CHAPTER 4

THE ARCHITECTURE OF THE PROPOSED CLUSTERING BASED G-MAR MODEL

This chapter describes the overall methodology and scope of the thesis in terms of the database and system characteristics considered in the study. The choices are made such that they match those selected in the majority of the previous studies. In addition, a clustering based approach is used in the generation of factions and to construct the Coalescent Dataset for generating rules.

4.1 DATABASE AND SYSTEM CHARACTERISTICS

Conceptually, a market-basket 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 be implemented in the following four different ways (Shenoy et al 2000), which are pictorially shown in Figure 4.1.

**Item-Vector (IV):** 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 represent the item on sale, its presence or absence, respectively, in the transaction as shown in Figure 4.1(a).

**Item-list (IL):** This is similar to IV, except that each row stores an ordered list of item identifiers (IID), representing only the items actually purchased in the transaction as shown in Figure 4.1(b).
**Tid-Vector (TV):** The database is organized as a set of columns with each column storing an IID and a bit-vector of 1’s and 0’s to represent the presence or absence, respectively, of the item in the itemset of customer transactions as shown in Figure 4.1(c).

**Tid-List (TL):** This is similar to TV, except that each column stores an ordered list of only the TIDs of the transactions in which the item was purchased as shown in Figure 4.1(d).
While a mining algorithm is free to dynamically change the database layout during the mining process, it is assumed that the initial database is always provided in the horizontal item-list (IL) format.

System Characteristics: While there has been significant work in designing algorithms for the parallel mining of association rules (Agrawal et al 1996, Han et al 1997, Zaki 1997, Parthasarathy et al 2001), this study,
4.2 CLUSTERING AND PREDICTION

Prediction is used to build models that help predict future data values. The difference is that classification predicts the categorical label of a tuple, while prediction models a continuous-valued function. The process of classification begins by identifying one of the attributes of the tuples as the class label. The data set that is used to build the model is called the training data set. Because the tuples in the training data have a provided class label, this is a supervised learning method. In the second step, the model is evaluated. Usually this involves a test data set that is independent of the training data. The model is used to classify the test data and the result is compared to the class labels of the test data. If a high ratio of correctly classified tuples is obtained, the model can be used to classify new tuples with unknown class label. It is important to keep test and training data separate. Most classification methods are susceptible to over fitting, that is given enough training they learn the structure of the training data. Classification rules, decision trees or mathematical formulae can for example, represent a learned model.

4.2.1 Clustering

The job of assigning tuples to pre-defined class labels is referred to as classification, whereas the task of discovering classes to which the tuples belongs to is referred to as clustering. In general, clustering is categorized as an unsupervised learning method, since there are no labeled data to train the algorithm. Hence, in clustering, the data is grouped into clusters. A general description of a cluster is that the tuples that lie in the same cluster should be
similar, and they should be dissimilar to tuples that are not in the same cluster.
Clustering can be used as a preprocessing step for classification. It can also be
used as a tool by itself, to identify different segments in the data. The most
often used measure of evaluating a cluster is the attribute distance. It is always
preferred when using clustering that, the distance between tuples belonging to
the same cluster to be less than the distance to tuples in different clusters.
When distance is used as measure, a metric is needed. A general form is the
Minkowski metric:

\[ d(x, x^1) = \left( \sum_{k=1}^{d} |x_k - x_k^1|^q \right)^{1/q} \]  (4.1)

When \( q = 2 \), it is more known as the Euclidean metric. Setting \( q = 1 \)
gives the Manhattan or “city block” distance. So to assess the result of a
clustering algorithm, the simplest and most widely used criterion function is
the sum-of-squared-error function where \( c \) is the number of clusters, \( D_i \) is the
tuples in cluster \( i \) and \( m_i \) is the mean value of the tuples in that cluster.

For interval data, there is a whole field devoted to the discovery and
analysis of data groupings that reflect the relative distances between data
points (Srikant et al 1995).

4.2.1.1 Basic Concepts

Clustering techniques vary tremendously in how they use distances
to determine groupings and there is no universal definition of what a cluster is
or what properties it must have. Before a formal definition of clustering is
presented, some of the notations are introduced.

Let \( R = \{ A_1, A_2, \ldots, A_m \} \) be a relation schema and \( r \) be a relation over
\( R \) where \( |R| = m \) and \( |r| = n \). The convention used is that the symbols from the
end of the alphabet X, Y refers to the sets of attributes and symbols from the beginning A, B refer to single attributes.

