A GRAPH-BASED ALGORITHM FOR FREQUENT CLOSED ITEMSETS MINING

Li Li
Donghai Zhai
Fan Jin

School of Computer and Communication Engineering,
Southwest Jiaotong University
Chengdu, 610031, P.R.CHINA

ABSTRACT

Frequent itemsets mining plays an essential role in data mining, but it often generates a large number of redundant itemsets that reduce the efficiency of the mining task. Frequent closed itemsets are subset of frequent itemsets, but they contain all information of frequent itemsets. The most existing methods of frequent closed itemset mining are apriori-based. The efficiency of those methods is limited to the repeated database scan and the candidate set generation. This paper proposes a graph-based algorithm for mining frequent closed itemsets called GFCG (Graph-based Frequent Closed itemset Generation). The new algorithm constructs an association graph to represent the frequent relationship between items, and recursively generates frequent closed itemsets based on that graph. It scans the database for only two times, and avoids candidate set generation. GFCG outperforms apriori-based algorithm in experiment study and shows good performance both in speed and scale up properties.

1 INTRODUCTION

Frequent itemsets mining plays an essential role in mining associations rules (R.Agrawl 1994; Brin 1997; H. Mannila 1997), partial periodicity (J.Han 1999), episodes (H.Mannila 1997) and many other important data mining areas.

Let \( I = \{i_1, i_2, \ldots, i_m\} \) be the set of items, \( D_{\text{num}} = \{T_1, T_2, \ldots, T_n\} \) be the transaction database, \( T_i \ (i \in [1, n]) \) be a record of transaction, it consisted of items from \( I \). Let \( S \) be an itemset, \( S = \{i \mid i \in I\} \). The support of \( S \), recorded as \( \text{supp} \ (S) \), is the rate of the transactions that contain itemset \( S \). If the support of \( S \) is not less than the minimum support threshold, \( S \) is called frequent itemset.

However, it is well known that mining task often generates large number of frequent itemsets, which reduces the efficiency of association rules mining or other tasks.

Instead of mining the complete set of frequent itemsets, we can mine the frequent closed itemsets. The frequent closed itemsets contain all information of frequent itemset and have the same power as frequent itemsets. It will largely reduce the number of redundant itemset and increase the efficiency of mining task.

For example, suppose a database contains only the same two transaction, \( T_1=\{i_1, i_2, \ldots, i_{100}\} \) and \( T_2=\{i_1, i_2, \ldots, i_{100}\} \). The minimum support threshold is 2, \( 2^{100-1} \) frequent itemsets will be generated. They are \( \{i_1\}, \{i_2\}, \ldots, \{i_{100}\}, \{i_1 i_2\}, \{i_1 i_3\}, \{i_{100} i_{100}\}, \ldots, \{i_1 i_2 \ldots i_{100}\} \). But the only one frequent closed itemset will be generated. It is \( \{i_1 i_2 \ldots i_{100}\} \).

Definition 1: \( S_1 \) and \( S_2 \) are two itemsets, \( S_1 \subseteq S_2 \), \( S_1 \) is covered by \( S_2 \) if and only if any transaction that contains \( S_1 \) also contains \( S_2 \), recorded as \( S_1 \supseteq S_2 \).

Lemma 1: let \( S_1, S_2, S_3 \) be itemsets, if \( S_1 \supseteq S_2 \) and \( S_2 \supseteq S_3 \), then \( S_1 \supseteq S_3 \).

Rational: Let transaction \( t \) contains \( S_1 \). For \( S_1 \supseteq S_2 \), \( t \) also contains \( S_2 \), for \( S_2 \supseteq S_3 \), \( t \) also contains \( S_3 \). From definition 1, \( S_1 \supseteq S_3 \).

Definition 2: \( S \) is an itemset, if there does not exist itemset \( S', S \supseteq S', \) i.e., \( S \) not covered by any itemset, then \( S \) is a closed itemset, if the support of \( S \) is not less than the minimum support threshold, then \( S \) is a frequent closed itemset.

Lemma 2: \( S \) is a frequent itemset, but not closed. Then there exists a frequent closed itemset \( S', S \supseteq S' \).

