Improved Apriori Algorithm for Mining Association Rules
Yu-gang DAI, Xiang ZHANG, Tao XU, Lin YE and Ya-jing MA
Northwest Minzu University, Key Laboratory of Chinese Language and Information Technology of
Ministry of Education, Lanzhou city, Gansu province, China
*Corresponding author

Keywords: Apriori algorithm, Association rules, Data mining, Frequent itemsets.

Abstract. Apriori algorithm as a classic algorithm in data mining, it has a good performance in a
small number of transactions in the database which has been widely used by people, but the
algorithm has two inherent flaws, affect the efficiency of Apriori algorithm mining information in
large database. Aiming at the Bottleneck Problem Restricting the Efficiency of Apriori Algorithm,
in this paper, two inherent flaws of Apriori algorithm are improved, in order to improve Apriori
algorithm in large database mining efficiency. The algorithm reduces the number of connections
and the number of database scan to shorten the database scan time. Experimental results show, the
optimized Apriori algorithm has some improvements in operation efficiency.

Introduction
With the continuous development of information technology, a lot of data and information
emerged from various industries. People need to dig out useful information from these vast
amounts of information, so the birth of data mining. Data mining is the process of discovering
interesting knowledge from large amounts of data stored in a database, data warehouse, or other
repository[1]. Association rules is an important area of data mining, the main research affairs
database in the causal relationship between the items. Apriori algorithm is one of the most
influential data mining algorithms for mining frequent itemsets of Boolean association rules. The
basic idea is to repeatedly scan the database, arbitrary subsets based on a frequent set are frequent
sets, a candidate set of length k + 1 may be iteratively generated from frequent sets of length k,
scan the database to verify if it is a frequent episode. However, when there are more transactions in
the database and the set of items is larger, the computation of the scan is large and takes a lot of
time.

Problem Description of Association Rules
The association rules were proposed by R. Agrawal et al. In 1993, is an important research topic in
data mining [2]. The mining of association rules is to find the association rules with the minimum
support and the minimum confidence given by the user in the transaction database D, can be
broken down into two sub-problems: One is to find all the items in the transaction database D
greater than or equal to the user-specified minimum support, that is frequent itemsets; The second
is to use frequent itemsets to generate the required association rules. There are many algorithms
that produce frequent itemsets, when these algorithms generate frequent itemsets, it is very
time-consuming to scan the database to generate candidate itemsets. Therefore, finding an efficient
production algorithm for frequent itemsets is the crux of the problem.

Let I = \{i_1, i_2, ..., i_m\} be a collection of all data items, Let D = \{T_1, T_2, ..., T_n\} be the set of all
transactions, a transaction database, each transaction T is a subset of the data items, that T \subseteq I. Each
transaction can be identified by a unique identifier TID. Let X be a collection of data items called
itemsets, an item set containing k data items is called a k-item set. The transaction T contains X if
and only if X \subseteq T. The support for data item set X in transaction database D is: support
(X)=P(X)=|TX|/|D|, Among them, TX={T \in D|X \subseteq T}. Under the user-defined support threshold
min_ support, if support(X)\geq min_ support, then X is a frequent itemset; otherwise X is a
non-frequent itemset. However, for ease of description below, the support for data itemset X is expressed as the number of transactions in D that contain X. An association rule is an implication of the form “X $\Rightarrow$ Y”, among them, X $\Rightarrow$ I, Y $\Rightarrow$ I and X$\cap$Y=$\emptyset$. If D contains the percentage of transactions XU Y is s, then s is the support of association rule X $\Rightarrow$ Y, which is the probability P(XU Y). If the transaction containing X in D contains the percentage of Y at the same time as c, then c is the trustworthiness of the association rule X $\Rightarrow$ Y, which is the conditional probability P(Y|X), which is: (1) support(X $\Rightarrow$ Y)= P(XU Y)= support(XU Y), (2) confidence(X $\Rightarrow$ Y)= P(Y|X)= support(XU Y)/support(X).

The problem with mining association rules is to generate all the satisfactions: support(X $\Rightarrow$ Y)≥ min_ support and confidence(X $\Rightarrow$ Y)≥ min_ confidence, Min_ support and min_ confidence are respectively the minimum support threshold and minimum confidence threshold given by the user. Association rules that satisfy both of these conditions are called strong association rules.

Mining association rules mainly contains two main steps:
(1) Generate all frequent itemsets from transaction database D.
(2) Generate strong association rules according to the frequent itemsets obtained.

In fact, the first sub-problem in the entire execution of mining association rules is the core. When all the frequent itemsets are found, the corresponding association rules will be easy to generate. The Apriori algorithm mainly deals with the first subproblem.

