CHAPTER 2

SINGLE SCAN FREQUENTSET GENERATION
IN ASSOCIATION RULE MINING

2.1 Introduction

Data mining is used in searching databases for unknown patterns and information [1]. In the business world, the behaviour of customers is used as a tool for promoting business [1, 8, 9]. Mining frequent items is a basic and important step in data mining applications. Different approaches are used to tackle the computationally intensive process of generating the frequent itemsets. These algorithms use bottom-up, top-down, or both the approaches. The bottom-up approach starts the frequent itemset generation from 1-length itemsets and proceeds towards the n-length itemsets. Its performance gets degraded when the length of the itemsets increases. The top-down approach performs the process in the opposite direction and is more suitable when the transaction length is long enough. Some algorithms use the special features of both the approaches.

The Apriori algorithm is a well known data mining algorithm [9]. This algorithm really paved the way for different versions of data mining algorithms. The Apriori and its derivatives also use the support-confidence constraints to produce association rules. The efficiency of Apriori deteriorates when

i) the user-specified minimum support is small and
ii) the length of the transactions is long.

Frequent Pattern growth is a tree projection algorithm [65] that uses the frequent pattern tree structure, which has all the transactions of database in a tree projection algorithm [65] that uses the frequent pattern tree structure, which has all the transactions of database in a

The technique proposed and discussed in this chapter has been published as a paper [f].
compressed form. It straightaway produces the frequent itemsets without candidate generation. However, it requires complete reconstruction of the frequent pattern tree structure for different support thresholds. *OPUS* [82] is also a mining algorithm, which produces the association rules without candidate generation.

The approach proposed in this chapter also works on the basis of support-confidence constraints. But the special feature of the approach here is that the scanning of database is carried out only once and the database need not be scanned any further.

Even though different versions of Apriori algorithms are available, the proposed work is compared with the performance of pure Apriori algorithm on a synthetic database. The rest of the chapter is organized as follows: Section 2.1.1 briefly reviews concepts in association rule mining, Section 2.2 explains the proposed approach, Section 2.3 compares the performance of the proposed algorithm with *Apriori* algorithm, and Section 2.4 presents a summary.

### 2.1.1 Rule mining

The process of association rule mining is an unsupervised data mining on a large volume of data [1]. The rules generated from the historical data, show how the presence of one item influences the presence of another data item in the data source. Now-a-days, these rules are mainly used in enhancing business activities [1, 20].

The process of producing association rules involves the task of finding the set of all the frequent itemsets and generating promising rules and making use of user-specified thresholds for obtaining the same.

Let us consider a typical association rule given below.

\[ X \rightarrow Y \], where \( X \), a set of items, is the antecedent and \( Y \), another set of items, is the consequent.
The support of each itemset is defined as:

\[ \text{Support}(X \cup Y) = \frac{|X \cup Y|}{|D|} \times 100, \quad X, Y \subseteq I \quad (2.1) \]

Similarly,

\[ \text{Confidence}(X \cup Y) = \frac{|X \cup Y|}{|X|} \times 100, \quad X, Y \subseteq I \quad (2.2) \]

Let \( D \) be a database of transactions. An itemset with \( k \)-items is called a \( k \)-itemset. An itemset is said to be frequent if its support is greater than a user-specified minimum value, \( \text{min} \_ \text{sup} \). A rule is said to be strong, if its confidence is greater than a user-defined minimum value, \( \text{min} \_ \text{conf} \).

**Apriori algorithm**

This is the algorithm which is the basis for all association rule mining algorithms. It is a bottom-up search, which traces all frequent itemsets in a horizontal manner. It is noteworthy that the candidates of length \( k \) can be formed only if the frequent itemsets of length \( (k-1) \) are not null. The Apriori algorithm consists of the following three steps[9].

1) Candidates of length \( k \) are generated by self-joining the frequent \( (k-1) \) length itemsets whose prefix of length \( (k-2) \) is the same.