Rational: \( S \) is not closed, from definition 2, there exists an itemset \( S_1, S \supseteq S_1 \). If \( S_1 \) is a closed itemset, then from \( \text{supp} \ (S_1) = \text{supp} \ (S) \), \( S_1 \) is a frequent closed itemset. If \( S_1 \) is not closed, for the same reason, there exists an itemset \( S_2, S \supseteq S_2 \).
Since the length of itemsets is limited, we can find \( S_k, S_{k-1}, S_k \) from lemma 1, \( S_k \).

**Lemma 3:** \( S \) is a frequent itemset, and then \( S \) is a frequent closed itemset if and only if \( S \) is not covered by any other frequent closed itemset.

**Rational:**
(1) \( S \) is a frequent closed itemset, and then \( S \) is not covered by any itemsets. So it is not covered by any frequent closed itemsets.
(2) \( S \) is a frequent itemset. If \( S \) is not closed, from lemma 2, there must exists a frequent closed itemset that cover it. That conflict to the condition.

When the concept of frequent closed itemset is introduced, the mining task is substantially reduced. The number of the frequent itemsets generated in that example is \( 2^{100} - 1 \), but the number of frequent closed itemset is 1. That frequent closed itemset is \{i_1, i_2, \ldots, i_{100}\} and it contains all information of frequent itemset. So the frequent closed itemset mining is very important to association rules mining or some other tasks.

The most method for the frequent closed itemsets mining are apriori-based, such as A-close (Nicolas Pasquier 1999). The efficiencies of those methods are limited to the bottleneck of repeated database scan and the candidate set generation.

This paper studies the efficient mining of frequent closed itemsets in large databases, and proposes a new algorithm, called GFCG. Nicolas Pasquier propose an Apriori-based mining algorithm, called A-close (Nicolas Pasquier 1999). Algorithm A-close is taken as comparison in this paper. From the performance study, the new algorithm shows good performance both in speed and scale up property.

## 2 THE FREQUENT CLOSED ITEMSET MINING ALGORITHM GFCG.

### 2.1 The construction of graph.

The algorithm GFCG adopts the structure of bit-vector. Every frequent item has a corresponding bit-vector. The number of bits in each bit-vector equals to the number of transactions in the database. The GFCG scans the database for the first time to find the frequent items, and initiates a bit-vector for every frequent item. All bits in bit-vector are set to 0. Then the algorithm scans the database for the second time. At this time the GFCG algorithm sets every bit in bit-vector. If one frequent item appears in \( k \)th transaction, then the \( k \)th bit of its bit-vector was set to 1. Then the GFCG algorithm does not scan the database anymore.

Let the minimum support threshold be 2. The frequent items in the database are items \( 1, 2, 3, 4, 5 \). The corresponding bit-vectors are \( BV_1=(01011) \), \( BV_2=(10100) \), \( BV_3=(01010) \), \( BV_4=(10101) \), \( BV_5=(10111) \).

**Property 1:** The support of the itemset \( i_1, i_2, \ldots, i_k \) is the number of 1s in \( BV_{i_1} \land BV_{i_2} \land \ldots \land BV_{i_k} \), where the notation “\( \land \)” is the logical AND operation.

When the bit-vectors have been initiated, it is not necessary to scan the database anymore. In the graph construction phase, GFCG algorithm constructs an association graph to indicate the relationships between frequent items. For the association graph, if the number of 1s in \( BV_{i_1} \land BV_j \) (\( i > j \) according to an order \( L \) of frequent items) is not less than the minimum support threshold, a directed edge from item \( i \) to item \( j \) is constructed, which is recorded as \( i \rightarrow j \). The edge between \( i, j \) means the itemset \( \{i, j\} \) is a frequent 2-itemset. Let the order be \( L = \{1,2,3,4,5\} \), the association graph for the above example is shown in figure 1, and the frequent 2-items are \( \{1,3\}, \{1,5\}, \{2,4\}, \{2,5\}, \{4,5\} \).

### Table 1: A database of Transaction

<table>
<thead>
<tr>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2 4 5</td>
</tr>
<tr>
<td>200</td>
<td>1 3</td>
</tr>
<tr>
<td>300</td>
<td>2 4 5</td>
</tr>
<tr>
<td>400</td>
<td>1 3 5</td>
</tr>
<tr>
<td>500</td>
<td>1 4 5</td>
</tr>
</tbody>
</table>
2.2 The generation of frequent closed itemsets.

When the graph has been constructed, the frequent closed itemsets can be mined recursively based on the bit-vectors and the association graph. The following lemmas are proposed.

