NOV-RSI: A Novel Optimization Algorithm for Mining Rare Significance Itemsets

Huan Phan[1,2,4(✉)] and Bac Le[3,4]

[1] Division of IT, VNUHCM-University of Social Sciences and Humanities, Ho Chi Minh City, Vietnam
huanphan@hcmussh.edu.vn
[2] Faculty of Mathematics and Computer Science, VNUHCM-University of Science, Ho Chi Minh City, Vietnam
[3] Faculty of IT, VNUHCM-University of Science, Ho Chi Minh City, Vietnam
lhbac@fithcmus.edu.vn
[4] Vietnam National University, Ho Chi Minh City, Vietnam

Abstract. Rare itemsets mining is an important task for potential applications such as the detection of computer attacks, fraudulent transactions in financial institutions, bioinformatics and medicine. In the traditional data mining on transaction databases, such items have no weight (equal weight, as equal to 1). However, in real world application, each item often has a different weight (the importance/significance of each item). Therefore, we need to mine weighted frequent/rare itemsets on transaction databases. In this paper, we propose an algorithm for mining rare significance itemsets based on NOT satisfy the downward closure property. We propose an efficient algorithm called NOV-RSI. The experimental results show that the proposed algorithm performs faster than other existing algorithms on both real-life datasets of UCI and synthetic datasets generated by IBM Almaden.

Keywords: Data mining · NOT satisfy the downward closure property · NOV-RSI algorithm · Rare significance itemset

1 Introduction

For more than two decades, most of the researches are for mining frequent itemsets with the weights/significance of all items are the same (*equal weight, as equal to 1*), the algorithmic approaches based on Apriori [1] and FP-Tree [2]. In addition, to speed up the execution of mining frequent itemsets, Phan et al. proposed NOV-FI [3] algorithm based on the Kernel_COOC array. Besides, rare itemsets mining is an important task for potential applications such as the detection of computer attacks, fraudulent transactions in financial institutions, bioinformatics and medicine. Algorithms such as Apriori-Inverse [4] and Rarity [5] implement an *Apriori-like* approach. Thereafter to speed up the execution of mining minimal rare itemsets, Szathmary et al. proposed Walky-G [6] algorithm based on the IT-Tree structure. But in real-world applications,

X. Yang et al. (Eds.): ADMA 2020, LNAI 12447, pp. 17–29, 2020.
https://doi.org/10.1007/978-3-030-65390-3_2

items can have different significance/importance in databases, and such databases are called weighted databases. Most algorithms for frequent weighted/significance itemsets mining are based on *satisfying the downward closure property* such as algorithms [7–9]. However, Huai et al. [10] proposed *Apriori-like* algorithms based on approach NOT *satisfy the downward closure property* (*very rare proposed algorithms following this approach*). This is a great challenge.

In this paper, we propose a novel algorithm called NOV-RSI for mining rare *significance* itemsets based on *NOT satisfying the downward closure property*. Furthermore, the proposed algorithm is easily expanded on parallel computing systems. The paper has algorithms as follows:

- *Algorithm 1:* Computing Kernel_LOOC array of co-occurrences/occurrences of kernel item in at least one transaction;
- *Algorithm 2:* Building list nLOOC_Tree based on Kernel_COOC array;
- *Algorithm 3:* NOV-RSI algorithm mining all rare significance itemset based on list of nLOOC-Tree.

This paper is organized as follows: in Sect. 2, we describe the basic concepts for mining frequent itemsets, rare itemsets (*the weights/significance of all items are the same or different*) and data structure for datasets. Some theoretical aspects of our approach relies, are given in Sect. 3. Besides, we describe our NOV-RSI algorithm to mine rare significance itemsets based on *Algorithm 1* and *Algorithm 2*. Details of implementation and experiment are discussed in Sect. 4. Finally, we conclude with a summary of our approach, perspectives and extensions of this future work.

2 Background

In this section, we present the basic concepts for mining frequent itemsets, rare itemsets (*the weights/significance of all items are the same or different*) and efficient data structure for dataset.