Apriori Algorithm

Apriori algorithm is a primitive algorithm proposed by R. Agrawal and R. Sakant in 1994 for mining frequent itemsets of Boolean association rules, Apriori uses an iterative method called layer-by-layer search, and k-sets are used to explore (k + 1) sets. Apriori algorithm is the first step of mining association rules, that is, discovering all the frequent itemsets from the transaction database, apriori algorithm uses the iterative method of layer-by-layer search to generate frequent itemsets. First scan the transaction database D, the cumulative count of each item, 1-frequent itemsets L_1 in the database D are found according to the minimum support set by the user, and then the 2-item candidate item set C_2 is generated through the connection operation of the 1-frequent itemsets L_1, and the transaction database is scanned again from item 2-frequent itemsets L_2 are found in the candidate item set C_2, then the 2-item frequent itemsets L_2 are connected to generate a 3-item candidate item set C_3, and the transaction database is scanned again to find 3 of the 3-item candidate sets - Item frequent itemsets L_3 and so on, until no more frequent k-item frequent itemsets or candidate itemsets of the pattern are empty, the Apriori algorithm ends.

Apriori algorithm description:
Enter: Transaction database D, minimum support threshold min_ support.
Output: Frequent itemset L in D.
Algorithm Description:
L_1=find_frequent_1_itemsets(D);
for(k=2;L_{k-1}≠Φ; k++){  
C_k=appriori_gen(L_{k-1});  // Produces candidate k-items
for each transaction t∈D{  // Scan D, count
  C=subset(C_k,t);    //A subset of C_k candidates is obtained from transaction t
  for each candidates c∈C_t
    c.count++;
  }  
L_k={c∈C_k|c.count≥min_ support;}
}
return L=U_kL_k

Apriori algorithm connection and pruning process:
for each Item set l_1∈L_{k-1}
for each Item set l_2∈L_{k-1}
if (l_1[1]=l_2[1]) ∧ ... ∧ (l_1[k-2]=l_2[k-2]) ∧ (l_1[k-1]<l_2[k-2])then{ c=l_1>l_2;
if has_infrequent_subset(c,L_{k-1}) then Delete c;
else add c to C_k;
return C_k;

Apriori Algorithm Improvement

Apriori Algorithm Deficiencies

According to the analysis of Apriori algorithm, association rule mining can generate association rules more efficiently, but there is also a serious flaw of inefficient algorithm.

The main reasons are as follows:
(1) Too many database scans, looking for every k-frequent itemsets (k = 1, 2, ..., k) requires scanning the database once and scanning k times in total. So when the database or k is too large, the algorithm will be too time-consuming or even impossible to complete.
(2) Pruning step of Apriori algorithm, c \in C_k, judging whether k (k-1) subsets of c are all in L_{k-1}, if we find a (k-1) subset is not in L_{k-1} to eliminate c. Because this process scans L_{k-1} many times, the efficiency of the algorithm is not ideal, especially when the generated C_k is large.

Apriori Algorithm Improvements

In order to improve the performance of Apriori algorithm, many variants of Apriori algorithm have been proposed to improve the efficiency of the original algorithm \[3\]. Some of these variations include the following: Hash-based techniques (hash entry into corresponding buckets), transaction compression techniques (compressing the number of transactions scanned for future iterations), partitioning techniques (partitioning data for finding candidate sets) Sampling techniques (mining of subsets of given data), dynamic itemset techniques (adding candidate sets at different points of the scan) \[4\].

This paper presents a hash mapping method based on the address of the candidate L_k in the hash technique. Hashing can significantly reduce the number of candidate sets to examine (especially 2 sets), and transaction compression can reduce the number of transactions in the database.

When frequent k-items L_k are generated from candidate item (k-1)-sets in C_{k-1}, L_{k-1} in each transaction set of C_{k-1} are sequentially connected to generate all k- is calculated by h (\{x, y\}), and the candidate L_k with the same address is hashed to the corresponding storage queue in the hash table. If the final value of the storage queue counter is smaller than the minimum support number, then the k-itemsets in this queue will not be frequent itemsets and can be deleted from the candidate k-itemsets. The specific algorithm is described as follows:

Enter: Transaction Database D.
Minimum support: min_ support.
L_1={frequent 1-itemsets(D)};       //A frequent itemset of length 1
C_1=database D;                  //Translates the set of transactions into a set of candidate transactions indexed by TID
for each all transactions t \in C_1 { //Determine the transaction t, TID contains the candidate
C_t={c \in L_1|(c-c[k]) \in t.set-of-itemset \land ((c-c[k-1]) \in t.set-of-itemset ) };
if (C_t \neq \phi) then { C_1’+=t.TID, C_1 > };
C_1=C_1’;
for (k=2;L_{k-1} \neq \phi;k++){
L_k=Apriori_ gen(L_{k-1},C_{k-1});     //Generate new frequent k-itemsets
C_k’=\phi;
for each all transactions t \in C_{k-1} { //Determine the transaction t, TID contains the candidate
C_t={c \in L_k|(c-c[k]) \in t.set-of-itemset \land ((c-c[k-1]) \in t.set-of-itemset ) };
if (C_t \neq \phi) then { C_k’+=t.TID, C_t > };
C_k=C_k’;}
return \( L_k = U_k L_{k-1} \);

Apriori_\text{gen}(L_{k-1}, C_{k-1}) \text{ function:}

\( L_{k-1} \): frequent \((k-1)\)-itemsets;

\( C_{k-1} \): Candidate Affairs Database;

Minimum support: \( n \_support \)

definition hashset= \{ \} // hashset table, the initial value is 0, the cumulative number of candidates for the same address
definition \( L_k = \{ \} \) // Store frequent \( k \)-candidate itemsets, including support

for each itemset \( l_i \in L_{k-1} \) \{ order of \( l_i \) = i // Record the serial number of \( l_i \) in \( L_{k-1} \) \}

for each all transactions \( t \in C_{k-1} \) \{ // the transaction \( t \) included in the project connection

for each itemset \( c_1 \in C_t \) \{for each itemset \( c_2 \in C_t \)

if \((c_1[1]=c_2[1]) \land \ldots \land (c_1[k-2]=c_2[k-2]) \land (c_1[k-1]<c_2[k-2])\)

then \( x=\text{order of } c_1 \);

\( y=\text{order of } c_2 \);

order=\text{hashset}[x \cdot 10 + y] \mod 7 ; // Hash function, determine the address in the hash table

\text{hashset}[\text{order}]+=1; // According to the order value \text{hashset}[\text{order}] assignment

\( c=c_1 \cdot c_2; \)

\( L_k[c]=\text{hashset}[\text{order}] \}

\}

for each itemset \( c \in L_k \) \{ if \( L_k[c]<\text{support} \) then delete \( c \); // Pruning \}

return \( L_k \);

**Algorithm Analysis Example**

Let \( T=\{100,200,300,400,500,600,700,800,900\} \), \( I=\{1,2,3,4,5\} \), \( \text{min\_support}=2 \), \( TID \) represent the transaction ID.

First, the set of transactions is converted into a candidate transaction set \( C_1 \) indexed by TID, the minimum support is 2, a frequent itemset \( L_1 \) of length 1 is generated, items less than the minimum support in \( C_1 \) are deleted and recorded as candidate transaction sets \( C_1' \). The process is as follows:

<table>
<thead>
<tr>
<th>TID</th>
<th>Item set</th>
<th>Item set</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1,2,5</td>
<td>{1}</td>
<td>6</td>
</tr>
<tr>
<td>200</td>
<td>2,4</td>
<td>{2}</td>
<td>7</td>
</tr>
<tr>
<td>300</td>
<td>2,3</td>
<td>{3}</td>
<td>6</td>
</tr>
<tr>
<td>400</td>
<td>1,2,4</td>
<td>{4}</td>
<td>2</td>
</tr>
<tr>
<td>500</td>
<td>1,3</td>
<td>{5}</td>
<td>2</td>
</tr>
<tr>
<td>600</td>
<td>2,3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>1,3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>1,2,3,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>1,2,3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first scan \( C_1 ' (C_1) \), respectively, the transaction TID items in the self-connection operation, the item number in \( L_1 \) to \( L_1 \) in the project sequence list, the establishment of hashset table. The process is as follows:

<table>
<thead>
<tr>
<th>( L_1 ) in the project sequence</th>
<th>\hline Item set</th>
<th>serial number</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{2}</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>{4}</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

hashset table

100 \{1,2\}, \{1,5\}, \{2,5\}
200 \{2,4\}
300 \{2,3\}
400 \{1,2\}, \{1,4\}, \{2,4\}

165
According to the hash function to create the following hashset table:

<table>
<thead>
<tr>
<th>Hash address</th>
<th>Hashset table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{15, 15}</td>
</tr>
<tr>
<td>2</td>
<td>{23}</td>
</tr>
<tr>
<td>3</td>
<td>{24}</td>
</tr>
<tr>
<td>4</td>
<td>{25}</td>
</tr>
<tr>
<td>5</td>
<td>{12}</td>
</tr>
<tr>
<td>6</td>
<td>{13}</td>
</tr>
</tbody>
</table>