A cluster is a set of tuples. For a specific set of attributes, X, certain restrictions are placed on the properties of these tuples when projected on X. For this reason, it can be said that a cluster is “defined on” X and the cluster is denoted as \( C_x \).

A possible quality measure on a one dimensional cluster is the range or smallest interval consisting all points or on two dimensions, the area of the smallest bounding box. However, the area does not reflect the density or coverage of points within the cluster. Hence, it is chosen to use a common measure from statistics, the average pair-wise (intra-cluster distance) or diameter of a cluster. \( \delta_x \) is used to denote a distance metric on values in the attribute set X, such as the Euclidean or Manhattan distance.

**Definition 4.1:** The diameter \( d \) on X of a set of tuples \( S = \{ t_i : 1 \leq i \leq N \} \) is the average pair wise distance between tuples projected on X.

\[
d(S[X]) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \delta_x (t_i[X],t_j[X])}{N(N-1)}
\]

In order to find clusters in finding factions, the quality of clustering is restricted using thresholds on the cluster size and the diameter.

**Definition 4.2:** A cluster \( C_x \) defined on a set of attributes X is any subset of \( r \) that satisfies the following for some density threshold \( d_0^r \) and the frequency threshold \( s_0 \).

\[
d(C_x[X]) \leq d_0^r \tag{4.3}
\]

\[
|C_x| \geq s_0 \tag{4.4}
\]
The first criterion ensures that the cluster is sufficiently dense. The second criteria ensures that the cluster is frequent, i.e., that it is supported by a sufficient number of tuples.

4.3 THE PROPOSED CLUSTERING BASED MODEL

The conventional hierarchical clustering algorithms such as single-link and complete-link suffer higher time complexity. As a result, a recent trend is to develop hybrid-clustering algorithms that exploit the advantages of both hierarchical and partitioned algorithms. Hence, a clustering algorithm Birch (Tian Zhang et al 1996) has been utilized for finding factions which are treated as a group of clusters. The idea is to use a standard clustering algorithm to identify the intervals of interest followed by the construction of Coalescent Dataset in order to check the applicability of the rules outside of the dataset. The clustering algorithm uses a single partitioning of the attributes into disjoint sets \(X_i\) over which there is a meaningful metric. Most often, each \(X_i\) an individual attribute or a small set of closely related attributes over similar domains. The clusters are created incrementally and represented by a compact summary. The summaries produced in the first phase are then used for the construction of the Coalescent Dataset approach.

4.3.1 The Clustering Methodology

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is an incremental and hierarchical clustering algorithm for large databases. The strongest point of the Birch algorithm is its support for very large databases (main memory is lower than the size of the DB).

There are two main building components in the Birch algorithm:

1. The hierarchical clustering component and
2. the main memory structure component
The idea of a hierarchical clustering is illustrated in Figure 4.2. The algorithm starts with single point clusters (every point in a database is a cluster, Clustering Feature CF shown in figure 4.2(a)). Then it groups the closest points into separate clusters (Figure 4.2(b), Figure 4.2(c)), and continues, until only one cluster remains (Figure 4.2(d)). The computation of the clusters is done with a help of distance matrix \( O(n^2) \) large and \( O(n^2) \) time.

**Figure 4.2(a) The initial dataset**

**Figure 4.2(b) The first step in Clustering process**

**Figure 4.2 (c) The second step in Clustering process**
Birch uses a main memory (of limited size) data structure called CF tree. The tree is organized in such a way that (i) the leave contain actual clusters and (ii) the size of any cluster in a leaf is not large than $R$.

4.3.2 Construction of the Clustering Feature Tree Using the BIRCH Algorithm

4.3.2.1 Clustering Algorithm

Phase 1: Scan all data and build an initial in-memory CF tree

Phase 2: Condense into desirable length by building a smaller CF tree

Phase 3: Global clustering

Phase 4: Cluster refining – this is optional, and requires more passes over the data to refine the results.

The algorithm scans the data set and inserts the incoming data instances into the CF tree one by one. The insertion of a data instance into the CF tree is carried out by traversing the CF tree top-down from the root according to an instance-cluster distance function. The data instance is inserted into the closest sub cluster under a leaf node.
If the insertion of a data instance into a sub cluster will cause the diameter of the sub cluster exceeding the threshold, then a new sub cluster is created. The creation of a new sub cluster may cause the parent of the new sub cluster, which is a leaf node in the CF tree, containing more children than the branching factor threshold.

