Enhanced Candidate Generation for Frequent Item Set Generation

M. Krishnamurthy1*, K. Manivannan2, A. Chilambuchelvan3, E. Rajalakshmi4 and A. Kannan5

1CSE, KCG College of Technology, Chennai - 600097, Tamil Nadu, India; mkrish@kcgcollege.com
2IT, RMK Engineering College, Kavaraipettai, Thiruvarur District - 601206, Tamil Nadu, India; manivannanid@gmail.com
3RMK Engineering College, Kavaraipettai, Thiruvarur District - 601206, Tamil Nadu, India; chill97@gmail.com
4CSE, SVCE, Sriperumbudur - 602117, Tamil Nadu, India; eraji@svce.ac.in
5IST, Anna University, Chennai - 600025, Tamil Nadu, India; kannan@annauniv.edu

Abstract

Frequent item sets is one of the most investigated fields of data mining. The significant feature is to find new techniques to reduce candidate item sets in order to generate frequent item sets efficiently. This paper introduces an efficient algorithm called Enhanced Candidate Generation for Frequent item set Generation (ECG for FIG) for finding frequent item sets from large databases. The existing algorithm for frequent item set generation scan the original database more than once, use more storage space, take more processing time. The proposed algorithm gives a solution to this by representing the transactions in the database with decimal numbers instead of binary values and strings. The original database is scanned only once and is converted into an equivalent decimal value to reduce the storage space. The subset generation concept is used to generate frequent item sets. Thus the proposed algorithm reduces the scanning time, processing time and the storage space respectively. When compared with the existing algorithms, the proposed algorithm takes very less execution time and memory. When implemented the algorithm using java and tested with WEKA tool, for 400 transactions of twenty five items, ECG for FIG is taking only 800 bytes of memory and 200000000 ns (two seconds), whereas all the other above mentioned algorithms are taking 20800 bytes of memory and more than two seconds.

Keywords: Decimal Conversion, Frequent Item Set Generation, Redundancy Elimination, Storage Space Reduction

1. Introduction

Mining of frequent item sets is an admired and important domain in data mining for analyzing data sets. Let \( N = \{N_1, N_2, N_3, \ldots, N_m\} \) be a set of items. Let \( D \) be the transactional database, where each transaction \( T \) is a set of items such that \( T \cap N \). Each transaction is associated with an identifier TID. A set of items is referred as item set. An item set that contains k-items is a k-item set. If an item set, I occurs for multiple times (normally \( > 1 \)), then it is called as frequent item set. If the support count of an item set I, satisfies the minimum support threshold, then the item set I is a frequent item set.

2. Literature Survey

The Association Rule mining was raised by R. Agarwal. The Apriori algorithm reveals useful itemsets and generates association rules. But, this algorithm generates enormous amount of candidates and the original database is scanned for many times. To avoid this disadvantage many effective algorithms for discovering association rules are introduced. In improved Apriori algorithm, the mining efficiency is unsatisfactory as concerned with memory for the database.

In the past, the omission of time dimension in association rule was very clearly mentioned by Banu Ozden...
et al. Different strategies were proposed after Apriori as in FP-growth, which outperforms all candidate generations, but still have problems in the case of no common prefixes within the data items. The temporal FP tree uses divide and conquer technique for construction and the traversing of tree which is used to decompose the mining task into a set of smaller task which reduces the search space. However, the Temporal FP Tree technique is better only when the data is dense. But, the performance is very less. Kuen-Fang Jea et al. proposed an efficient and flexible online algorithm to discover large item sets for association rules without presenting a default threshold. It uses a combinatorial approximation technique which is suitable for online mining applications. Even though the Online Combinatorial Approximation (OCA) and Find Item sets algorithms use different data for online mining, the Find Item sets take count of all large item sets but OCA needs the counts of all one, two-item sets for approximation. These lead to inaccurate results.

