Expert Systems with Applications 42 (2015) 6648–6657

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An N-list-based algorithm for mining frequent closed patterns

Tuong Le, Bay Vo *

Division of Data Science, Ton Duc Thang University, Ho Chi Minh City, Viet Nam
Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Viet Nam

ARTICLE INFO

Article history:
Available online 2 May 2015

Keywords:
Data mining
Frequent closed pattern
N-list structure

ABSTRACT

Frequent closed patterns (FCPs), a condensed representation of frequent patterns, have been proposed for the mining of (minimal) non-redundant association rules to improve performance in terms of memory usage and mining time. Recently, the N-list structure has been proven to be very efficient for mining frequent patterns. This study proposes an N-list-based algorithm for mining FCPs called NAFCP. Two theorems for fast determining FCPs based on the N-list structure are proposed. The N-list structure provides a much more compact representation compared to previously proposed vertical structures, reducing the memory usage and mining time required for mining FCPs. The experimental results show that NAFCP outperforms previous algorithms in terms of runtime and memory usage in most cases.

1. Introduction

Frequent pattern mining (Deng, Wang, & Jiang, 2012; Dong & Han, 2007; Guil & Marín, 2013; Han, Pei, & Yin, 2000; Song, Yang, & Xu, 2008) is important in association rule mining (Agrawal & Srikant, 1994; Vo et al., 2013; Vo, Hong, et al., 2013), sequential mining (Agrawal & Srikant, 1995; Mabroukeh & Ezeife, 2010), and classification (Liu, Hsu, & Ma, 1998). Many algorithms have been proposed for mining frequent patterns, such as Apriori (Agrawal & Srikant, 1994), FP-growth (Han et al., 2000), methods based on IT-tree (Vo, Hong, & Le, 2012; Zaki & Gouda, 2003), and methods for mining frequent patterns in incremental databases (Hong, Lin, & Wu, 2009; Vo, Le, Hong, & Le, 2014). Moreover, several interesting problems related to pattern mining have been considered, such as high-utility pattern mining (Hu & Mojsilovic, 2007), concise representation of frequent itemsets (Jin, Xiang, & Liu, 2009), mining of discriminative and essential frequent patterns (Fan et al., 2008), proportional fault-tolerant frequent itemset mining (Poernomo & Gopalkrishnan, 2009), approximate frequent pattern mining (Gupta, Fang, Field, Steinbach, & Kumar, 2008), frequent weighted itemset mining (Yun, Shin, Ryu, & Yoon, 2012; Vo et al., 2013; Vo, Hong, et al., 2013), frequent pattern mining from uncertain data (Aggarwal, Li, Wang, & Wang, 2009; Bernecker, Kriegel, Renz, Verhein, & Zuefle, 2009), and erasable itemset mining (Deng & Xu, 2012; Le & Vo, 2014; Le, Vo, & Coenen, 2013; Le, Vo, & Nguyen, 2014).

In many cases, a large number of frequent patterns are identified, making them difficult to interpret and mine rules from. One solution is to use a condensed representation that reduces the overall size of the collection of patterns and rules. The two main kinds of condensed representation are frequent closed patterns (FCPs) (Pasquier, Bastide, Taouil, & Lakhal, 1999; Pei, Han, & Mao, 2000; Zaki & Hsiao, 2002, 2005) and maximal frequent patterns (MFPs) (Gouda & Zaki, 2005). A collection of FCPs can be used to deduce all frequent patterns. The small number of FCPs compared to that of all frequent patterns greatly reduces memory and computation time requirements. Many algorithms have been proposed for mining FCPs, including the Close algorithm (Pasquier et al., 1999), CLOSET (Pei et al., 2000), CLOSET+ (Grahne & Zhu, 2005), CHARM (Zaki & Hsiao, 2002), dCHARM (Zaki & Hsiao, 2005), DBV-Miner (Vo et al., 2012), and DCI_PLUS (Sahoo, Das, & Goswami, 2015).

Recently, Deng et al. (2012) proposed the N-list structure and the PrePost algorithm for mining frequent patterns. The N-list structure helps to reduce the mining time and memory usage because it is much more compact than previously proposed vertical structures. Additionally, the support of a candidate FCPs called frequent pattern can be determined through N-list intersection operations. Vo and Coenen (2014a) proposed an improved variation of the PrePost algorithm to enhance the performance of the frequent itemset mining process. The N-list structure is also used for mining erasable itemsets (Deng & Xu, 2012; Le et al., 2013) and top-rank-*k* frequent itemsets (Deng, 2014; Huynh, Le, Vo, &

* Corresponding author at: Division of Data Science, Ton Duc Thang University, Ho Chi Minh City, Viet Nam.
E-mail addresses: lecungtuong@tdt.edu.vn, tuonglecung@gmail.com (T. Le), vodinhbay@tdt.edu.vn, bayvodinh@gmail.com (B. Vo).

http://dx.doi.org/10.1016/j.eswa.2015.04.048
0957-4174/

Le, 2015). The above studies showed that the N-list structure is effective for mining patterns.

The present study proposes an N-list-based algorithm for mining NAFCP. The main contributions of this study are as follows. A number of theorems for fast determining FCPs based on the N-list structure are proposed. NAFCP is proposed based on these theorems. NAFCP inherits the advantages of previous efficient algorithms (Deng et al., 2012; Vo, Le, Coenen, & Hong, in press). Finally, experiment studies are conducted to show the effectiveness of NAFCP compared with a number of existing algorithms.