The split of a leaf node is conducted by first identifying the pair of sub clusters under the leaf node that are separated by the largest inter-cluster distance. Then, all other sub clusters are dispatched to the two new leaf nodes based on their proximity to these two sub clusters.

Split of a leaf node may results in a non-leaf node containing more children than the pre-defined branching factor threshold. If so, then the non-leaf nodes are split recursively based on a measure of inter-cluster distance. If the root node is split, then the height of the CF tree is increased by one.

When the split of nodes terminates at a node, merge of the closest pair of child nodes is conducted as long as the latest split did not form these two nodes. A merge operation may lead to an immediate split, if the nodes formed by merge contain too many child nodes.

An example of the CF tree is illustrated in Figure 4.3. Initially, the data points in one cluster. As the data arrives, a check is made whether the size of the cluster does not exceed R (Figures 4.3(a) - 4.3(b)). If the cluster size grows too big, the cluster is split into two clusters, and the points are redistributed (Figure 4.3(c)). The points are then continuously inserted to the cluster which enlarges less (Figure 4.3(d)). At each node of the tree, the CF tree keeps information about the mean of the cluster and the mean of the sum of squares to compute the size of the clusters efficiently. The tree structure also depends on the branching parameter T, which determines the maximum number of children each node can have.
The Birch algorithm starts with a dataset, and tries to guess the size of the cluster $R$ so the tree can fit in the main memory. If the tree does not fit into the main memory, it reduces the cluster $R$ and rebuilds the tree. The process is repeated until the tree fits into main memory. The Birch algorithm can also include a number of post processing phrases to remove outliers and improve clustering.

![Figure 4.3 The Idea of CF Tree](image)

The Birch algorithm builds a dendrogram called Clustering Feature tree (CF tree) while scanning the data set. Each node in the CF tree represents a cluster of objects and is characterized by a 3-tuple: $(N, LS, SS)$, where $N$ is the number of objects in the cluster and $LS$, $SS$ are defined in the following sections.
Each non-leaf node contains a number of child nodes. A threshold called the branching factor limits the number of children that a non-leaf node can contain. Each leaf node contains a number of sub clusters that contains a group of data instances. The diameter of a sub cluster under a leaf node can not exceed a threshold.

For a leaf node, \( N \)

\[
LS = \sum_{P \in N} \bar{P}_i
\]

(4.5)

\[
SS = \sum_{P \in N} |\bar{P}_i|^2
\]

(4.6)

For a non-leaf node, which has child nodes \( N_1, N_2 \ldots N_k \),

\[
\bar{LS} = \sum_{i=1}^{k} \bar{LS} \text{ of } N_i
\]

(4.7)

\[
SS = \sum_{i=1}^{k} SS \text{ of } N_i
\]

(4.8)

### 4.4 THE PROPOSED METHOD FOR THE GENERATION OF FACTIONS USING THE CLUSTERING TECHNIQUE

This section, describes the faction generation procedure using the clustering algorithm. The clustering phase is divided into two phases, where the first phase consists of the identification of clusters and the next phase on combining clusters to generate factions. Later, the Coalescent Dataset was constructed using the factions generated.
4.4.1 Phase 1 – Identifying Clusters

The basic idea behind Birch is that clusters can be incrementally identified and refined in a single pass over the data. Each cluster is represented by a Clustering Feature (CF) that is a succinct summary of the properties of the cluster. From the CFs of two clusters, the CF of their union and a number of distance metrics can be derived. Hence, clusters can be obtained and new points added to clusters using only the CFs.

A Clustering Feature (CF) contains the number of tuples, the linear sum of the tuples and the square sum of the tuples of a cluster. For \( C_x = \{t_1, \ldots, t_N \} \), the CF is defined as:

\[
CF(C_x) = \left( N, \sum_{i=1}^{N} t_i [X], \sum_{i=1}^{N} t_i [X]^2 \right)
\]  \( (4.9) \)

A height-balanced tree of CF vectors guides the clustering process. An internal node contains a list of (CF, pointer) pairs where each CF summarizes all data points in the descendents. Leaf nodes contain lists of CFs. The CF-tree is built incrementally by inserting new data points individually. Each data point is inserted by locating a closest clustering feature at each level of the tree and following the corresponding pointer recursively. At each level, the closest clustering feature is updated to reflect the insertion of the new point. At the leaf, the point is added to the closest cluster, if the diameter of the augmented cluster does not exceed a threshold diameter. Otherwise, a new cluster is created. When leaf nodes are full, they are split and the CF-tree that guides insertion is adjusted, more in a way a B+ tree is adjusted in response to the insertion of a new tuple.

