APPENDIX 1

DATASET LAYOUT AND DATA STRUCTURE

The general characteristics of the datasets used by the association rule mining algorithms are presented. Also, the data structure used by the proposed algorithms is explained.

DATASET LAYOUT

Most of the association rule mining algorithms works with transactional datasets. However other formats of datasets can also be used. Theoretically, a transactional database is a two-dimensional matrix where the rows represent individual customer purchase transactions and the columns represent the items on sale. This matrix can use two kinds of layouts: Horizontal and Vertical data layout.

Horizontal Data Layout

Horizontal data layout can be implemented in the following two different ways, which are shown in Figure A 1.1:

Item-vector

The database is organized as a set of rows with each row storing a transaction identifier (TID) and a bit-vector of 1's and 0's to show the presence or absence of items on sale.
**Item-list**

Each row consists of an ordered list of item-identifiers (IID) and the items actually purchased in the transaction.

<table>
<thead>
<tr>
<th>TID</th>
<th>ItemID</th>
<th>TID</th>
<th>ItemID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 1 1 ---</td>
<td>1</td>
<td>1 4 5 -</td>
</tr>
<tr>
<td>2</td>
<td>0 1 1 0 1 -- -</td>
<td>2</td>
<td>2 3 5 -</td>
</tr>
<tr>
<td>-</td>
<td>- - - - - - -</td>
<td>-</td>
<td>- - - -</td>
</tr>
<tr>
<td>a) Item-Vector</td>
<td>b) Item-List</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure A 1.1 Horizontal Data layout**

**Vertical Data Layout**

Vertical data layout can be implemented in the following two different ways, which are shown in Figure A 1.2.

**TID-Vector**

The database is organized as a set of columns with each column represents the IIDs and each row contains a bit-vector of 1's and 0's to represent the presence or absence, respectively.

**TID-List**

Each column consists of an ordered list of only the TIDs of the transactions.
While a mining algorithm is free to dynamically change the database layout during the mining process, this thesis considered the database in horizontal item-list format.

TEST DATASET DESCRIPTION

Usually, the datasets are of two types: Sparse dataset and dense dataset.

Sparse Dataset

Sparse dataset consists of data for which only a small fraction of the attributes are non-zero or non-null in any given row. Examples of sparse data include market basket and text mining data. For example, a market basket problem, there might be 1,000 products in the company’s catalog, and the average size of a basket is 10 products. In this example, a record has on average 10 out of 1000 attributes that are not null. This implies that the fraction of non-zero attributes on the table is 10/1000, or 1%. This is called as density of the dataset. This density is typical for market basket and text processing problems. Data that has a significantly higher density can require extremely large amounts of memory space to build associations.
**Dense Dataset**

Datasets with large number of attributes are known as dense datasets. In many bioinformatics applications, the datasets are dense in nature. An effective extraction of association rules on this dataset requires careful planning. One option is to start with a high minsup threshold and repeat it for lower values until desirable results are obtained. Another option is to recode some of the uninteresting attribute values to NULL, if such recoding makes the data set sparse.

In general, association rule mining algorithms work by iterative method, they work best for sparse data sets. Algorithm performance degrades exponentially with increasing number of frequent items per record. Most of the association rule mining algorithms are designed to use sparse data. However, the proposed algorithms in the thesis are designed to work with both kinds of data.

**TRIE DATA STRUCTURE**

A data structure called trie with a hash table is used in the implementation of the algorithms described in the thesis. This data structure is used effectively during the candidate generation and support counting operation. The trie data structure is a tree for storing strings in which each node corresponds to a prefix. The root is associated with the empty string. The descendants of each node represent strings that begin with the prefix stored at that node. The name of the data structure comes from the word retrieval. A common application of a trie is that of storing dictionary words, where there are usually lots of words with the same prefix. Tries represent words in a very compact way, and they allow for very fast word lookup, insertion and deletion.
The itemsets with their support value form a record. Records are stored in tables. If an operation requires a trie, then a trie is built over the itemsets. All itemsets are sorted in lexicographic order. Each node in the trie has a value, which is a 1-long item. Because of lexicographic order, the value of each child is greater than the value of its parent. An itemset is represented as a path in the trie, starting from the root node. Each node has a pointer back to its parent. Children of a node are stored in a hash table. Each terminal node (marked with double circles) has a pointer to its corresponding record. An example of trie structure is given in Figure A 1.3.

<table>
<thead>
<tr>
<th>$C_3$</th>
<th>supp</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ABC}</td>
<td>2</td>
</tr>
<tr>
<td>{ABE}</td>
<td>3</td>
</tr>
<tr>
<td>{ACE}</td>
<td>2</td>
</tr>
<tr>
<td>{BCE}</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure A 1.3 An example of Trie

A solution is proposed to reduce the trie access during the support count operation. The solution is based on the observation that the support of a large number of itemsets is asked lots of times. Since the large number of association rules share the same antecedents and/or consequents. The idea is that once the support of an itemset is retrieved from the trie, the algorithm stores the itemset-support pair temporarily in a hash. Next time, when the support of the same itemset is to be fetched, then first it is looked up in the hash. If it is there, then its stored support value is sent back. If it is not in the
hash, then the trie is accessed. The hashing operation performed during Frequent Itemset generation is depicted in Figure A 1.4.

![Diagram showing the generation of association rules, hashing, and trie structure for frequent itemsets.](figure14.png)

**Figure A 1.4 Hashing used in the generation of FI**

This data structure is used in all the implementation of the proposed algorithms except Exact-AR. In the implementation of Exact_AR, this data structure is further enhanced to perform optimized support count. It uses double hashing technique as explained in Chapter 6.2.