Yuh-Jiuan Tsay et al. proposed a CBAR algorithm which creates partial cluster tables for experimenting the Food Mart transaction. The large item sets can be immediately mined even when the database is very large. CBAR only requires a single scan of the transaction database, followed by contrast with the partial cluster tables. This not only prunes considerable amounts of data, reducing the time needed to perform data scans and requiring less contrast, but also ensures the correctness of the mined results. But it uses more number of cluster tables and does not concentrate on storage management.

Yeong-Chyi Lee et al. proposed an efficient algorithm Mining association rules with multiple minimum supports using maximum constraints where the minimum support for an item set is set as the maximum of the minimum supports of the items contained in the item set and the characteristic of level-by-level processing is used such that the original Apriori algorithm can be easily extended to find the large item sets. The granular computing technique of bit strings is also used to speed up the proposed data mining algorithm. It does not take any steps to reduce the times of scanning the transaction database.

Jie Dong et al. address an algorithm which uses a special data structure called Bit Table for compressing the database and for the candidate item sets generation. Mining frequent item sets in transaction databases, time-series database and many other kinds of databases is an important task. This paper proposes an elective algorithm named as BitTableFI. In the algorithm, a special data structure BitTable is used horizontally and vertically to compress database for quick candidate item sets generation and support count. The algorithm can also be used in many Apriori-like algorithms to improve the performance. However, it does not take any pruning or other technique to reduce the size of candidate item sets and the times of scanning the transaction database.

M. Krishnamurthy et al. proposed an algorithm which incorporates a flexible support count during mining and it satisfies two goals: it eliminates the computations after the processing phase, and it suits for the need of users. So, a frequency threshold which is flexible is used by considering real time transactions and it provides flexibility for effective decision making in many applications. It also provides increasing memory performance.

M. Krishnamurthy et al. proposed a new algorithm for generating frequent item sets without new candidate generation. This algorithm focuses mainly on reducing query frequencies and storage resources. Suppose in this algorithm, if we compute frequency of frequent k-item sets from (k-1) item sets and if k is greater than size of transaction T, then there is no need to scan transaction T which is generated by (k-1) item sets.

M. Krishnamurthy et al. proposed an improved algorithm ICBV for finding frequent item sets. The original database is read only once and the transactions are changed into bit vectors (0 or 1). The duplicate transactions are avoided by fixing a counter variable and cluster table is created depending upon the item presence. The algorithm work efficiently when compared with Apriori and CBVAR. However, the transaction table needs more storage space and it uses many intermediate tables to generate the final frequent item sets.

M. Krishnamurthy et al. proposed a new algorithm CBVAR for generating frequent item sets using a single scan towards the database. This algorithm reduces the computation time and uses less candidate generation. All the items in a transaction are changed into bits 0’s or 1’s. If the item is present, then it is marked as 1 else it is marked as 0. A cluster is created by reading the original data set for one time. The cluster table gives out the frequent 1-item sets. The frequent k-item sets are obtained by using Logical AND between the items in a cluster table. But, CBVAR
algorithm is not concentrating on redundancy elimination and rule generation. It uses multiple set of tables to find the frequent item sets. It uses k-tables to generate k-frequent item sets which leads to wastage of time and memory.

Xiaojun Cao\textsuperscript{15} addressed an algorithm AGC_AR. This algorithm produces the association rules based on granular computing which improves feasibility and effectiveness. This algorithm decomposes the information system in to logical operations of elementary particles in accordance with the requirements, in turn gives out the frequent item sets. However, this algorithm uses more number of candidate generations and scans the database for obtaining the support of each candidate.

Kamlesh Malpani et al.\textsuperscript{6} proposed a new algorithm to avoid joining and pruning which is based on Logical Operation (AND, OR). This effective algorithm is used for mining association rules in large databases. This algorithm compute frequent item sets using logic OR and AND operations.

Atefe Ramezani, et al.\textsuperscript{1} presented an algorithm that reduces confidence of sensitive rules to below minimum threshold by eliminating selective item among items of consequent sensitive rule (R.H.S.) for all selective transactions. Bharathi T, et al.\textsuperscript{3} presented a Modified Artificial Fish Swarm Algorithm (MFSA) which has many benefits that includes higher convergence rate, flexibility, fault tolerance and high accuracy. Chandrasekar Ravi, et al.\textsuperscript{4} proposed a technique called Negative and Positive Fuzzy Association Rule Mining (NP-FARM), which mines both negative and positive association rules from a fuzzy transaction dataset.