The rest of this paper is organized as follows. Section 2 presents the related work. The definitions of FCPs, PPC-tree, and N-list structure are presented in Section 3. Theorems concerning the N-list structure, FCPs, and NAFCP are presented in Section 4. Section 5 describes the operation of NAFCP. Section 6 shows the results of experiments comparing the runtime and memory usage of NAFCP with those of dCHARM (Zaki & Hsiao, 2005) and DCI_PLUS (Sahoo et al., 2015) to show its effectiveness. Finally, Section 7 summarizes the results and offers some future research topics.

Table 1
Example transaction dataset.

Transaction	Items
1	*A*, *C*, *T*, *W*
2	*C*, *D*, *W*
3	*A*, *C*, *T*, *W*
4	*A*, *C*, *D*, *W*
5	*A*, *C*, *D*, *T*, *W*, *E*
6	*C*, *D*, *T*, *E*

2. Related work

Most previously proposed algorithms for mining FCPs can be categorized as being either (i) generate-and-test, (ii) divide-and-conquer, or (iii) hybrid approaches. The generate-and-test (Apriori-based) approach uses a level-wise search to discover FCPs. A well-known example is the Close

Algorithm 1. PPC-tree creation algorithm

Input: DB, $minSup$

Output: $\mathcal{R}$, I_1, H_1, and *threshold*

Method name: Construct_PPC_tree

1 scan DB to find I_1 and their frequencies
2 sort I_1 in order of descending frequency
3 create H_1, the hash table of I_1
4 create the root of a PP-tree, $\mathcal{R}$, and label it as 'null'
5 let *threshold* = $\lceil minSup \times n \rceil$
6 for each transaction $T \in DB$ do
7 begin for
8 remove the items whose supports do not satisfy the *threshold*
9 sort items in order of descending frequency
10 Insert_Tree(T, $\mathcal{R}$)
11 end for
12 traverse PP-tree to generate *pre* and *post* values associated with each node
13 return $\mathcal{R}$, I_1, H_1, and *threshold*

1 Procedure **Insert_Tree**(T, $\mathcal{R}$)
2 while (T is not null) do
3 begin while
4 $t \leftarrow$ the first item of T and $T \leftarrow T \setminus t$
5 if $\mathcal{R}$ has a child N such that $n(N) = t$ then
6 $f(N) = f(N)+1$
7 else
8 create a new node N with $n(N) = t$, $f(N) = 1$ and $childs(\mathcal{R}) = N$
9 Insert_Tree(T, N)
10 end while

Fig. 1. PPC-tree creation algorithm.

algorithm (Pasquier et al., 1999). The divide-and-conquer approach adopts a divide-and-conquer strategy and uses some compact data structures to efficiently mine FCPs. Examples are CLOSET (Pei et al., 2000) and CLOSET+ (Grahne & Zhu, 2005). The hybrid approaches integrates the previous two. Typically, the database is first transformed into a vertical data format. The approach then utilizes some pruning properties to quickly prune non-closed patterns. Examples are CHARM, dCHARM (Zaki & Hsiao, 2005), DBV-Miner (Vo et al., 2012), and DCI_PLUS (Sahoo et al., 2015).

The Nodelist (Deng & Wang, 2010) and N-list (Deng et al., 2012) data structures have recently been proposed for mining frequent patterns. They are based on PPC-tree, which stores the sufficient information associated with frequent 1-itemsets. N-list and Nodelist are a set of sorted nodes in PPC-tree. Nodelists are built from descendant nodes whereas N-lists are built from ancestor nodes. They both have two important properties: (i) an N-list or Nodelist associated with a frequent $(k+1)$-itemset can be constructed by joining the N-lists or Nodelists of its subset with length k, and (ii) the support of an itemset can be determined by summing the counts registered in the nodes of its N-list or Nodelist. The PrePost (based on N-list) and PPV (based on Nodelist) algorithms are faster than state-of-the-art algorithms, including dEclat and Eclat_goethals. N-list is better than Nodelist because (i) the length of the N-list of an itemset is much smaller than that of its Nodelist, and (ii) N-lists have a property called the single path property. Therefore, the PrePost algorithm is more efficient than the PPV algorithm. Based on PPC-tree, NC_set (Deng & Xu, 2012; Le et al., 2013), a structure similar to N-list and Nodelist, has been proposed for mining erasable itemsets. In addition, Nodelist and N-list have been used to improve the performance of mining top-rank-k frequent patterns by Deng (2014) and Huynh et al. (2015), respectively. Deng and Lv (2014) proposed the Nodeset structure, where a node is encoded only by pre-order or post-order code in the PPC-tree, for mining frequent patterns. Extensive experiments have shown that the Nodeset structure is more efficient than N-list and Nodelist structures. Deng and Lv (2015) recently proposed the PrePost + algorithm for mining frequent patterns based on the N-list structure and children–parent equivalence pruning. Their results show that the N-list structure is very effective for mining patterns.

The present study proposes an algorithm based on the N-list structure for mining FCPs. Experiments are conducted to compare the runtime and memory usage of the proposed algorithm with those of dCHARM (Zaki & Hsiao, 2005) and DCI_PLUS (Sahoo et al., 2015) to show the effectiveness of the proposed algorithm.

