Accepted Manuscript

An N-list-based Algorithm for Mining Frequent Closed Patterns

Tuong Le, Bay Vo

PII: S0957-4174(15)00290-0
DOI: http://dx.doi.org/10.1016/j.eswa.2015.04.048
Reference: ESWA 9995

To appear in: Expert Systems with Applications

Please cite this article as: Le, T., Vo, B., An N-list-based Algorithm for Mining Frequent Closed Patterns, Expert Systems with Applications (2015), doi: http://dx.doi.org/10.1016/j.eswa.2015.04.048

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
An N-list-based Algorithm for Mining Frequent Closed Patterns

Tuong Le\textsuperscript{1,2}, Bay Vo\textsuperscript{1,2,*}

\textsuperscript{1}Division of Data Science and \textsuperscript{2}Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam

\{lecungtuong,vodinhbay\}@tdt.edu.vn, \{tuonglecung, bayvodinh\}@gmail.com

\textbf{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.

\textbf{Keywords:} data mining, frequent closed pattern, N-list structure

1 Introduction

Frequent pattern mining (Deng et al., 2012; Dong & Han, 2007; Guil & Marín, 2013; Han et al. 2000; Song et al., 2008) is important in association rule mining (Agrawal & Srikant, 1994; Vo et al., 2013), sequential mining (Agrawal & Srikant, 1995; Mabroukeh & Ezeife, 2010), and classification (Liu et al., 1998). Many algorithms have been proposed for mining frequent

* Corresponding author.
patterns, such as Apriori (Agrawal & Srikant, 1994), FP-growth (Han, Pei, & Yin, 2000), methods based on IT-tree (Zaki & Gouda, 2003; Vo et al., 2012), and methods for mining frequent patterns in incremental databases (Hong et al., 2009; Vo et al., 2014b). 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 et al., 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 et al., 2008), frequent weighted itemset mining (Yun et al., 2012; Vo et al., 2013), frequent pattern mining from uncertain data (Aggarwal et al., 2009; Bernecker et al., 2009), and erasable itemset mining (Deng & Xu, 2012; Le, Vo, & Coenen, 2013; Le & Vo, 2014; 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 et al. 1999; Pei et al. 2000; Zaki & Hsiao, 2002; Zaki & Hsiao, 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 et al., 2005), CHARM (Zaki & Hsiao, 2002), dCHARM (Zaki & Hsiao, 2005), DBV-Miner (Vo et al., 2012), and DCI_PLUS (Sahoo et al., 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 et al. (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 et al., 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 et al., 2014a). 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.

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 (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 et al., 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, Vo, & Coenen, 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 (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 DCL_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 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 $ACW$.

**Table 1.** Example transaction dataset

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$A, C, T, W$</td>
</tr>
<tr>
<td>2</td>
<td>$C, D, W$</td>
</tr>
<tr>
<td>3</td>
<td>$A, C, T, W$</td>
</tr>
<tr>
<td>4</td>
<td>$A, C, D, W$</td>
</tr>
<tr>
<td>5</td>
<td>$A, C, D, T, W, E$</td>
</tr>
<tr>
<td>6</td>
<td>$C, D, T, E$</td>
</tr>
</tbody>
</table>
Let $\text{minSup}$ (minimum support threshold) be a given threshold. A pattern $X$ is called a frequent pattern if $\sigma(X) \geq \text{minSup} \times n$. 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 $\text{minSup} = 50\%$. $AW$ 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 $n(N_i)$, $f(N_i)$, $\text{childs}(N_i)$, $\text{pre}(N_i)$, and $\text{post}(N_i)$, 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.