2) Calculation of the support of each itemset of length \( k \) by scanning all the transactions in the database.

3) Pruning the candidates having less than the user-specified minimum support in order to reduce the number of candidates. The itemsets above the threshold are called frequent itemsets of length-\( k \).

**2.2 The Proposed Technique**

A pictorial representation of the proposed approach is given in Fig. 2.1. This approach uses a table data structure and a vector data structure called the \textit{itemsupport}, kept in the main memory. The table has two columns – the first
column for itemsets, and the second one for the number of occurrences of the itemset (Table 2.1). There are \( m \) rows in the table where \( m \) is the number of distinct itemsets. An element of the itemsupport vector indicates the support of each item, i.e., the number of occurrences of that item in the transactional database.

First, the itemsupport vector is initialized to all zeros. The procedure starts by scanning all the transactions once in the database. As each transaction is scanned, the corresponding itemset is entered in the first column of the table if it is not already present and its count is marked as 1 in the second column. If the itemset is already in the table, the count (column 2) of the corresponding itemset entry is incremented by 1. Similarly, the support of each element involved in the transaction in the itemsupport vector is incremented appropriately.

![Fig. 2.1 Single scan mining algorithm.](image)

First, the itemsupport vector is initialized to all zeros. The procedure starts by scanning all the transactions once in the database. As each transaction is scanned, the corresponding itemset is entered in the first column of the table if it is not already present and its count is marked as 1 in the second column. If the itemset is already in the table, the count (column 2) of the corresponding itemset entry is incremented by 1. Similarly, the support of each element involved in the transaction in the itemsupport vector is incremented appropriately.
Example: Consider a database given in Table 2.1 that contains transactions on five items \( \{a, b, c, d, e\} \).

<table>
<thead>
<tr>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>( abd )</td>
</tr>
<tr>
<td>T200</td>
<td>( abde )</td>
</tr>
<tr>
<td>T300</td>
<td>( abcde )</td>
</tr>
<tr>
<td>T400</td>
<td>( abd )</td>
</tr>
<tr>
<td>T500</td>
<td>( abd )</td>
</tr>
<tr>
<td>T600</td>
<td>( abcde )</td>
</tr>
<tr>
<td>T700</td>
<td>( abcde )</td>
</tr>
<tr>
<td>T800</td>
<td>( abc )</td>
</tr>
<tr>
<td>T900</td>
<td>( abde )</td>
</tr>
<tr>
<td>T1000</td>
<td>( acde )</td>
</tr>
</tbody>
</table>

After scanning the first transaction, the itemset \( abd \) is entered in the table with the count value of 1. The itemsupport vector is also incremented appropriately to \([1 \ 1 \ 0 \ 1 \ 0]\). After reading all the transactions, the contents of the table and the itemsupport vector are as given in Table 2.2 and itemsupport vector respectively.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( abd )</td>
<td>1</td>
</tr>
<tr>
<td>( abde )</td>
<td>2</td>
</tr>
<tr>
<td>( abcde )</td>
<td>3</td>
</tr>
<tr>
<td>( abcd )</td>
<td>2</td>
</tr>
<tr>
<td>( abc )</td>
<td>1</td>
</tr>
<tr>
<td>( acde )</td>
<td>1</td>
</tr>
</tbody>
</table>
After scanning the database once and constructing the table and the itemsupport vector, those single items in the itemsupport vector with support less than the specified min_sup value are removed from the table and itemsupport vector. The remaining items in the itemsupport vector are grouped into a set called the Frequent Items.

For example, if min_sup value is 70% then all items in the itemsupport vector whose count is less than 70% are not considered, as infrequent items cannot have supersets that are frequent. Moreover, all infrequent items in entries of the table are eliminated and the resulting patterns, if found to be the same, are merged to form the new reduced table entry with its supports summed to form the new count.