**Lemma 4:** \{i_1 i_2 \ldots i_k\} is a frequent itemset. If there is no association edge starting from item \(i_k\) to item \(u\), then \{\(i_1 i_2 \ldots i_{k-1} u\)\} can't be frequent itemset.

**Proof:** Because there is no association edge starting from item \(i_k\) to item \(u\), \{\(i_k u\)\} is not frequent itemset. From the property of apriori, any itemset that contains infrequent itemset can’t be frequent itemset. So \{\(i_1 i_2 \ldots i_k u\)\} can’t be frequent itemset.

According to lemma 4, if we want to extend the frequent \(k\)-itemset \{\(i_1, i_2 \ldots i_k\)\} to frequent \((k+1)\)-itemset, we only need to check those items that at the end of edge starting from item \(i_k\). Let \(u\) be one of such item and the number of 1s in \(BV_{i1} \vee BV_{i2} \vee \ldots \vee BV_{ik} \vee BV_{u}\) is not less than minimum support threshold, then \{\(i_1 i_2 \ldots i_k u\)\} is also a frequent itemset. Take the database in Table 1 as example. \{\(2 \ 4\)\} is a frequent itemset, and there is directed edge starting from item 4 to 5. We check the number of 1s in \(BV_{2} \vee BV_{4} \vee BV_{5}\). The number of 1s in \(BV_{2} \vee BV_{4} \vee BV_{5}\) is 2, not less than the minimum support threshold, so \{\(2 \ 4\)\} is a frequent itemset.

**Definition 3:** \(L\) is an order of frequent items. All the items in frequent itemsets are sorted according to the order \(L\). \(f\) \(=\{i_1 i_2 \ldots i_k\}\) is a frequent itemset. Then the set of frequent itemset

\[
S_f = \{s \mid s = \{i_1, i_2, \ldots, i_k\} \cup \{j_1, j_2, \ldots, j_m\}, \text{for all } i_m > j_n, 1 \leq m \leq k, 1 \leq n \leq l\}
\]

(1)
is called the cluster of itemsets that have the same prefix \(f\).

\[
S_f' = \{s \mid s = \{i_1, i_2, \ldots, i_k\} \cup \{j\}, \text{for all } i_m > j, 1 \leq m \leq k\}
\]

(2)
is called the extended itemset cluster of \(f\). Obviously, \(S_f' \subseteq S_f\) hold.

**Lemma 5:** \(I = \{i_1, i_2, \ldots, i_k\}\) is a set of frequent items. All the frequent itemsets, \(S\), can be represented as

\[
S = S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}
\]

where \(S_{i_k}\) is the cluster of itemsets that have the same prefix \(i_k\).

**Proof:** Since the items in each frequent itemset are sorted according to order \(L\), then all frequent itemsets can be classified according to the first item. Each class can be represented as \(S_{i_k} i \in I\).

**Lemma 6:** \(f = \{i_1 i_2 \ldots i_k\}\) is a frequent itemset. \(S_f\) can be represented as

\[
S_f = \cup S_{i_k} \cup u, u < i_m, 1 \leq n \leq k
\]

(4)

**Proof:** Same as the proof of lemma 5, \(S_f\) can be classified according to the item \(u\) that follows the \(f\) immediately, \(u < i_m, 1 \leq n \leq k\).

According to lemma 5 and lemma 6, we can find out all of the frequent itemsets recursively.

**Lemma 7:** \(S_i = \{i_1, i_2 \ldots i_k\}\) is a frequent itemset. Support of \(S_i\) is \(\delta_1\). \(S_{i_2} = \{i_1, i_2 \ldots i_{k-1} u\}\) is a frequent itemset. Support of \(S_i\) is \(\delta_2\). If \(\delta_2 = \delta_1\), then \(S_i = \{i_1, i_2 \ldots i_k\}\) can’t be frequent closed itemset.

**Proof:** Let \(P = \{p_1, p_2, \ldots, p_\delta\}\) be the set of transactions that contains \(S_i\). \(Q = \{q_1, q_2, \ldots, q_\delta\}\) be the set of transactions that contains \(S_2\). Since the transactions that contain \(S_2\) must contain \(S_i\), \(Q \subseteq P\), \(\delta_2 \leq \delta_1\) hold. For \(\delta_2 = \delta_1\), we have \(P = Q\). That is to say any transaction that contains \(S_i\) must contains \(S_2\). From the definition of frequent closed itemset, \(\{i_1, i_2 \ldots i_k\}\) can’t be frequent closed itemset.