<table>
<thead>
<tr>
<th>Algorithm 1. PPC-tree creation algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $DB$, $\text{minSup}$</td>
</tr>
<tr>
<td><strong>Output:</strong> $\mathcal{R}$, $I_1$, $H_1$, and $\text{threshold}$</td>
</tr>
<tr>
<td><strong>Method name:</strong> Construct_PPC_tree</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
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 \quad t \leftarrow \text{the first item of } T \text{ and } T \leftarrow T \setminus t
5 \quad \text{if } \mathcal{R} \text{ has a child } N \text{ such that } n(N) = t \text{ then}
6 \quad \quad f(N) = f(N) + 1
7 \quad \text{else}
8 \quad \quad \text{create a new node } N \text{ with } n(N) = t, f(N) = 1 \text{ and } \text{childs}(\mathcal{R}) = N
9 \quad \quad Insert_{-}Tree(T, N)
10 end while
11 end

---

**Fig. 1.** PPC-tree creation algorithm.

### 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 \( C_i = (pre(N_i), pre(N_i), f(N_i)) \).

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:

\[
NL(A) = \bigcup_{\{N_i \in \mathcal{R} \mid n(N_i) = A\}} C_i
\]  

(1)

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

\[
\sigma(A) = \sum_{C_i \in NL(A)} f(C_i)
\]  

(2)
**N-list associated with k-patterns:** Let $X_A$ and $X_B$ 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. $NL(X_A)$ and $NL(X_B)$ are two N-lists associated with $X_A$ and $X_B$, respectively. Vo et al. (2014) presented an effective method for determining the N-list associated with $X_{AB}$ as follows: for each PP-code $C_i \in NL(X_A)$ and $C_j \in NL(X_B)$, if $C_i$ is an ancestor of $C_j$ and:

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

The support of $X_{AB}$, denoted by $\sigma(X_{AB})$, is calculated as follows:

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

Vo et al. (2014) also presented an algorithm for determining the intersection of two N-lists, shown in Fig. 4.

---

**Algorithm 2. Improved N-list intersection function**

**input:** $PS_1$, $PS_2$

**output:** $PS_3$ and frequency

**Method name:** N-list-Intersection

1. $PS_3 \leftarrow \emptyset$
2. let $sF$ be the sum of frequencies of $PS_1$ and $PS_2$
3. let $i = 0$, $j = 0$, and $\text{frequency} = 0$
4. while $i < |PS_1|$ and $j < |PS_2|$ do
5.   begin while
6.   if $\text{pre}(PS_1[i]) < \text{pre}(PS_2[j])$ then
7.     if $\text{post}(PS_1[i]) > \text{post}(PS_2[j])$ then
8.       if $|PS_3| > 0$ and $\text{pre}(PS_3[|PS_3| - 1]) = \text{pre}(PS_1[i])$ then
9.         $f(PS_3[|PS_3| - 1]) += f(PS_2[j])$
10.     else
11.       add the tuple $\langle \text{pre}(PS_1[i]), \text{post}(PS_1[i]), f(PS_2[j]) \rangle$ to $PS_3$
12.   end if
13. end if
14. end if
15. end while
16. end while
17. return $PS_3$ and $\text{frequency}$
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 2.3 and $NL(XB) \subseteq NL(XA)$, we have:

$$NL(XAB) = \bigcup_{C_i} \langle \text{pre}(C_i), \text{post}(C_i), f(C_j) \rangle$$  \hfill (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.
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.

**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 \( R, I_1, H_1, \) and threshold
2. Generate-\text{N}\_list\((R)\)
3. Find-\text{FCIs}(I_1)
4. return \( FCIs \)

1. Procedure Generate-\text{N}\_list\((R)\)
2. Let \( C \leftarrow \langle pre(R), post(R), f(R) \rangle \)
3. Let \( H = H_1[n(R)] \) is the node of \( n(R) \) in \( H_1 \)
4. Add \( C \) to \( NL(H) \)
5. \( f(H) += f(R) \)
6. for each \( child \) in \text{childs}(R) do
7. Generate-\text{N}\_list\((child)\)