The itemsupport vector containing the frequent items alone is as,

\[ \text{itemsupport vector: } [10 \ 10 \ 7 \ 9 \ \cdot] \]

On eliminating the infrequent elements (i.e., item \( e \)) the table gets reduced as shown below,

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( abd )</td>
<td>1</td>
</tr>
<tr>
<td>( abd )</td>
<td>2</td>
</tr>
<tr>
<td>( abcd )</td>
<td>3</td>
</tr>
<tr>
<td>( abcd )</td>
<td>2</td>
</tr>
<tr>
<td>( abc )</td>
<td>1</td>
</tr>
<tr>
<td>( acd )</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( abd )</td>
<td>3</td>
</tr>
<tr>
<td>( abcd )</td>
<td>5</td>
</tr>
<tr>
<td>( abc )</td>
<td>1</td>
</tr>
<tr>
<td>( acd )</td>
<td>1</td>
</tr>
</tbody>
</table>

Now the generation of frequent itemsets is explained below using only the reduced table. This procedure uses the top-down approach. Since an itemset is
present in all its supersets, the number of occurrences of an itemset can be obtained by adding the number of occurrences of all its supersets (including itself) in the database of transactions.

Hence,

\[
s(C_k), \text{ support of a candidate itemset } C_k \text{ of length } k = (\text{number of occurrences of itemset } C_k) + (\text{number of occurrences of all supersets of } C_k \text{ of length } k+1) + (\text{number of occurrences of all supersets of } C_k \text{ of length } k+2) + \ldots + (\text{number of occurrences of all supersets of } C_k \text{ of length } n), \quad (2.3)
\]

\[k = 1, 2, \ldots, n, \text{ where } n \text{ is the number of all the items.}\]

The value of each term on the RHS of the Equation (2.3) can be obtained from the table of occurrences of itemsets (Table 2.4) through string comparison.

For example, referring to the above database given in Table 2.1,

support of the itemset \( abd \)

\[= \text{occurrence of } abd + \text{occurrence of supersets of length 4(i.e.,abcd)} \]
\[= 3 + 5 \]
\[= 8 \text{ transactions} \]
\[= 8/10 \times 100 \]
\[= 80\% \]

Similarly, the support of \( abcd \)

\[= \text{occurrences of } abcd \]
\[= 5 \text{ transactions} \]
\[= 5/10 \times 100 \]
\[= 50\% \]
Similarly, the support of \( abc \)
\[
= \text{occurrence of } abc^+ \text{ occurrence of supersets of length } 4 (\text{i.e., } abcd)
\]
\[
= 1 + 5
\]
\[
= 6 \text{ transactions}
\]
\[
= \frac{6}{10} \times 100
\]
\[
= 60\%
\]

Support of all the remaining itemsets are calculated in a similar manner.

The supports of all itemsets are computed by scanning the table once for each itemset. Then, those itemsets whose supports are greater than the specified minimum value are selected as the frequent itemsets. Association rules can then be formulated from the frequent itemsets, following the usual procedure \[1\].

In this technique, multiple scans of the transactional database are avoided. But it necessitates multiple scans of the table in the main memory. This technique is much faster than the Apriori method because

1) scanning the table in the main memory is much faster than scanning the database in the disk.

2) the number of entries of itemsets in the table is less than the number of transactions, which in the worst case could be \( 2^n \) entries. In practice, the number of distinct itemsets of transactions is much less than \( 2^n \).

The following algorithm gives the logical sequence of the proposed work.