2.1 Mining Weighted/Significance Frequent, Rare Itemset

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m distinct items. A set of items $X = \{i_1, i_2, \ldots, i_k\}$, $\forall i_j \in I$ $(1 \leq j \leq k)$ is called an *itemset*, an itemset with k items is called a *k-itemset*. $\mathbf{D}$ be a dataset containing n transaction, a set of transaction $T = \{t_1, t_2, \ldots, t_n\}$ and each transaction $t_j = \{i_{k1}, i_{k2}, \ldots, i_{kl}\}$, $\forall i_{kl} \in I$ and a set of weight/significance $SIG = \{sig_{i1}, sig_{i2}, \ldots, sig_{im}\}$, $\forall sig_{ik} \in [0, 1]$ respective to each item.

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

$$\sup(X) = count(X)/n \tag{1}$$

Definition 2. Let $X = \{i_1, i_2, \ldots, i_k\}$, $\forall i_j \in I$ $(1 \leq j \leq k)$, significance of itemset X to compute $sig(X) = max(sig_{i1}, sig_{i2}, \ldots, sig_{ik})$.

The *significance support* of itemset X to computes as follow:

$$sigsup(\mathrm{X}) = sig(\mathrm{X}) \times sup(\mathrm{X}) \tag{2}$$

Definition 3. Let *maxsigsup* be the threshold maximum significance support value specified by user. An itemset X is a rare significance itemset if $sigsup(\mathrm{X}) < maxsigsup$, denoted ***RSI*** is the set of all the rare significance itemset.

See an Example transaction database $\mathcal{D}$ in Tables 1 and 2.

Table 1. The transaction database $\mathcal{D}$ used as our running example

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

Table 2. Items significance of $\mathcal{D}$

Item	A	B	C	D	E	F	G	H
Significance	0.55	0.70	0.50	0.65	0.40	0.60	0.30	0.80

Example 1. See Table 1 and 2. There are eight different items I = {A, B, C, D, E, F, G, H} and ten transactions T = $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$. And see Table 2 and $maxsigsup = 0.20$. Consider item X = {G}, $sup(\mathrm{G}) = 0.50$, $sig(\mathrm{G}) = 0.30$, $sigsup(\mathrm{G}) = 0.15 < maxsigsup$, we have itemset X = {G}$\in$ ***RSI***. However, itemset Y = {G, A}, $sup(\mathrm{GA}) = 0.30$, $sigsup(\mathrm{GA}) = max(\mathrm{sig_G}, \mathrm{sig_A}) = 0.70$, $sigsup(\mathrm{GA}) = sig(\mathrm{GA}) \times sup(\mathrm{GA}) = 0.70 \times 0.50 = 0.35 \geq maxsigsup$, we have item G $\notin$ ***RSI*** (DOES NOT *satisfy the downward closure property*).

Property 1. $\forall i_k \in \mathrm{I}$, $sigsup(\mathrm{i_k}) < maxsigsup$: $i_k \in$ ***RSI***.

2.2 Data Structure for Transaction Database

The binary matrix is an efficient data structure for mining frequent itemsets [3]. The process begins with the transaction 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.

3 The Proposed Algorithms

3.1 Generating Array Contain Co-occurrence Items of Kernel Item

In this section, we describe the framework of the algorithm that generates co-occurrence items of items in transaction database.

Definition 4. [3] Project set of item i_k on database $\mathcal{D}$: $\pi(i_k) = \{t_j \in \mathcal{D} \mid i_k \subseteq t_j\}$ is set of transaction contain item i_k. According to Definition 1

$$count(i_k) = |\pi(i_k)| \tag{3}$$

Definition 5. [3] Project set of itemset $X = \{i_1, i_2, \ldots, i_k\}$, $\forall i_j \in I\ (1 \leq j \leq k)$: $\pi(X) = \pi(i_1) \cap \pi(i_2) \ldots \pi(i_k)$.

$$count(X) = |\pi(X)| \tag{4}$$

Definition 6. (Reduce search space) Let $\forall i_k \in I\ (i_1 \succ i_2 \succ \ldots \succ i_m)$ items are ordered in *significance descending*, i_k is called a kernel item. Itemset $X_{lexcooc} \subseteq I$ is called co-occurrence items with the kernel item i_k, as to 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 7. (Reduce search space) Let $\forall i_k \in I\ (i_1 \succ i_2 \succ \ldots \succ i_m)$ items are ordered in *significance descending*, i_k is called a kernel item. Itemset $Y_{lexlooc} \subseteq I$ is called occurrence items with item i_k in as least one transaction, but not co-occurrence items, so that $1 \leq |\pi(i_k \cup i_j)| < |\pi(i_k)|$, $\forall i_j \in Y_{lexlooc}$. Denoted as $lexlooc(i_k) = Y_{lexlooc}$.