1. Procedure Find-\text{FCIs}(Is)
2. for \( i \leftarrow |Is| - 1 \) to 0 do
\begin{verbatim}
FCIs_{next} \leftarrow \emptyset

for j \leftarrow i-1 to 0 do
    begin for
        if N-list-check(NL(Is[i]), NL(Is[j])) = true then
            if f(Is[i]) = f(Is[j]) then
                Is[i] = Is[i] \cup Is[j]
                remove Is[j]
                i--
            else
                Is[i] = Is[i] \cup Is[j]
                for each f in FCIs_{next}
                    update f = f \cup Is[j]
                    continue
            end for
    FCIs \leftarrow Is[i] \cup Is[j]
    NL(FCI) and f(FCI) \leftarrow N-list-intersection(NL(Is[i]), NL(Is[j]))
    if f(FCI) \geq \text{threshold} then
        add FCI to FCIs_{next}
    end for

if Subsumption-check(Is[i]) = false then
    add Is[i] to FCIs
    add index of FCI in FCIs to Hash[f(FCI)]

Find-FCIs(FCIs_{next})

Function N-list-check(N_1, N_2)
    let i = 0 and j = 0
    while j < |N_1| and i < |N_2| do
        if pre(N_2[i]) < pre(N_1[j]) and post(N_2[i]) > post(N_1[j]) then
            j++
        else
            i++
    if j = |N_1| then
        return true
    return false
\end{verbatim}
Function Subsumption-check($FCI$)

1. let $Arr \leftarrow \text{Hash}[f(FCI)]$
2. for $i \leftarrow 0$ to $|Arr| - 1$
3. if $FCI \subseteq FCIs[Arr[i]]$ then
4. return true
5. return false

Fig. 3. NAFCP algorithm.

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 \textbf{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 $\text{minSup} = 50\%$ to illustrate the operation of this algorithm. First, NAFCP removes all items whose frequency does not satisfy the $\text{minSup}$ threshold and sorts the remaining items in descending order of frequency (Table 2).

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Ordered frequent items</th>
</tr>
</thead>
</table>

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

**Fig. 4.** Illustration of creation of PPC-tree using example DB with \( \text{minSup} = 50\% \).
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. Figure 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. Figure 7 shows the list of FCPs obtained by NAFCP using the example DB with $minSup = 50\%$.
Fig. 7. FCPs identified using NAFCP for example DB with minSup = 50%.

6 Performance studies

All experiments presented in this section were performed on a laptop with an Intel Core i3-3110M 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†. 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 3,196 transactions and 76 items. The Mushrooms dataset contains 8,124 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.

Table 3. Statistical summary of experimental datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th># of trans</th>
<th># of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accidents</td>
<td>Sparse</td>
<td>340,183</td>
<td>468</td>
</tr>
<tr>
<td>Chess</td>
<td>Dense</td>
<td>3,196</td>
<td>76</td>
</tr>
<tr>
<td>Connect</td>
<td>Dense</td>
<td>67,557</td>
<td>130</td>
</tr>
</tbody>
</table>

† Downloaded from http://fimi.cs.helsinki.fi/data/.
## Dataset Information

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th># of trans</th>
<th># of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mushroom</td>
<td>Dense</td>
<td>8,124</td>
<td>120</td>
</tr>
</tbody>
</table>

### 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.

![Runtime Graph](image)

**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.
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 \( \text{minSup} = 60\% , 50\% , \text{and} 40\% \); Fig. 13 with \( \text{minSup} = 75\% \) and 70\%; Fig. 14 with \( \text{minSup} = 80\% \) and 70\%; and Fig. 15 with \( \text{minSup} = 10\% \) and 8\%). However, for small thresholds, NAFCP clearly outperforms dCHARM and DCI_PLUS in terms of memory usage (Figs. 12-15).

![Fig. 12. Memory usage of NAFCP, dCHARM, and DCI_PLUS for Accidents dataset.](image)

![Fig. 13. Memory usage of NAFCP, dCHARM, and DCI_PLUS for Chess dataset.](image)
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.

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 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


Highlights

- Two theorems for fast determining closed patterns based on N-list structure are presented.
- An N-list-based algorithm for mining closed patterns is then proposed.
- The proposed algorithm outperforms a number of classical algorithms in terms of runtime and memory usage in most cases.