Based on the above discussions, the GFCG algorithm is described in the following programs.

**Algorithm GFCG**

Input database \(D\), support threshold \(\text{minsup}\)

output All of the frequent closed itemset \(C\)

GFCG(\(D\), \text{minsup}, \(C\))

\(
\{C = \phi; F = \phi; \\
F = \text{CreateFrequentItems}(D, F, \text{minsup}); \\
\text{CreateBitVector}(D, F); \\
\text{CreateGraph}(F); \\
\text{for all item } i \in F \\
\text{MineSamePrefixFreq}(\{i\}, BV_i, i.\text{count}, C); \\
\}
\)

CreateFrequentItems(D, F, \text{minsup})

\(
\{ \text{\#N is number of transactions} \\
F = \phi; \\
\text{for} \{ j=1; j \leq N; j++ \} \{ \\
\text{for all items } i \text{ in } j\text{th transaction} \\
\{i.\text{count}++; \}
F=\{i \mid i \text{ is an item and } i.\text{count} \geq \text{minsup}; \}
\}
\)

CreateBitVector(D, F)

\(
\{ \text{for all items } i \text{ in } F \\
\text{allocate BV, and set all bit in } BV_i \text{ to } 0; \\
\text{for} \{ j=1; j \leq N; j++ \} \{ \\
\text{for all items } i \text{ in } j\text{th transaction} \\
\{ \text{set the jth bit of } BV_i \text{ to } 1 \} \}
\}
\)

CreateGraph(F)

\(
\{ \text{let } L \text{ be an order of the items in } F; \\
\text{for all frequent items } i \in F \\
\text{for all frequent items } j \in F \ i > j \\
\text{if (number of 1 in } BV_i \text{ equals } \text{minsup then} \\
i.\text{link.add}(j); //create edge } i-j \\
\}
\)

//BV_i is the bit vector of \(i\)

MineSamePrefixFreq(I, BV_i, \text{supp}, C)
\[
\text{let } i \text{ be the last item in itemset } I; \\
\text{for all } j, j \neq i \text{ link } \{ \\
I \leftarrow \{ j \}; \ BV_j \leftarrow BV_j \times BV_i; \\
\text{let } n_{\text{newapp}} \text{ be the number of Is in } BV_j; \\
\text{if } (n_{\text{newapp}} \geq \text{minsup}) \{ \\
\text{if } (n_{\text{newapp}} = \text{minsup}) \{ \\
\text{covered } \leftarrow \text{TRUE } ; \\
\text{MineSamePrefixFreq}(I, BV_j, n_{\text{newapp}}, C); \\
\} \} \\
\text{if } (\text{covered } = \text{FALSE } ) \{ \\
\text{if } (I \text{ is not covered by } i, I \in C) \\
C \leftarrow C \cup I; \\
\}
\]

The purpose of using the parameter BV\_i and \text{n}_{\text{newapp}} in function \text{MineSamePrefixFreq} is to acquire the bit-vector and the support of itemset \text{I}'s extension conveniently. Using the two parameters, we can avoid generating bit-vector and support of new itemset from the beginning.

From lemma 5 and lemma 6, we know that GFCG algorithm can finds out all frequent closed itemsets through the method of MineSamePrefixFreq. Let the order of the frequent closed itemsets be found out be \( L, I = \{ i_1, i_2, \ldots, i_k \} \) is the frequent itemset current find. \( S_{\text{before}} \) is the set of itemset before \( I \) in \( L \), \( S_{\text{after}} \) is the set of itemsets after \( I \) in \( L \).

**Lemma 8:** Let \( S' \) be the extended itemset cluster of \( I \), then there does not exist frequent itemset \( P \) that covers \( I, P \in S' \), if and only if, there does not exist \( P' \) that covers \( I, P' \in S_{\text{after}} \).

**Proof:**

(1) From the way that GFCG finds frequent itemsets, we have the relationship \( S' \subseteq S_{\text{after}} \).

(2) If there exists a frequent closed itemset \( P' \in S_{\text{after}}, P' \) covers \( I \), then \( P' \in S_i \), \( S_i \) is the cluster of itemsets that have the same prefix \( I \). Let \( P'' = \{ i_1, i_2, \ldots, i_k \} \cup \{ j_1, j_2, \ldots, j_l \} \), then \( P'' \) covers \( I \) too, \( P' \in S' \). That conflict to the fact that there does not exist frequent itemset \( P' \) that covers \( I, P' \in S' \).