3. Basic principles

3.1. Frequent closed patterns

Consider a transaction dataset *DB* that comprises n transactions such that each transaction contains a number of items. An example *DB* featuring $n = 6$ transactions is presented in Table 1. This *DB* is used for illustrative purposes throughout the remainder of this paper. The support of a pattern X, denoted by $\sigma(X)$, where $X \in I$ and I is the set of all items in *DB*, is the number of transactions

Algorithm 2. Improved N-list intersection function

input: PS_1, PS_2

output: PS_3 and *frequency*

Method name: N-list-Intersection

```
1  PS3 ← ∅
2  let sF be the sum of frequencies of PS1 and PS2
3  let i = 0, j = 0, and frequency = 0
4  while i < |PS1| and j < |PS2| do
5  begin while
6    if pre(PS1[i]) < pre(PS2[j]) then
7      if post(PS1[i]) > post(PS2[j]) then
8        if |PS3| > 0 and pre(PS3[|PS3| -1]) = pre(PS1[i]) then
9          f(PS3[|PS3|-1]) += f(PS2[j])
10       else
11         add the tuple ⟨pre(PS1[i]), post(PS1[i]), f(PS2[j])⟩ to PS3
12       frequency += f(PS2[j++])
13     else
14       sF = sF - f(PS1[i++])
15   else
16     sF = sF - f(PS2[j++])
17   if sF < threshold then // use early abandoning strategy
18     return null // stop the procedure
19 end while
20 return PS3 and frequency
```

Fig. 2. Improved N-list intersection function.

containing all the items in X. A pattern with k items is called a k-pattern and I_1 is the set of frequent 1-patterns sorted in order of descending frequency. For convenience, pattern $\{A,C,W\}$ is written as ACW.

Let $minSup$ (minimum support threshold) be a given threshold. A pattern X is called a frequent pattern if $\sigma(X) \geqslant minSup \times n$. A frequent pattern is called an FCP if none of its supersets has the same support. For example, consider the example DB in Table 1 and let $minSup = 50\%$. AW and ACW are two frequent patterns because $\sigma(AW) = \sigma(ACW) = 4 > 50\% \times 6$. However, AW is not an FCP because ACW is its superset and has the same support.

3.2. PPC-tree

Deng and Xu (2012) presented an FP-tree-like structure called the PPC-tree. Each node in the tree holds five values, namely $n(N_i)$, $f(N_i)$, $childs(N_i)$, $pre(N_i)$, and $post(N_i)$, which are the frequent 1-patterns in I_1, the frequency of this node, the set of child nodes associated with this node, the order number of the node when traversing the tree in pre-order form, and the order number when traversing the tree in post-order form, respectively. Deng and Xu (2012) also proposed a PPC-tree construction algorithm (Fig. 1). An example of PPC-tree creation is presented in Section 4.1.

3.3. N-list structure

N-list associated with 1-patterns: The PP-code of each node N_i in a PPC-tree comprises a tuple of the form $C_i = \langle pre(N_i), pre(N_i), f(N_i)\rangle$.

The N-list associated with a pattern A, denoted by $NL(A)$, is the set of PP-codes associated with nodes in the PPC-tree that equal A. Thus:

Algorithm 3. NAFCP

input: A dataset DB and $minSup$

output: $FCIs$, the set of all frequent closed patterns

```
1   Construct_PPC_tree(DB, minSup) to generate R, I1, H1, and threshold
2   Generate-N-list(R)
3   Find-FCIs(I1)
4   return FCIs

1   Procedure Generate-N-list(R)
2   Let C ← ⟨pre(R), post(R), f(R)⟩
3   Let H = H1[n(R)] is the node of n(R) in H1
4   Add C to NL(H)
5   f(H) += f(R)
6   for each child in childs(R) do
7       Generate-N-list(child)

1   Procedure Find-FCIs(Is)
2   for i ← |Is| - 1 to 0 do
3       FCIs_next ← ∅
4       for j ← i-1 to 0 do
5       begin for
6           if N-list-check(NL(Is[i]), NL(Is[j])) = true then
7               if f(Is[i]) = f(Is[j]) then
8                   Is[i] = Is[i] ∪ Is[j]
9                   remove Is[j]
10                  i--
11              else
12                  Is[i] = Is[i] ∪ Is[j]
13                  for each f in FCIs_next
14                      update f = f ∪ Is[j]
15                  continue
16      end for
17      FCI ← Is[i] ∪ Is[j]
```

Fig. 3. NAFCP algorithm.

```
18    NL(FCI) and f(FCI) ← N-list-intersection(NL(Is[i]), NL(Is[j]))
19    if f(FCI) ≥ threshold then
20      add FCI to FCIs_next
21    if Subsumption-check(Is[i]) = false then
22      add Is[i] to FCIs
23      add index of FCI in FCIs to Hash[f(FCI)]
24  Find-FCIs(FCIs_next)

1   Function N-list-check(N1, N2)
2   let i = 0 and j = 0
3   while j < |N1| and i < |N2| do
4     if pre(N2[i]) < pre(N1[j]) and post(N2[i]) > post(N1[j]) then
5       j++
6     else
7       i++
8   if j = |N1| then
9     return true
10  return false

1   Function Subsumption-check(FCI)
2   let Arr ← Hash[f(FCI)]
3   for i ← 0 to |Arr|-1 do
4     if FCI ⊆ FCIs[Arr[i]] then
5       return true
6   return false
```

Fig. 3 (*continued*)

$$NL(A) = \bigcup_{\{N_i \in \mathscr{R} | n(N_i)=A\}} C_i \quad (1)$$

where C_i is the PP-code associated with N_i. The support for A, $\sigma(A)$, is calculated by:

$$\sigma(A) = \sum_{C_i \in NL(A)} f(C_i) \quad (2)$$

N-list associated with k-patterns: Let XA and XB be two $(k-1)$-patterns with the same prefix X (X can be an empty set) such that A is before B according to the I_1 ordering. $NL(XA)$ and $NL(XB)$ are two N-lists associated with XA and XB, respectively. Vo et al. (2014) presented an effective method for determining the N-list associated with XAB as follows: for each PP-code $C_i \in NL(XA)$ and $C_j \in NL(XB)$, if C_i is an ancestor of C_j and:

1. If $\exists C_z \in NL(XAB)$ such that $pre(C_z) = pre(C_i)$ and $post(C_z) = post(C_i)$, then the algorithm updates the frequency count of C_z, $f(C_z) = f(C_z) + f(C_i)$.
2. Otherwise, the algorithm adds $\langle pre(C_i), post(C_i), f(C_j)\rangle$ to $NL(XAB)$.