Algorithm Generating Array of Co-occurrence Items
This algorithm is generating co-occurrence items of items in transaction database and archived into the *Kernel_COOC* array and each element has 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 at least one transaction.

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

Algorithm 1. Generating Array of Co-occurrence Items

Input : Dataset $\mathcal{D}$
Output: *Kernel_COOC array, matrix BiM*

```
1:  foreach Kernel_COOC[k] do
2:      Kernel_COOC[k].item = ik
3:      Kernel_COOC[k].sup = 0
4:      Kernel_COOC[k].cooc = 2^m - 1
5:      Kernel_COOC[k].looc = 0
6:  foreach tj ∈ T do
7:      foreach ik ∈ tj do
8:          Kernel_COOC[k].sup ++
9:          Kernel_COOC[k].cooc = Kernel_COOC[k].cooc AND vectorbit(tj)
10:         Kernel_COOC[k].looc = Kernel_COOC[k].looc OR vectorbit(tj)
11: sort Kernel_COOC array in descending by significance
12: foreach ik ∈ tj do
13:     Kernel_COOC[k].cooc = lexcooc(ik)
14:     Kernel_COOC[k].looc = lexlooc(ik)
```

We illustrate **Algorithm 1** on Example 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 t_1 to t_{10}
Transaction t_1 = {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

The same, transaction t_{10} = {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 is as follows (Table 3):

Table 3. Kernel_COOC array are ordered in support ascending order (line 1 to 11)

Item	H	B	D	F	G	E	A	C
sup	0.10	0.20	0.20	0.30	0.50	0.70	0.80	0.80
cooc	E	A, C, E	A, C	A, C	A, C		C	A
looc		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

Execute command line 12, 13 and 14 in **Algorithm 1**:

We added the *sig* field to demonstrate items the ordered by *significance descending*. We have $looc(G) = \{B, D, E, F\}$, where $B \succ D \succ F \succ E \succ G$, so $lexlooc(G) = \{\emptyset\}$ and result on Table 4.

Table 4. Kernel_COOC array are co-occurrence items ordered in significance descending

Item	H	B	D	F	A	C	E	G
sig	0.80	0.70	0.65	0.60	0.55	0.50	0.40	0.30
sup	0.10	0.20	0.20	0.30	0.80	0.80	0.70	0.50
cooc	E	A, C, E	A, C	A,C	C	∅	∅	∅
looc	∅	G	F, G	E, G	E, G	E, G	G	∅

3.2 Generating List NLOOC-Tree

In this section, we describe the algorithm generating list *nLOOC-Tree* based on *Kernel_LOOC* array. Each node within the *nLOOC_Tree*, 2 main fields:

– nLOOC_Tree[k].*item*: kernel item k;
– nLOOC_Tree[k].*sup*: support of item k;

The framework of **Algorithm 2** is as follows:

Algorithm 2. Generating list nLOOC-Tree (*self-reduced search space*)

Input : Kernel_COOC array, matrix BiM
Output: *List nLOOC-Tree*
1: **foreach** Kernel_COOC[k] **do**
2: nLOOC_Tree[k].*item* = *Kernel_COOC*[k].*item*
3: nLOOC_Tree[k].*sup* = *Kernel_COOC*[k].*sup*
4: **foreach** $i_j \in t_l$ **do**
5: **foreach** $i_j \in$ *Kernel_COOC*[k].*looc* **do**
6: **if** $i_j \notin$ *child node of* nLOOC_Tree[k] **then**
7: Add child node i_j to nLOOC_Tree[k]
8: **else**
9: Update support of child node i_j on nLOOC_Tree[k]
10: **return** list nLOOC_Tree

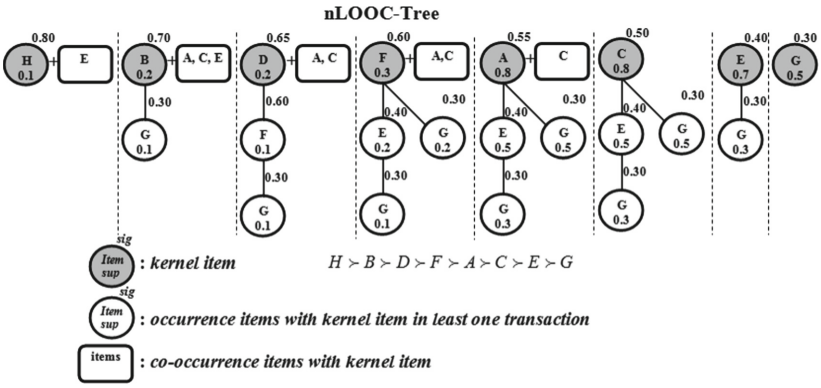

Fig. 1. List nLOOC-Tree based on Kernel_COOC array

Each **nLOOC-Tree** has the characteristics following Fig. 1:

- The height of the tree is less than or equal to the number of items that occur at least in one transaction with the kernel item (*items are ordered in significance support ascending order*).
- Single-path is an *ordered pattern* from the root node (*kernel item*) to the leaf node and the support of the pattern is the support of the leaf node ($i_k \rightarrow i_{k+1} \rightarrow \ldots \rightarrow i_\ell$).
- Sub-single-path is part of *single-path* from the root node to any node in an ordered pattern and the *sub-single-path* support is the support of the child node at the end of the *sub-single-path*.

Example 2. Consider *kernel item F*, we observe **nLOOC-Tree**(F) generating *single-path* {$\underline{\mathbf{F}} \rightarrow$ E $\rightarrow$ G}, *sup*($\underline{\mathbf{F}}$EG) = 0.10 and *sigsup*($\underline{\mathbf{F}}$EG) = 0.06; *sub-single-path* {$\underline{\mathbf{F}} \rightarrow$ E}, *sup*($\underline{\mathbf{F}}$E) = 0.20 and *sigsup*($\underline{\mathbf{F}}$E) = *sig*($\underline{\mathbf{F}}$E) $\times$ *sup*($\underline{\mathbf{F}}$E) = 0.60 $\times$ 0.20 = 0.12.

3.3 Algorithm Generating All Rare Significance Itemsets

In this section, we describe the framework of the algorithm generating all *rare significance itemsets* based on the nLOOC-Tree and Kernel_COOC.

The power set of any itemset X is the set of all subsets of X, including the empty set and X itself, variously denoted as $\wp(X)$. The set of subsets of X of cardinality greater than or equal to k is sometimes denoted by $\wp_{\geq k}(X)$.

Lemma 1. (*Generating rare significance itemset from co-occurrence items*) $\forall i_k \in I$, if $sigsup(i_k) < maxsigsup$ and itemset $X_{lexcooc}$ is set of for all element of $lexcooc(i_k)$ then $sup(i_k \cup x_{lexcooc}) < maxsigsup$, $\forall x_{lexcooc} \in \wp_{\geq 1}(X_{lexcooc})$ and itemset $\{i_k \cup x_{lexcooc}\} \in$ **RSI**, $\forall x_{lexcooc} \in \wp_{\geq 1}(X_{lexcooc})$.

Proof. According to Definition 6, (1), (2) and (3): itemset $X_{lexcooc}$ is set of co-occurrence items with the kernel item i_k, as to satisfy $\pi(i_k) \equiv \pi(i_k \cup x_{lexcooc})$, $\forall x_{lexcooc} \in \wp_{\geq 1}(X_{lexcooc})$. Therefore, we have $sup(i_k) = sup\,(i_k \cup x_{lexcooc})$, $sigsup(i_k) = sigsup(i_k \cup x_{lexcooc}) = sig(i_k \cup X_{lexcooc}) \times sup(i_k) = sig(i_k) \times sup(i_k) < maxsigsup$ and according to Definition 7: itemsets$\{i_k \cup x_{lexcooc}\} \in$ **RSI**, $\forall x_{lexcooc} \in \wp_{\geq 1}(X_{lexcooc})$■.

Example 3. See Table 4. Consider the item D as kernel item ($maxsigsup = 0.15$), we detect co-occurrence items with the item D as $lexcooc(D) = \{A, C\}$ then $\wp_{\geq 1}(\{A, C\}) = \{A, C, AC\}$, $sigsup(\underline{D}A) = sigsup(\underline{D}C) = sigsup(\underline{D}AC) = sig(D) \times sup(D) = 0.65 \times 0.20 = 0.13 < maxsigsup$ and itemsets {*DA, DC, DAC*}are rare significance itemset.