**Lemma 9:** Algorithm GFCG can finds out all frequent closed itemsets.

**Proof** From the lemma 3, itemset \( I \) is a frequent closed itemset if and only if \( I \) not covered by any frequent closed itemset. That means \( I \) is neither covered by any frequent closed itemset in \( S_{\text{before}} \), nor covered by any frequent closed itemset in \( S_{\text{after}} \). From lemma 8, if \( I \) is not covered by any itemset of extended itemset cluster of \( I \), we can say \( I \) is not covered by any frequent closed itemset in \( S_{\text{after}} \). In function MineSamePrefixFreq’s 3th step, we set the variable “covered” to FALSE. In 10th step, if \( I \) is covered by any itemset of the extended itemset cluster of \( I \), then set the variable “covered” to TRUE. \( I \) can’t be frequent closed itemset. \( S_{\text{before}} \) equals to the current set of frequent closed itemsets \( C \) that have been found out. If we can confirm that \( I \) not covered by any itemset of \( C \), \( I \) is frequent closed itemset. From the discussion above, GFCG can finds out all the frequent closed itemset.

**3. EXPERIMENT EVALUATION AND PERFORMANCE STUDY**

In this section, we present a performance comparison between the GFCG algorithm and the Apriori-based algorithm A-close. All of the experiments are performed on 300MHz PC machine with 128 megabytes main memory, running on Microsoft Windows Me. All the programs are written in Microsoft/ Visual C++6.0. The data set used for performance study include one real data set and two synthetic data sets. The real data set is the Traditional Chinese Medicine (TCM) database, which has 1378 prescription records, each prescription has several drugs. The synthetic data sets are generated using the procedure described in (R Agrawal 1994). The parameters of the synthetic data sets are described in table 1. We first test the speed property of the two algorithms on the real and synthetic data set by changing the minimum support threshold. The results of the experiments are shown in figure 2, 3 and 4. Then we test the scale up property of the two algorithms by fixing the minimum support threshold to specified value, 1%, and changing the size of the data set. The results of the experiments are shown in figure 5.

From the above mentioned performance study, we can see that GFCG algorithm has good performance both in speed and scale-up property.

The good performance of GFCG comes from the following reasons. Firstly, GFCG adopts the technique of bit-vector, which substantially compresses the information of the transaction database. Secondly, the construction of association graph makes the algorithm avoid candidate set generation. The generation of new frequent closed itemset can be guided by the association graph efficiently. Furthermore, by the construction of graph, the times of database scan are substantially reduced. Only twice database scans are needed. Finally, the concept of extended itemset cluster also reduces the search space when judge weather one frequent itemset is closed.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average length</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Total item</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Number of transaction</td>
<td>5K</td>
<td>10K</td>
</tr>
</tbody>
</table>

**4. CONCLUSION**

For the task of mining frequent closed itemset, this paper proposes a graph-based algorithm named GFCG. The new algorithm adopts the technique of bit-vector, and constructs an association graph to represent the frequent relationships...
between frequent items. The GFCG generates frequent closed itemset based on that association graph. GFCG also introduces the concept of extended itemset cluster, which largely reduces the search space. The experiment evaluation and performance study on real data set and synthetic data set show that the new algorithm outperforms apriori-based algorithm and has good performance both in speed and scale up property.

REFERENCES


AUTHOR BIOGRAPHIES

LI LI is graduated student in Institute of Neural Network and Information Technology of Southwest Jiaotong University, P.R.CHINA. His research focuses on Rough set theory, combinatorial optimization and data mining. He can be contacted by e-mail at <italylili@163.com>

DONG-HAI ZHAI is graduated student in Institute of Neural Network and Information Technology of Southwest Jiaotong University, P.R.CHINA. His research focuses on fuzzy inference, neural network, combinatorial optimiza-
tion. He can be contacted by e-mail at <zhaidh@yeah.net>, <zhaidonghai@sina.com.cn>

**FAN JIN** is the director of Institute of Neural Network and Information Technology, Southwest Jiaotong University, P.R.CHINA. His research focuses on combinatorial coding, combinatorial optimization, neural network, computational intelligence. He can be contacted by e-mail at <fan_jin@sc.cninfo.net.cn>