The support of XAB, denoted by $\sigma(XAB)$, is calculated as follows:

$$\sigma(XAB) = \sum_{C_i \in NL(XAB)} f(C_i) \quad (3)$$

Vo et al. (2014) also presented an algorithm for determining the intersection of two N-lists, shown in Fig. 2.

4. N-list-based algorithm for mining FCPs

Prior to introducing the proposed algorithm, a number of basic concepts are proposed. Let XA and XB be two frequent patterns (XA is before XB in frequent 1-pattern order and X can be an empty set), and $NL(XA)$ and $NL(B)$ be the N-lists associated with XA and XB, respectively.

Definition 1 (*N-list subset operation*). $NL(XB) \subseteq NL(XA)$ if and only if $\forall C_i \in NL(XB)$, $\exists C_j \in NL(XA)$ such that C_j is an ancestor of C_i.

Theorem 1. If $NL(XB) \subseteq NL(XA)$, then XB is not a closed pattern.

Proof. According to the method for determining the N-list associated with a k-pattern presented in Section 3.3 and $NL(XB) \subseteq NL(XA)$, we have:

$$NL(XAB) = \bigcup \langle pre(C_i), post(C_i), f(C_j)\rangle \quad (4)$$

where $C_i \in NL(XA)$, $C_j \in NL(XB)$, and C_i is an ancestor of C_j. Therefore, $\sigma(XAB) = \sum_{C_i \in NL(XAB)} f(C_i) = \sum_{C_j \in NL(XB)} f(C_j) = \sigma(XB)$. Based on the FCP definition, XB is not a closed pattern. Theorem 1 is proven. □

Theorem 2. If $NL(XB) \subseteq NL(XA)$ and $\sigma(XB) = \sigma(XA)$, then XA and XB are not closed patterns.

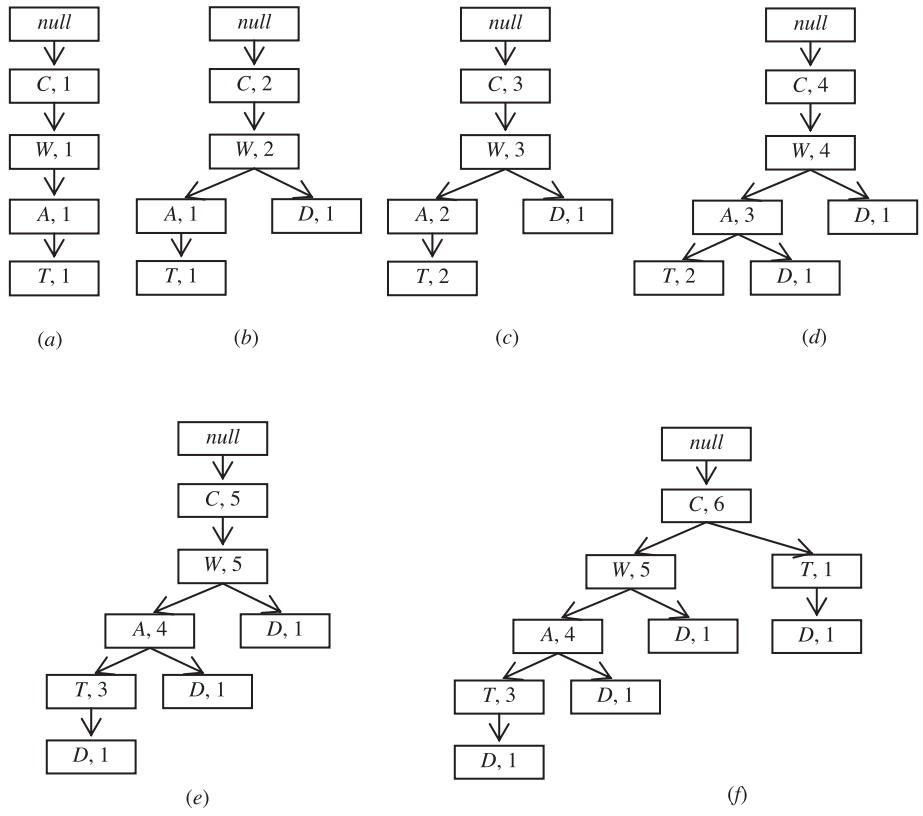

Fig. 4. Illustration of creation of PPC-tree using example *DB* with *minSup* = 50%.

Proof. According to Theorem 1, we have $\sigma(XAB) = \sigma(XB)$. We also have $\sigma(XB) = \sigma(XA)$ according to hypothesis $\Rightarrow \sigma(XAB) = \sigma(XB) = \sigma(XA)$. Based on the FCP definition, *XA* and *XB* are not FCPs. Therefore, Theorem 2 is proven. □

We utilize Theorems 1 and 2 in NAFCP, presented in Fig. 3, as follows: before the algorithm combines two patterns *XB* and *XA*, if $NL(XB) \subseteq NL(XA)$, then two cases are considered:

1. If $\sigma(XB) = \sigma(XA)$, then the algorithm updates node *XB* to *XAB* and removes *XA* because *XB* and *XA* are not closed patterns according to Theorem 2.
2. Otherwise, the algorithm updates node *XB* to *XBA* because only *XB* is not a closed pattern according to Theorem 1.