The value in the hashset table represents the number of items hashed to the hashset [order], and a frequent 2-itemset set \(L_2\) can be obtained. Delete items in \(C_2\) less than the minimum support, denoted as candidate transaction set \(C_2'\), as follows:

\[L_2\] in the project sequence

<table>
<thead>
<tr>
<th>Item set</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>4</td>
</tr>
<tr>
<td>{1 3}</td>
<td>4</td>
</tr>
<tr>
<td>{1 5}</td>
<td>2</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 4}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

\[C_2' (C_2)\]

<table>
<thead>
<tr>
<th>TID</th>
<th>Item set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{{1 2}, {1 5}, {2 5}}</td>
</tr>
<tr>
<td>200</td>
<td>{{2 4}}</td>
</tr>
<tr>
<td>300</td>
<td>{{2 3}}</td>
</tr>
<tr>
<td>400</td>
<td>{{1 2}, {2 4}}</td>
</tr>
<tr>
<td>500</td>
<td>{{1 3}}</td>
</tr>
<tr>
<td>600</td>
<td>{{2 3}}</td>
</tr>
<tr>
<td>700</td>
<td>{{1 3}}</td>
</tr>
<tr>
<td>800</td>
<td>{{1 2}, {1 3}, {1 5}, {2 3}, {2 5}}</td>
</tr>
<tr>
<td>900</td>
<td>{{1 2}, {1 3}, {2 3}}</td>
</tr>
</tbody>
</table>

The second scan \(C_2' (C_2)\), the same number of items in \(L_2\), respectively, the transaction TID items in the self-connection operation, the establishment of the following hashset table:

<table>
<thead>
<tr>
<th>L2 in the project sequence</th>
<th>Hashset table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item set</td>
<td>serial number</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>3</td>
</tr>
<tr>
<td>{2 3}</td>
<td>4</td>
</tr>
<tr>
<td>{2 4}</td>
<td>5</td>
</tr>
<tr>
<td>{2 5}</td>
<td>6</td>
</tr>
</tbody>
</table>

\{1 2 5\} is generated by \{1 2\}, \{2 5\}; \{1 2 3\} is generated by \{1 2\}, \{2 3\}; hashset table is created as follows:

<table>
<thead>
<tr>
<th>Hash address</th>
<th>hashset table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>{1 2 3}</td>
<td>{1 2 5}</td>
</tr>
</tbody>
</table>

Finally get the following frequent 3-itemset set \(L_3\):

\{1 2 5\}
End

**Time Complexity Analysis of Aprioritid** [5] Algorithm Based on Hash Mapping

Based on the AprioriTid algorithm, a candidate set \( C_k \) is generated by using \( L_{k-1} \times L_{k-1} \). When \( L_{k-1} \) is large, the obtained \( C_k \) will be very large. No matter how pruning is optimized, the efficiency will not be greatly improved and the time complexity is \( O(n^2) \). After that, the transaction in \( C_{k-1} \) is scanned and the support of each candidate itemset in \( C_k \) is calculated to find the frequent itemsets \( L_k \) whose support is not less than the minimum support, and the time complexity is \( O(mn) = O(n^2) \). And so on, with the reduction of transactions and \( L_{k-1} \) in \( C_{k-1} \), the time complexity decreases.

The reason that the AprioriTid algorithm based on the hash mapping method can reduce the time complexity is that it sequentially connects \( L_{k-1} \) in each transaction set of \( C_{k-1} \), which greatly reduces the value of \( n \), so the time complexity \( O(n^2) \) is also reduced. The frequent \( k \)-itemsets \( L_k \) are obtained at the same time as the hashes, eliminating the time it takes to find frequent itemsets \( L_k \) whose support is not less than the minimum support.

**Conclusion**

Association rules mining problem is divided into two sub-problems: Discover frequent itemsets and generate association rules. Among them, finding frequent itemsets is the key to mining association rules. In this paper, we propose a hash algorithm based AprioriTid algorithm, which can solve frequent itemsets more quickly. As we concatenate \( L_{k-1} \) in each transaction set of \( C_{k-1} \) sequentially, we can greatly reduce the value of \( n \) and reduce the time complexity \( O(n^2) \) can get the frequent \( k \)-itemsets \( L_k \) while hashing, which eliminates the time it takes to find frequent itemsets \( L_k \) whose support is not less than the minimum support. Finally, compared with AprioriTid, the experiment proves that AprioriTid algorithm based on hash mapping method is effective.

**Acknowledgement**


**References**