3. Proposed Work

In order to reduce the space and time in mining frequent item set, we introduce a new algorithm called Enhanced Candidate Generation for Frequent Item set Generation (ECG for FIG) which is the improved version of Improved Cluster Based Mining Algorithm for Frequent Items Set Generation using Bit Vectors (ICBV)\textsuperscript{10}. In the existing algorithm frequent item sets are generated by scanning the original database for many times and transactions are given in the form of bit vectors. As cluster tables are created depending on number of one's it increases processing time and storage requirements.

4. Proposed Algorithm

This section explains the ECG for FIG algorithm for implementation.

Input: Transaction database
Output: Frequent Item sets

begin
{ Convert the given transaction database to its equivalent decimal value.
Get the minimum threshold (min_thres)
In the table, combine the identical rows into one row and increment CNT accordingly until all rows are unique.
Calculate support count for each items where,
\begin{align*}
A: \text{support count (sc)} &= \frac{\text{(Presence of item in each transaction)}}{\text{(Total no. of transactions)}}
\end{align*}
Print frequent 1-item sets
else
Delete the item
} Determine frequent k-item sets
begin
{ Generate all the subsets that are related to the given itemsets.
Set min_thres to new min_thres for each frequent itemsets are needed.
Calculate sc for each subsets using A.
if sc > min_thres
{ Print the frequent itemsets
else
Delete the items.
} } } } } }

4.1 Decimal Conversion

The given transaction database is converted to their equivalent decimal values. Suppose a database contains 3-items; then the possible transactions are $2^3 = 8$ combinations which are 000, 001, 010, 011, 100, 101, 110 and 111. The corresponding decimal values are 0, 1, 2, 3, 4, 5, 6, 7 respectively. Let us illustrate with an example. Con-
Consider a table consisting of eight transactions consisting of three items Apple, Orange and Grapes which are denoted as A, O, G respectively for convenience.

Table 1 represents transactions containing items with its bit vector and its equivalent decimal values. The Table 2 represents the space requirements of Table 1 when used in Java. Instead of storing the transactions using string or bit vectors, the equivalent decimal values can be stored to reduce the storage space. In turn the scanning time will also be reduced.

Figure 1 shows the memory requirements of string, bit vectors and the decimal values.

### 4.2 Subset Generation

A subset is a part of a set. If every element of A is a element of B, then we can say A is a subset of B. This can be denoted as $A \subseteq B$.

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$  \hfill (1)

#### Table 1. Transaction table with Bit vector

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Bit Vectors</th>
<th>Decimal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>O</td>
</tr>
<tr>
<td>T1</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>G</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T3</td>
<td>O</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>O,G</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T5</td>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T6</td>
<td>A,G</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T7</td>
<td>A,O</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T8</td>
<td>A,O,G</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Table 2. Space requirements

<table>
<thead>
<tr>
<th>TID</th>
<th>String</th>
<th>Bit Vector</th>
<th>Decimal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T2</td>
<td>6</td>
<td>1+1+1=3</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>6</td>
<td>1+1+1=3</td>
<td>2</td>
</tr>
<tr>
<td>T4</td>
<td>6+6=12</td>
<td>1+1+1=3</td>
<td>3</td>
</tr>
<tr>
<td>T5</td>
<td>6</td>
<td>1+1+1=3</td>
<td>4</td>
</tr>
<tr>
<td>T6</td>
<td>6+6=12</td>
<td>1+1+1=3</td>
<td>5</td>
</tr>
<tr>
<td>T7</td>
<td>6+6=12</td>
<td>1+1+1=3</td>
<td>6</td>
</tr>
<tr>
<td>T8</td>
<td>6+6+6=18</td>
<td>1+1+1=3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>63</td>
<td>7*3=21</td>
<td>7*2=14</td>
</tr>
</tbody>
</table>

### 4.3 Sample Transaction

Implementation of ECG for FIG: Let us assume the following Pharmacy table as transaction database which contains names of tablets. Table 3 consists of 18 transactions where they are converted to their equivalent decimal values. For convenience, the real time items Benoquin, Dialyte, Ibuprofen, Nutradrops and Veetids have been named as A, B, C, D and E respectively in the upcoming tables.