NAFCP first creates the PPC-tree, and then traverses it to generate the N-lists associated with the frequent 1-itemsets. Then, a divide-and-conquer strategy is used to mine FCPs. NAFCP calls the **Find-FCIs** procedure recursively to create FCP candidates. NAFCP checks whether these two nodes satisfy Theorems 1 and 2. If so, the algorithm combines the elements without combining the N-lists and continues to the next element. If not, the algorithm creates a child node, *X*, representing the elements. If *X* is frequent and does not exist in the current list of FCPs, the algorithm adds it to the results and the list of candidates for the next level of the recursive process.

5. Illustrative example

5.1. PPC-tree creation

The example *DB* from Table 1 is used with *minSup* = 50% to illustrate the operation of this algorithm. First, NAFCP removes all items whose frequency does not satisfy the *minSup* threshold and sorts the remaining items in descending order of frequency (Table 2).

The algorithm then inserts, in turn, the remaining items in each transaction into the PPC-tree, as shown in Fig. 4.

The algorithm traverses the full tree, shown in Fig. 4(f), to generate the required *pre* and *post* values associated with each node. The final PPC-tree is presented in Fig. 5.

Table 2
Example *DB* after removal of infrequent 1-patterns and sorting in descending order of frequency.

Transaction	Ordered frequent items
1	*C, W, A, T*
2	*C, W, D*
3	*C, W, A, T*
4	*C, W, A, D*
5	*C, W, A, T, D*
6	*C, T, D*

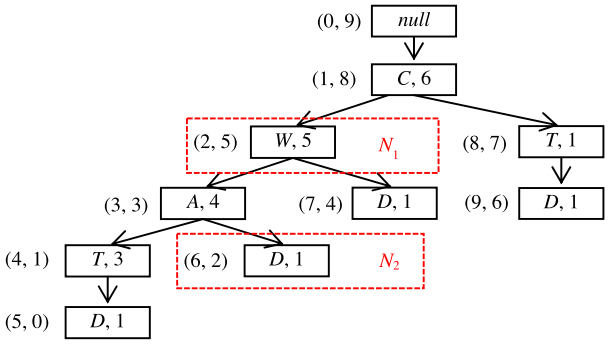

Fig. 5. Final PPC-tree created from example *DB* with *minSup* = 50%.

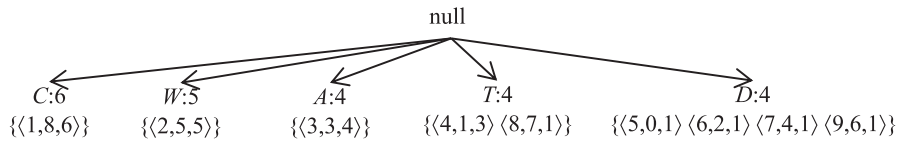

Fig. 6. Frequent 1-patterns and their N-lists.

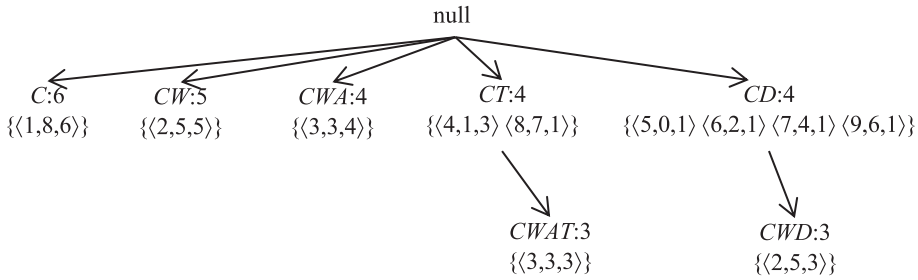

Fig. 7. FCPs identified using NAFCP for example *DB* with *minSup* = 50%.

Table 3
Statistical summary of experimental datasets.

Dataset	Type	# of trans	# of items
Accidents	Sparse	340,183	468
Chess	Dense	3196	76
Connect	Dense	67,557	130
Mushroom	Dense	8124	120

5.2. Mining FCPs using N-list structure

NAFCP uses the **Generate-N-list** function with the final PPC-tree in Fig. 5 to create the N-lists associated with the frequent 1-patterns. Fig. 6 shows the frequent 1-patterns and their N-lists.

Then, NAFCP calls the **Find-FCIs** procedure recursively to create FCP candidates. It combines each element in the input with the remaining elements in reverse order. NAFCP checks whether these two nodes satisfy Theorems 1 and 2. If so, the algorithm combines the elements without combining the N-lists and continues to the next element. If not, the algorithm creates a child node, X, representing the elements. If X is frequent and the current results (the list of FCPs) do not include an FCP, Y, such that $X \subseteq Y$ and $\sigma(X) = \sigma(Y)$, the algorithm adds X to the current results and the list of candidates for the next level of the recursive process. Fig. 7 shows the list of FCPs obtained by NAFCP using the example *DB* with *minSup* = 50%.

6. Performance studies

All experiments presented in this section were performed on a laptop with an Intel Core i3-3110 M 2.4-GHz CPU and 4 GB of RAM. The operating system was Microsoft Windows 8 Professional (64-bit). All the programs were coded in C# in Microsoft Visual Studio 2012 and run on the Microsoft.Net Framework (version 4.5.50709). The experiments were conducted on the following UCL datasets: Accidents, Chess, Connect, and Mushroom.[1] Some statistics concerning these datasets are shown in Table 3. The Accidents dataset reports the traffic accidents from the National Institute of Statistics. It contains 340,184 transactions and 468 items. The Chess dataset is derived from the moves of chess games. It contains 3196 transactions and 76 items. The Mushrooms dataset contains 8124 transactions, each of which describes 23 attributes of gilled mushrooms. Each transaction is related to 23 items, with a total of 120 items. The Connect dataset contains 67,557 transactions and 130 items.