Lemma 2. (*Generating rare significance itemset from occurrence items with kernel item k in at least one transaction*) $\forall i_k \in I$, $sigsup(i_k) < maxsigsup$, $X_{lexcooc} = lexcooc(i_k) \wedge \forall sp_j \in$ nLOOC-Tree(i_k), if $sigsup(sp_j) < maxsigsup$ then $\{i_k \cup ssp_\ell\} \in$ **RSI**, $\forall ssp_\ell \in sp_j$ and $\{i_k \cup ssp_j \cup x_{lexcooc}\} \in$ **RSI**, $\forall x_{lexcooc} \in \wp_{\geq 1}(X_{lexcooc})$.

Proof. According to Definition 6, 7 and Lemma 1: we have $|\pi(i_k \cup y_{lexlooc})| < |\pi(i_k)| \equiv |\pi(i_k \cup X_{lexcooc})|$, $y_{lexlooc} \equiv sp_j \in$ nLOOC-Tree(i_k) contain of single-paths/sub-single-paths, and $sigsup(i_k \cup sp_j) < maxsigsup$, $\{i_k \cup sp_j\} \in$ **RSI**. Therefore, we have $sigsup(i_k \cup sp_j \cup x_{lexcooc}) < maxsigsup$ and $\{i_k \cup sp_j \cup X_{lexcooc}\} \in$ **RSI**, $x_{lexcooc} \in \wp_{\geq 1}(X_{lexcooc})$■.

Example 4. See Table 4 and Fig. 1. Consider the item D as kernel item ($maxsigsup = 0.15$) with $sigsup(D) = 0.13 < maxsigsup$, we detect *occurrence items with kernel item D in at least one transaction* as $Y_{lexlooc} = lexlooc(D) = \{F, G\}$; we observe **nLOOC-Tree**(D) generating *single-path* $\{\underline{D} \rightarrow F \rightarrow G\}$, $sup(\underline{D}FG) = 0.10$ and $sigsup(\underline{D}FG) = 0.65 \times 0.10 = 0.065 < maxsigsup$ then itemsets {*DF, DG, DFG*} are rare significance itemset and itemsets {*DAF, DCF, DACF, DAG, DCG, DACG, DAFG, DCFG, DACFG*} $\in$ **RSI**.

Property 2. $\forall sp_j \in$ nLOOC-Tree(i_k) $\wedge$ $sig(i_k) \times minsup\_leafnode(sp_j) \geq maxsigsup$: $\{i_k \cup sp_j\} \notin$ **RSI** (*minsup_leafnode* is minimum support value of each leaf node on single-paths in nLOOC-Tree(i_k)).

The framework of **Algorithm 3** is presented as follows:

Algorithm 3. Generating all rare significance itemsets satisfy *maxsigsup*

Input : *maxsigsup*, *Kernel_COOC array,* nLOOC_Tree

Output: RSI consists all rare significance itemsets

1: **foreach** Kernel_COOC[k] **do**
2: **if** *sigsup*(*Kernel_COOC*[*k*].*item*) < *maxsigsup* **then**
3: RSI[k] =∪{i_k ∪ Powerset(*Kernel_COOC*[*k*].*cooc*)}//*lem1*
4: *SSP* = GenPath(nLOOC_Tree(*Kernel_COOC[k].item*))
5: RSI[k] =∪{i_k ∪ Powerset(*Kernel_COOC*[*k*].*cooc*) ∪ ssp_j}, ∀ ssp_j ∈ *SSP* //*lem2*
6: **else**
7: **if** (*Kernel_COOC*[*k*].*looc* ≠∅) ∨ (*sig*(i_k)×*min_sup*(leaf node)< *maxsigsup*) **then**
8: *SSP*=GenPath(nLOOC_Tree(*Kernel_COOC[k].item*))
9: **foreach** ssp_j ∈ *SSP* **do**
10: **if** *sigsup*(i_k∪ssp_j) < *maxsigsup* **then**
11: RSI[k] =∪{ i_k∪ssp_j}
12: RSI[k] =∪{ i_k∪ssp_j ∪ Powerset(*Kernel_COOC*[*k*].*cooc*)}
13: **return** RSI

3.4 The Algorithm Diagram NOV-RSI

In this section, we represent the diagram of **NOV-RSI** algorithm for high-performance mining *rare significance itemsets*, as follows Fig. 2:

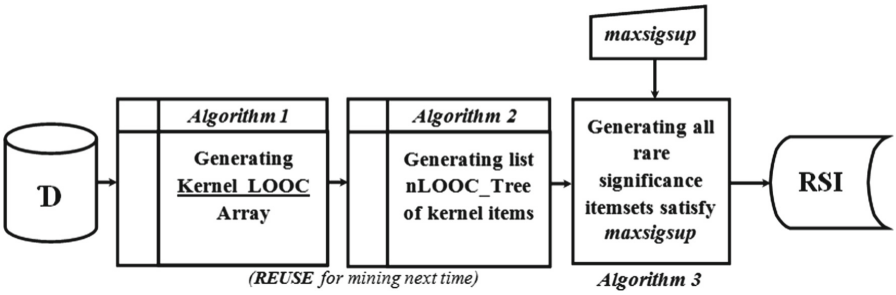

Fig. 2. The diagram algorithm for NOV-RSI.

We illustrate **Algorithm 3** on Example database in Table 1, 2 and *maxsigsup* = 0.10. After the processing **Algorithm 1** result the *Kernel_COOC* array in Table 4 and **Algorithm 2** presented the list **nLOOC_Tree** in Fig. 1.

Consider *kernel item H*, *sigsup*(H) = 0.80 × 0.10 = 0.08 < *maxsigsup* (*Lemma 1 - line 3*) generating rare significance itemset of *kernel item H* as $RSI_{[H]}$ = {(**H**; 0.08), (**H**E; 0.08)};

Consider *kernel item D*, *sigsup*(D) = 0.65 × 0.20 = 0.13 > *maxsigsup*, *lexcooc*(D) = {A, C} have $\wp_{\geq I}$({A, C}) = {A, C, AC}. We observe **nLOOC-Tree**(D) have single-path/sub-single-path {**D** → F → G}, {**D** → F} and {**D** → G}: *sigsup*(**D**FG) = 0.65 × 0.10 = 0.065 < *maxsigsup*; *sigsup*(**D**F) = 0.65 × 0.10 = 0.065 < *maxsigsup*; *sigsup*(**D**G) = 0.65 × 0.10 = 0.065 < *maxsigsup* (*Lemma 2 - line 5*) generating rare significance itemset of *kernel item D* as $RSI_{[D]}$ = {(**D**FG, 0.065), {(**D**F, 0.065), {(**D**G, 0.065), (**D**AFG, 0.065), (**D**CFG, 0.065), (**D**ACFG, 0.065), (**D**AF, 0.065), (**D**CF, 0.065), (**D**ACF, 0.065), (**D**AG, 0.065), (**D**CG, 0.065), (**D**ACG, 0.065)};

Consider *kernel item B*, *sigsup*(B) = 0.70 × 0.20 = 0.14 > *maxsigsup*, *lexcooc*(B) = {A, C, E} have $\wp_{\geq I}$({A, C, E}) = {A, C, E, AC, AE, CE, ACE}. We observe **nLOOC-Tree**(B) have single-path {**B** → G}: *sigsup*(**B**G) = 0.70 × 0.10 = 0.07 < *maxsigsup* (*Lemma 2 - line 5*) generating rare significance itemset of *kernel item B* as $RSI_{[B]}$ = {(**B**G, 0.07), (**B**AG, 0.07), (**B**CG, 0.07), (**B**EG, 0.07), (**B**ACG, 0.07), (**B**AEG, 0.07), (**B**CEG, 0.07), (**B**ACEG, 0.07)};

Consider *kernel item F*, *sigsup*(F) = 0.60 × 0.30 = 0.18 > *maxsigsup*, *lexcooc*(F) = {A, C} have $\wp_{\geq I}$({A, C}) = {A, C, AC}. We observe **nLOOC-Tree**(F) have single-path/sub-single-path {**F** → E → G}, {**F** → E} and {**F** → G}: *sigsup*(**F**EG) = 0.60 × 0.10 = 0.06 < *maxsigsup*; *sigsup*(**F**E) = 0.60 × 0.20 = 0.12 > *maxsigsup*; *sigsup*(**F**G) = 0.60 × 0.20 = 0.12 > *maxsigsup* generating rare significance itemset of *kernel item F* as $RSI_{[F]}$ = {(**F**EG, 0.06), (**F**AEG, 0.06), (**F**CEG, 0.06), (**F**ACEG, 0.06)} (*Lemma 2 - line 5*);

Consider *kernel item E*, *sigsup*(E) = 0.40 × 0.70 = 0.28 > *maxsigsup*. We observe **nLOOC-Tree**(E) have single-path {**E** → G}, *minsup_leafnode*({**E** → G}) = 0.30 and sig(E) × *minsup_leafnode*({**E** → G}) = 0.40×0.30 = 0.12 > *maxsigsup*, so $RSI_{[E]}$ = {∅} (*Property 5 - line 7*).