Total Memory Size of Table 3 = bytes

Total Memory Size of Table 4 = 36 bytes

If the transaction pattern is same for multiple transactions, then the transactions are combined into a single transaction and the count value is updated accordingly. Instead of storing the transaction pattern similar to the already existing transaction, they are stored in a single transaction. Those duplicates transactions are...
deleted which reduce the storage space. For example in Table 4, transactions T3, T7, T12 the transaction patterns are same and are combined and count value is updated to three.

Then support threshold is computed using the formula.

\[
\text{Support threshold for an item } i = \frac{\text{Presence of item } i \text{ in each transaction}}{\text{Total no. of transactions}}
\]  

(2)

Support threshold for items A, B, C, D and E are: \( SC_A = 10 / 18 = 50\% \), \( SC_B = 8 / 18 = 40\% \), \( SC_C = 14 / 18 = 70\% \), \( SC_D = 9 / 18 = 45\% \), \( SC_E = 9 / 18 = 45\% \)

For example, if minimum threshold is 45 percent, except B all are frequent 1-item sets. This algorithm uses the horizontal representation so the memory space is reduced. Table 6 denotes frequent 1-item sets formed by eliminating the repetitive transactions. Then subset generation is used to generate frequent k-item sets. Based on the Table 7, support threshold is calculated using the formulae (2).
Enhanced Candidate Generation for Frequent Item Set Generation

<table>
<thead>
<tr>
<th>Items</th>
<th>Transactions</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>28,23,20,21,30,18,26,29,19,22</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>28,12,5,23,20,21,13,30,9,14,22</td>
<td>14</td>
</tr>
<tr>
<td>D</td>
<td>23,30,18,26,6,14,19,11,22</td>
<td>9</td>
</tr>
<tr>
<td>E</td>
<td>5,23,21,13,29,19,11</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 7. Frequent item sets using Subset Generation

In the Table 7 repetitive transactions are eliminated. The minimum threshold for frequent 2-item sets is 30 percent and for frequent 3-item sets be 15 percent. The support threshold which is greater than the minimum threshold, then it is considered as frequent item set.

The same procedure is applied for the generation of all other item set. Instead of generating newer tables, the same table is used for generating item sets. Thus, the frequent two item sets are \{A, C\}, \{A, D\}, \{C, E\} and the frequent three-item sets are \{A, C, D\}, \{A, C, E\}.

4.4 Memory Requirements

Table 8 compares the memory required for each table specified above. The string table occupies more space. Whereas the decimal table occupies less storage space compared to other tables.

Table 8. Memory utilization

<table>
<thead>
<tr>
<th>Table Type</th>
<th>String Table</th>
<th>Bit Vector Table</th>
<th>Decimal Table</th>
<th>Decimal Table After Duplicate Removal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>250</td>
<td>180</td>
<td>36</td>
<td>32</td>
</tr>
</tbody>
</table>

In the Table 7 repetitive transactions are eliminated. The minimum threshold for frequent 2-item sets is 30 percent and for frequent 3-item sets be 15 percent. The support threshold which is greater than the minimum threshold, then it is considered as frequent item set.

The same procedure is applied for the generation of all other item set. Instead of generating newer tables, the same table is used for generating item sets. Thus, the frequent two item sets are \{A, C\}, \{A, D\}, \{C, E\} and the frequent three-item sets are \{A, C, D\}, \{A, C, E\}.

5. Conclusion

The proposed algorithm uses single scan, which reduces the database scans, so that computation time is also very less. Moreover, transactions are converted to equivalent decimal value and it uses a horizontal representation of transaction database. Hence the storage space used is very less. Future work in this could be the use of this algorithm for high utility dataset and for dense data set where memory and scanning time plays a major role.

6. References