6.1. Execution time

Most of the computational resources required by NAFCP are used to create the PPC-tree; therefore, given large *minSup* values, the algorithm is not better than dCHARM and DCI_PLUS (Fig. 8 with *minSup* = 60%, 50%, and 40%; Fig. 10 with *minSup* = 80% and 70%; and Fig. 11 with *minSup* = 10%). However, for small thresholds, NAFCP is much faster than dCHARM (Figs. 8–11). This can be explained as follows. Using N-lists, the algorithm compresses the input data. The N-lists associated with 1-patterns are much smaller than the diffsets associated with the 1-patterns. Therefore, NAFCP is generally more efficient than dCHARM and DCI_PLUS.

6.2. Memory usage

For large thresholds, the difference in the memory usage of dCHARM, DCI_PLUS, and NAFCP is very small (Fig. 12 with *minSup* = 60%, 50%, and 40%; Fig. 13 with *minSup* = 75% and 70%; Fig. 14 with *minSup* = 80% and 70%; and Fig. 15 with *minSup* = 10% and 8%). However, for small thresholds, NAFCP clearly outperforms dCHARM and DCI_PLUS in terms of memory usage (Figs. 12–15).

7. Conclusion and future work

This paper proposed an effective algorithm for mining FCPs using the N-list structure in terms of mining time and memory usage. First, two theorems supporting NAFCP, based on the N-list structure, were developed. Then, using these theorems, and inheriting the advantages of previous algorithms (Deng et al., 2012; Vo et al., 2014), NAFCP was proposed. Finally, in order to confirm the effectiveness of the proposed algorithm in terms of runtime and memory usage, experiments were conducted on a number of datasets using NAFCP, dCHARM, and DCI_PLUS. The experimental results show that NAFCP outperforms dCHARM and DCI_PLUS in

[1] Downloaded from http://fimi.cs.helsinki.fi/data/.

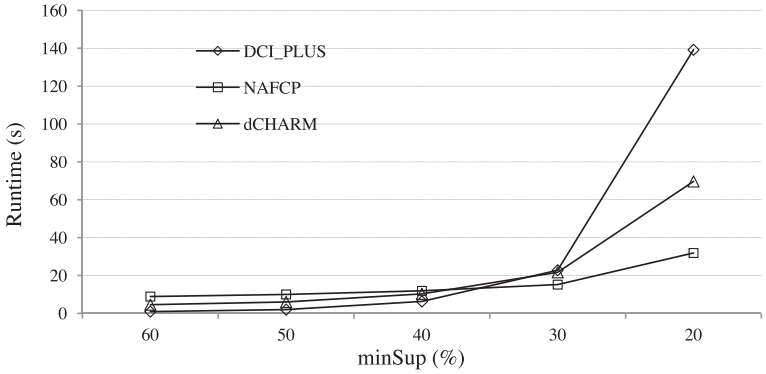

Fig. 8. Execution time of NAFCP, dCHARM, and DCI_PLUS for Accidents dataset.

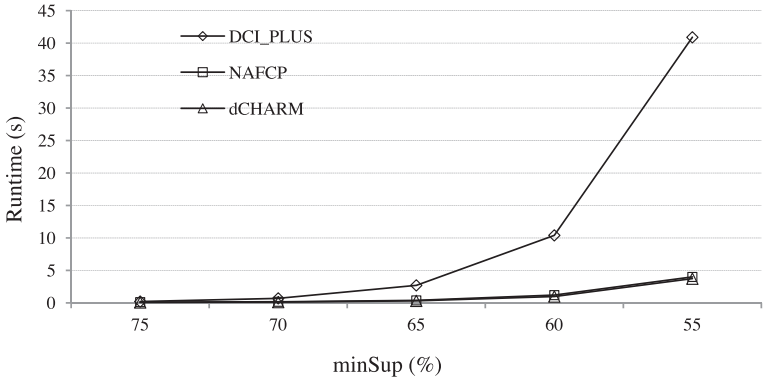

Fig. 9. Execution time of NAFCP, dCHARM, and DCI_PLUS for Chess dataset.

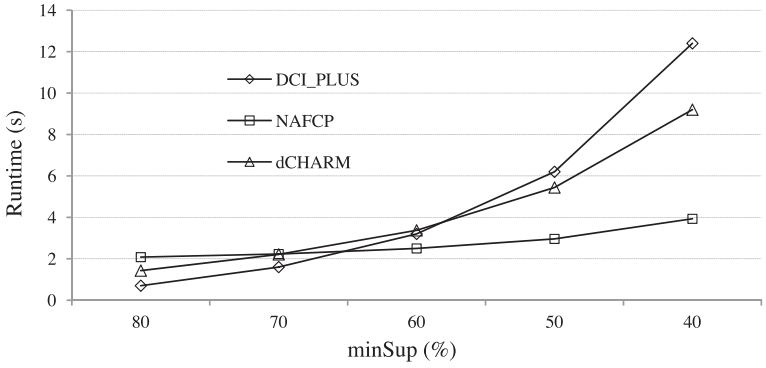

Fig. 10. Execution time of NAFCP, dCHARM, and DCI_PLUS for Connect dataset.