Consider *kernel item C* (similarly *kernel item E*), *sigsup*(C) = 0.50 × 0.80 = 0.40 ≥ *maxsigsup*. We observe **nLOOC-Tree**(C) have single-paths {**C** → E → G}, {**C** → G}, *minsup_leafnode*({**C** → E → G}, {**C** → G}) = 0.30 and sig(C) × *minsup_leafnode*{**C** → E → G}, {**C** → G}) = 0.50 × 0.30 = 0.15 > *maxsigsup*, so $RSI_{[C]}$ = {∅} (*Pro* 2 - *line 7*).

Consider *kernel item A* (similarly *kernel item C*), *sigsup*(A) = 0.55 × 0.80 = 0.44 ≥ *maxsigsup,* $RSI_{[A]}$ = {∅}.

Table 5 shows the rare significance itemsets at *maxsigsup = 0.10.*

Table 5. RSI satisfy *maxsigsup* = 0.10 (Example database in Table 1 and 2)

Kernel item	Rare significance itemsets – RSI (#RSI = 26)			
H	(**H**; 0.08)	(**H**E; 0.08)		
B	(**B**G; 0.07)	(**B**AG; 0.07)	(**B**CG; 0.07)	(**B**EG; 0.07)
	(**B**ACG; 0.07)	(**B**AEG; 0.07)	(**B**CEG; 0.07)	(**B**ACEG; 0.07)
D	(**D**F; 0.065)	(**D**G; 0.065)	(**D**AF; 0.065)	(**D**CF; 0.065)
	(**D**AG; 0.065)	(**D**CG; 0.065)	(**D**ACF; 0.065)	(**D**ACG; 0.065)
	(**D**FG; 0.065)	(**D**AFG; 0.065)	(**D**CFG; 0.065)	(**D**ACFG; 0.065)
F	(**F**EG; 0.06)	(**F**AEG; 0.06)	(**F**CEG; 0.06)	(**F**ACEG; 0.06)

4 Experiments

All experiments were performed on a PC with a Core Duo CPU T2500 2.0 GHz, 4 Gb main memory, running Microsoft Windows 7 Ultimate. All codes were compiled using C#, MVStudio 2010, .Net Framework 4.

We experimented on two instance types of datasets, see Table 6:

- 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.

Table 6. Datasets description in experiments

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

Additionally, we build one table to save the significance values of items by random real values in the range of 0 *to* 1. This is the first proposed algorithm for RSI mining based on approach DOES NOT *satisfy the downward closure property*. To evaluate the performance of the proposed algorithm, we modified (DOES NOT *satisfy the downward closure property*) the **AprioriInverse** [4] and **Rarity** [6] to mine RSI called the **WaprioriInverse** and **WRarity** algorithm. Therefore, we have compared the NOV-RSI algorithm with algorithms **WAprioriInverse** and **WRarity**.

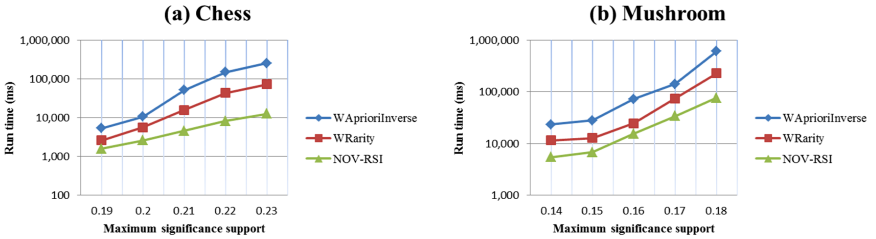

Fig. 3. Running time of the three algorithms on **Chess** and **Mushroom** datasets.

Figure 3(a) and (b) show the running time of the compared algorithms on real datasets *Chess* and *Mushroom*. The NOV-RSI algorithm runs faster than WAprioriInverse and WRarity algorithms in all maximum significance supports.

