets, *Knowledge-Based Systems*, 21, 507–513.

[4] Agarwal, R., Prasad, V. (2000). Depth first generation of long patterns, *In*: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining, p.108–118.

[5] Burdick, D., Calimlim, M., Gehrke, J. (2001). MAFIA: a maximal frequent itemset algorithm for transactional databases, *In: IEEE Intl. Conf. on Data Engineering*, p. 443–452, 2001.

[6] Gouda, K., Zaki, M. J. (2005). GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets, *In*: IEEE International Conference on Data Mining and Knowledge Discovery, 11, 1–20.

[7] Wang, H. (2015). MaxMining: A Novel Algorithm for Mining Maximal Frequent Itemset, *Applied Mechanics and Materials*, p. 1765-1768.

[8] Bajaj, S. B. (2017). Newgenmax: A Novel Algorithm for Mining Maximal Frequent Itemsets Using The Concept of Subset Checking, *International Journal of Engineering Applied Sciences and Technology*, 2 (5) 101-113.

[9] Jabbour, S., Mana, F. Z., Sais, L. (2017). On Maximal Frequent Itemsets Enumeration, *In*: Abraham A., Haqiq A., Muda A., Gandhi N. (eds) Proceedings of the Ninth International Conference on Soft Computing and Pattern Recognition (SoCPaR 2017). *Advances in Intelligent Systems and Computing*, vol 737. Springer, Cham, p. 151-160.

[10] Pan, Z., Liu, P., Yi, J. (2018). An Improved FP-Tree Algorithm for Mining Maximal Frequent Patterns, 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Changsha, Hunan, China, p. 309-312. doi:10.1109/ICMTMA.2018.00082.

[11] Philippe, F.V., Jerry, C. W. L., Bay, V., Tin, C. T., Ji, Z., Bac, L. (2017). A survey of itemset mining, Wiley Interdisc. Rew - Data Mining and Knowledge Discovery, 7(4).

[12] Phan, H., Le, B. (2018). A Novel Parallel Algorithm for Frequent Itemsets Mining in Large Transactional Databases, *In*: Perner P. (eds) Advances in Data Mining. Applications and Theoretical Aspects. ICDM 2018. *Lecture Notes in Computer Science*, Vol 10933. Springer, Cham, p. 272 – 287, 2018.