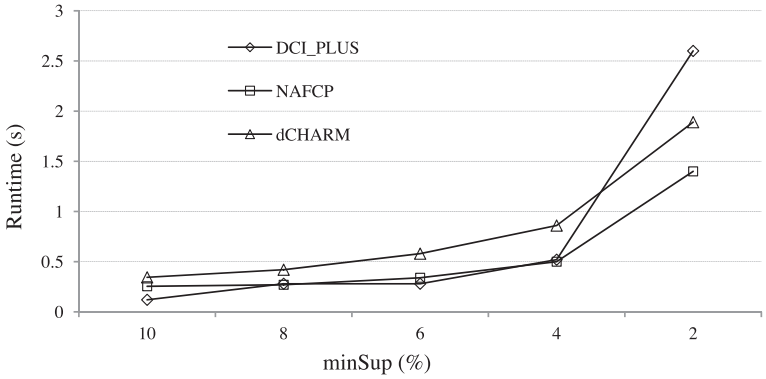

Fig. 11. Execution time of NAFCP, dCHARM, and DCI_PLUS for Mushroom dataset.

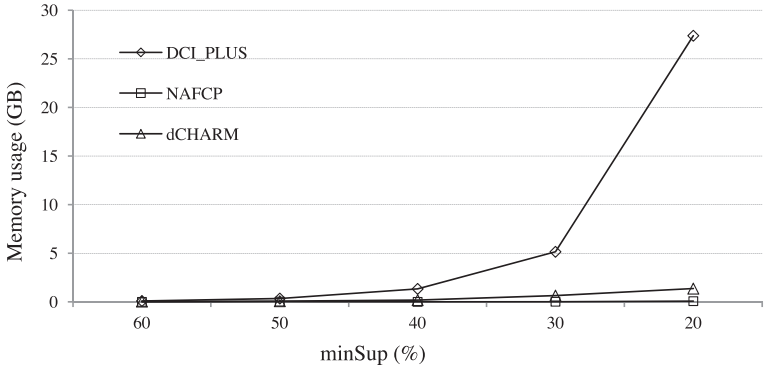

Fig. 12. Memory usage of NAFCP, dCHARM, and DCI_PLUS for Accidents dataset.

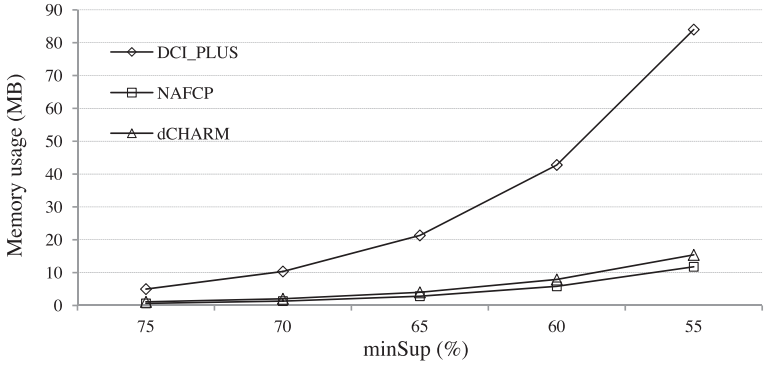

Fig. 13. Memory usage of NAFCP, dCHARM, and DCI_PLUS for Chess dataset.

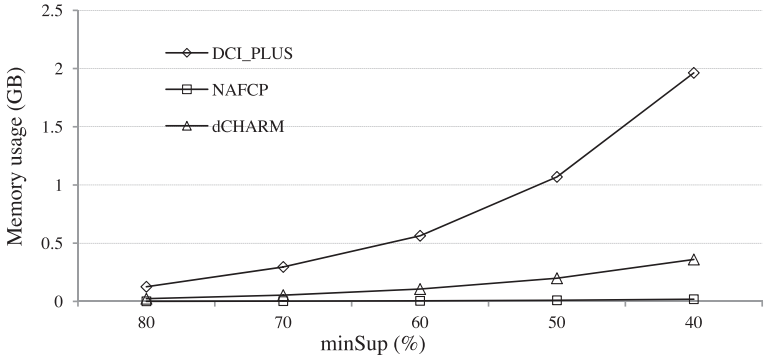

Fig. 14. Memory usage of NAFCP, dCHARM, and DCI_PLUS for Connect dataset.

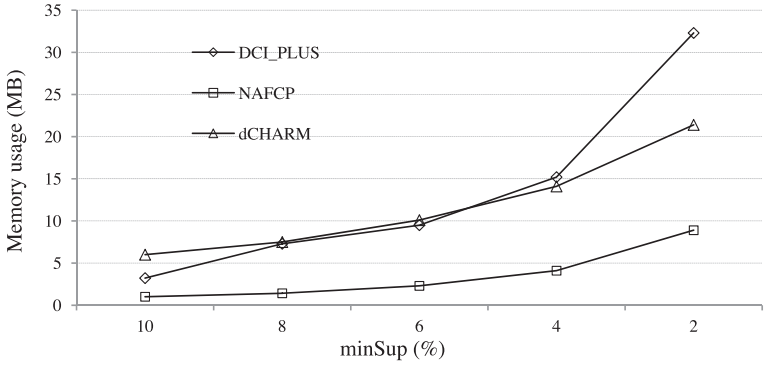

Fig. 15. Memory usage of NAFCP, dCHARM, and DCI_PLUS for Mushroom dataset.

terms of runtime and memory usage in most cases. Using the proposed algorithm, expert and intelligent systems that use FCPs can improve their performance.

This study has some limitations. An algorithm for mining FCPs was proposed but it was not incorporated into any expert and intelligent systems. In addition, although the mining time and memory usage of the proposed algorithm are better than those of a number of existing algorithms, memory usage for large datasets can still be improved.

For future work, the authors intend to focus on optimizing the N-list structure to improve mining time and memory usage of mining FCPs. The N-list structure will be applied to the mining of frequent maximal patterns, top-rank-*k* FCPs, and top-rank-*k* frequent maximal patterns. We will also apply the proposed method to the mining of patterns with constraints.