Algorithm

i) Initialize all entries in \( \text{itemsupport vector} \) to zero.

ii) Read a new transaction \( t \) from the database \( D \).

iii) For each item in the transaction \( t \), increment the corresponding support value in the \( \text{itemsupport vector} \) by 1.

iv) For each pattern \( p \) identified,
a. If $p$ is a newly identified pattern, make it a new table entry and initialize its count to 1.
b. If $p$ is an already existing pattern, then increment corresponding count by 1.
v) If $D$ is not null, goto step ii 
vi) Eliminate the infrequent items in the \textit{itemsupport vector} and form the frequent items.
vii) Determine the frequent itemsets from the table by calculating the support using the formula,

$$\text{Support}(p) = \text{count}(p) + \text{count(supersets of } p)$$ \hfill (2.4)

\section*{2.3 Performance Evaluation}

This section gives details of the experimental evaluation of the performance of the proposed algorithm.

\subsection*{2.3.1 Experimental setup and results}

An analysis was made to evaluate the performance of the proposed technique with pure \textit{Apriori} algorithm on a Pentium IV with 256 MB RAM running on Windows 2000 OS. The time taken to generate frequent itemsets by each of the algorithm is shown in Fig. 2.2, 2.3, and 2.4

\begin{center}
\includegraphics[width=0.5\textwidth]{chart.png}
\end{center}

(a) \textit{Apriori}
Fig. 2.2 Performance evaluation for T5.I3.D100K.

(a) Apriori

(b) Proposed

Fig. 2.3 Performance evaluation for T10.I4.D100K.

(b) Proposed
Datasets for simulation are created synthetically as suggested by [9], with following parameters: the average size of the transactions $|T|$, the average size of the maximal potentially large itemsets $|l|$, and the number of transactions $|D|$. Three datasets T5.I3.D100K, T10.I4.D100K, and T15.I5.100K have been used to measure the performance of the proposed method. In the first dataset, $|T|$ is 5, $|l|$ is 3, and $|D|$ is 100K. For the second dataset, the values of $|T|$, $|l|$, and $|D|$ are chosen as 10, 4, and 100K. In the third dataset, the values of $|T|$, $|l|$, and $|D|$ are 15, 5, and 100K.

The simulation results show that the time complexity of the Apriori algorithm reduces when the support is high and increases when the support is
low which is clearly known from the Fig. 2.2a, 2.3a and 2.4a. Fig. 2.2a shows the
time requirement of Apriori algorithm for the supports varying from 10 to 50 on
the dataset, T5.I3.D100K. It consumes time from 300 to 50 seconds. Thus the
best time is 50 seconds for this dataset.

Fig. 2.3a shows the time requirement of the Apriori algorithm on the
second dataset, T10.I4.D100K. Here, the time consumption is between from 400
to 180 seconds for the supports varying from 10 to 50. Thus the best time is 180
seconds for the second dataset. For the third dataset T15.I5.100K, the time varies
from 711 seconds to 520 seconds. Thus the best time is 520 seconds for third
dataset.

Fig. 2.2b, 2.3b, and 2.4b show the time complexity of the proposed
method. It is constant for all support values. The time required to produce
frequent sets is 2.14 seconds, 4.02 seconds, and 23.4 seconds for the datasets

Fig. 2.5 shows the comparison of the best times achieved by the Apriori
for the three datasets and the time taken by the proposed algorithm.

![Graph showing time comparison]

**Fig. 2.5 Comparison of the best performance of the Apriori algorithm and the performance of proposed algorithm.**
2.3.2 Discussion and limitations

The space complexity and time complexity are minimal when the dataset is dense. In the case of sparse dataset, the amount of memory required for processing is high. It is minimized by reconstructing the table with the help of frequent items available in Itemsupport vector.

2.4 Summary

This proposed technique determines the frequent itemsets with a single scan of the transactional database. During this single database scan, the information about itemsets and their occurrences are captured into a table kept in the main memory. Based on the support threshold value, items which do not satisfy the threshold value are removed from the table entries and the new reduced table is formed. While determining the frequent itemsets, this table is scanned instead of the database. This results in considerable reduction of the time taken for computation.

This chapter has discussed a fast method for generating frequent itemsets and the next chapter discusses a new method for fast generation of maximal frequent itemsets.