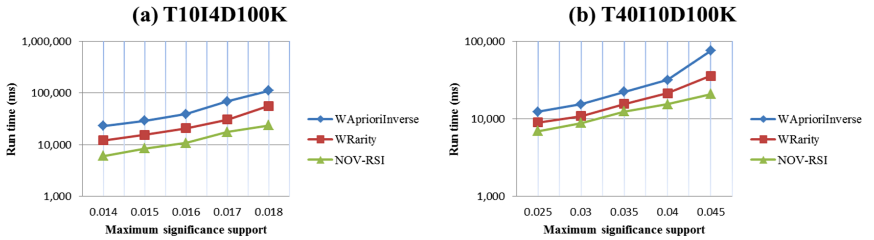

Fig. 4. Running time of the three algorithms on **T10I4D100K** and **T40I10D100K** datasets.

Figure 4(a) and (b) show the running time of the compared algorithms on synthetic datasets *T10I4KD100K* and *T40I10D100K*. The NOV-RSI algorithm runs faster than WaprioriInverse and WRarity algorithms.

In the experiment mentioned above, results suggest the following comparison of these algorithms when running time is concerned: NOV-RSI runs faster than algorithms WaprioriInverse and WRarity algorithms in all *maxsigsup* on real and synthetic datasets.

5 Conclusion

According to this paper, we presented a high-performance algorithm for mining rare significance itemsets on transaction databases, comprising three phases: *the first phase*, we quickly detect a Kernel_COOC array of co-occurrences and occurrences of kernel item in at least one transaction; *the second phase*, we build the list of nLOOC-Tree base on the Kernel_COOC and a binary matrix of dataset (*self-reduced search space*); *the last phase*, the algorithm is proposed for fast mining RSI based on nLOOC-Tree. Besides, when using mining RSI with *other maxsigsup value*, the proposed algorithm only performs mining RSI based on the nLOOC-Tree that is calculated previously (*the second phase - Algorithm 2*), there by reducing the significant processing time. The experiment's results show that the proposed algorithms perform better than other existing algorithms.

The results from the algorithm proposed: In the future, we will expand the NOV-RSI algorithm to be able to mine *rare significance itemsets* on Multi-Cores, Many-CPUs, GPU and distributed computing systems such as Hadoop, Spark.

Acknowledgements. This work was supported by the following institutions VNUHCM-University of Social Sciences and Humanities; VNUHCM-University of Science, Vietnam National University, Ho Chi Minh City, Vietnam.

References

1. Agrawal, R., Imilienski, T., Swami, A.: Mining association rules between sets of large databases. In: ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
2. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent pattern tree approach. Data Min. Knowl. Disc. **8**(1), 53–87 (2004)
3. Phan, H., Le, B.: A novel algorithm for frequent itemsets mining in transactional databases. In: Ganji, M., Rashidi, L., Fung, B.C.M., Wang, C. (eds.) PAKDD 2018. LNCS (LNAI), vol. 11154, pp. 243–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04503-6_25
4. Koh, Y.S., Rountree, N.: Finding sporadic rules using Apriori-Inverse. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 97–106. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_13
5. Troiano, L., Birtolo, C.: A fast algorithm for mining rare itemsets. In: IEEE 19th International Conference on Intelligent Systems Design and Applications, pp. 1149–1155 (2009)
6. Szathmary, L., Valtchev, P., Napoli, A., Godin, R.: Efficient vertical mining of mRI. In: 19th International Conference on Concept Lattices and Their Applications, pp. 269–280 (2012)
7. Lan, G.C., Hong, T.P., Lee, H.Y., Lin, C.W.: Tightening upper bounds for mining weighted frequent itemsets. Intell. Data Anal. **19**(2), 413–429 (2015)
8. Kiran, R.U., Kotni, A., Reddy, P.K., Toyoda, M., Bhall, S., Kitsuregawa, M.: Efficient discovery of weighted frequent itemsets in very large transactional databases: a re-visit. In: Proceedings of the IEEE International Conference on Big Data (Big Data), pp. 723–732 (2018)
9. Yun, U., Shin, H., Ryu, K.H., Yoon, E.: An efficient mining algorithm for maximal weighted frequent patterns in transactional databases. Knowl.-Based Syst. **33**, 53–64 (2012)
10. Huai, Z., Huang, M.: A weighted frequent itemsets Incremental Updating Algorithm base on hash Table. In: 3rd International Conference on Communication Software and Networks (ICCSN), pp. 201–204. IEEE (2011)