Acknowledgment

This research was funded by the Foundation for Science and Technology Development of Ton Duc Thang University (FOSTECT), website: http://fostect.tdt.edu.vn, under Grant FOSTECT.2014.BR.07.

References

Aggarwal, C. C., Li, Y., Wang, J., & Wang, J. (2009). Frequent pattern mining with uncertain data. In *SIGKDD'09* (pp. 29–38).

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

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *ICDE'95* (pp. 3–14).

Bernecker, T., Kriegel, H., Renz, M., Verhein, F., & Zuefle, A. (2009). Probabilistic frequent itemset mining in uncertain databases. In *SIGKDD'09* (pp. 119–127).

Deng, Z. H. (2014). Fast mining top-rank-*k* frequent patterns by using Node-lists. *Expert Systems with Applications, 41*(4), 1763–1768.

Deng, Z. H., & Lv, S. L. (2015). PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via children-parent equivalence pruning. *Expert Systems with Applications, 42*(13), 5424–5432.

Deng, Z. H., & Lv, S. L. (2014). Fast mining frequent itemsets using Nodesets. *Expert Systems with Applications, 41*(10), 4505–4512.

Deng, Z. H., & Wang, Z. H. (2010). A new fast vertical method for mining frequent itemsets. *International Journal of Computational Intelligence Systems, 3*(6), 733–744.

Deng, Z., Wang, Z., & Jiang, J. J. (2012). A new algorithm for fast mining frequent itemsets using N-lists. *SCIENCE CHINA Information Sciences, 55*(9), 2008–2030.

Deng, Z. H., & Xu, X. R. (2012). Fast mining erasable itemsets using NC_sets. *Expert Systems with Applications, 39*(4), 4453–4463.

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

Fan, W., Zhang, K., Cheng, H., Gao, J., Yan, X., Han, J., Yu, P., & Verscheure, O. (2008). Direct mining of discriminative and essential frequent patterns via model-based search tree. In *SIGKDD'08* (pp. 230–238).

Gouda, K., & Zaki, M. J. (2005). GenMax: An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery, 11*(3), 223–242.

Grahne, G., & Zhu, J. (2005). Fast algorithms for frequent itemset mining using FP-trees. *IEEE Transactions on Knowledge and Data Engineering, 17*, 1347–1362.

Guil, F., & Marín, R. (2013). A Theory of evidence-based method for assessing frequent patterns. *Expert Systems with Applications, 40*, 3121–3127.

Gupta, R., Fang, G., Field, B., Steinbach, M., & Kumar, V. (2008). Quantitative evaluation of approximate frequent pattern mining algorithms. In *SIGKDD'08* (pp. 301–309).

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In *SIGMOD'00* (pp. 1–12).

Hong, T. P., Lin, C. W., & Wu, Y. L. (2009). Maintenance of fast updated frequent pattern trees for record deletion. *Computational Statistics and Data Analysis, 53*(7), 2485–2499.

Hu, J., & Mojsilovic, A. (2007). High-utility pattern mining: A method for discovery of high-utility item sets. *Pattern Recognition, 40*(11), 3317–3324.

Huynh, Q., Le, T., Vo, B., & Le, B. (2015). An efficient and effective algorithm for mining top-rank-*k* frequent patterns. *Expert Systems with Applications, 42*(1), 156–164.

Jin, R., Xiang, Y., & Liu, L. (2009). Cartesian contour: A concise representation for a collection of frequent sets. In *SIGKDD'09* (pp. 417–425).

Le, T., Vo, B., & Coenen, F. (2013). An efficient algorithm for mining erasable itemsets using the difference of NC-Sets. In *IEEE SMC'13* (pp. 2270–2274). Manchester, UK.

Le, T., & Vo, B. (2014). MEI: An efficient algorithm for mining erasable itemsets. *Engineering Applications of Artificial Intelligence, 27*, 155–166.

Le, T., Vo, B., & Nguyen, G. (2014). A survey of erasable itemset mining algorithms. *WIREs Data Mining Knowledge Discovery, 4*(5), 356–379.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In *SIGKDD'98* (pp. 80–86).

Mabroukeh, N. R., & Ezeife, C. I. (2010). A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys, 43*(1), 1–41.

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

Pei, J., Han, J., & Mao, R. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the 5th ACM-SIGMOD workshop on research issues in data mining and knowledge discovery* (pp. 21–30). Dallas, Texas, USA.

Poernomo, A., & Gopalkrishnan, V. (2009). Towards efficient mining of proportional fault-tolerant frequent itemsets. In *SIGKDD'09* (pp. 697–705).

Sahoo, J., Das, A. K., & Goswami, A. (2015). An effective association rule mining scheme using a new generic basis. *Knowledge and Information Systems, 43*(1), 127–156.

Song, W., Yang, B., & Xu, Z. (2008). Index-BitTableFI: An improved algorithm for mining frequent itemsets. *Knowledge-Based Systems, 21*(6), 507–513.

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

Vo, B., Le, T., Coenen, F., & Hong, T. P. (in press). Mining frequent itemsets using the N-list and subsume concepts. *International Journal of Machine Learning and Cybernetics*. http://dx.doi.org/10.1007/s13042-014-0252-2.

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

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

Vo, B., Le, T., Hong, T. P., & Le, B. (2014). An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining. *Applied Intelligence, 41*(3), 759–775.

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

Zaki, M., & Gouda, K. (2003). Fast vertical mining using diffsets. In *SIGKDD'03* (pp. 326–335).

Zaki, M. J., & Hsiao, C. J. (2002). CHARM: An efficient algorithm for closed itemset mining. In *SDM'02* (pp. 457–473).

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