For each \( X_i \), an initial diameter threshold distance metric and \( d_0^X \) is selected. As data values are inserted into the CF tree, if a clusters diameter
exceeds the threshold, it is split. The split may increase the size of the tree. If the memory is full, the tree is reduced by increasing the diameter threshold and rebuilding the tree. The rebuilding is done by re-inserting leaf CF nodes into the tree. Hence, the data or the portion of the data that has already been scanned does not need to be rescanned (Zhang et al 1996). With the higher threshold, some clusters are likely to be merged reducing the space required for the tree.

As the CF-tree is being built, small clusters may be paged out to disk. Since this is done before all data has been scanned, clusters may be wrongly categorized into the complete tree to ensure that they are indeed outliers.

4.4.2 Phase 2 – Combining Clusters

All clusters found in Phase 1 will satisfy the density threshold. In Phase 2, all clusters that also satisfy the frequency threshold (that is, clusters with significant support) are used. These clusters form the set of attributes that are considered as factions. If for some $X_i$ there are no frequent clusters, $X_i$ is omitted from consideration in Phase 2.

Clusters are combined based on the frequency with which they appear together in the dataset. A cluster has been defined to be a set of tuples, but in Birch, summaries are discovered. Hence, a way is required to determine the cluster to which each point belongs. For each point, a centroid closest to the point can be found and define the tuple to be in the cluster represented by the centroid. Due to local and incremental nature of Birch, this cluster may not be the same cluster to which the tuple was assigned when it was originally inserted into the CF-tree.
The distance between two clusters can be defined using any of a number of standard statistical measures, including the average inter-cluster distance or the centroid Manhattan distance. The distance measures are therefore applied to the clusters projected on specific attributes.

Having constructed the CF tree, the clustering algorithm then invokes a hierarchical clustering algorithm to cluster the sub clusters under each leaf node. The algorithm takes as input the mixed type of data and the hybrid clustering mechanism is applied. A set of data instances is said to have the numerical data type, if the data instances are described by attributes that take numerical values. In case of numerical data, the following distance functions can be used and a distance function must satisfy the following requirements:

- \( d(x, y) \geq 0 \) \hspace{4cm} (4.10)
- \( d(x, x) = 0 \) \hspace{4cm} (4.11)
- \( d(x, y) = d(y, x) \) \hspace{4cm} (4.12)
- \( d(x, y) \leq d(x, z) + d(y, z) \) \hspace{4cm} (4.13)

The Euclidean distance:

\[
\sum_{i=1}^{n} \sqrt{(x_i - y_i)^2}
\]  

(4.14)

The Manhattan distance

\[
\sum_{i=1}^{n} |x_i - y_i|
\]  

(4.15)

Weighted Distance

\[
\sum_{i=1}^{n} w_i \sqrt{(x_i - y_i)^2}
\]  

(4.16)
A set of data instances is said to have the categorical data type, if the data instances are described by attributes that take categorical values. A numerical attribute can be converted to a categorical attribute by specifying a number of ranges.

4.5 THE OVERALL ARCHITECTURE OF THE PROPOSED MODEL

![Figure 4.4 The Model Architecture](image-url)
4.6 PERFORMANCE

The comparison between the correct set of large item sets and the estimated set of large itemsets is based on two factors:

1. The actual items in the itemsets and
2. The support values associated with the matching itemsets

The Figure 4.5 shows an example for generating factions using the clustering based approach proposed for the model.

Figure 4.5 $C_X$ is a cluster on $X$ and $C_Y$ a cluster on $Y$
Figure 4.6 Clusters over Age, Dependents and Income

The three clusters as shown in the figure 4.6 described by their bounding intervals results in the generation of three factions namely the age, which lies between the interval [41-47], the number of dependents [2-5] and income ranging between [10000-14000] respectively.

4.6.1 Efficiency and Performance of the Proposed G-MAR Model

In order to assess the performance of the proposed G-MAR model, VIPER (Shenoy et al 2000) and FP-growth (Han et al 2000) are chosen for comparison among the suite of Association Rule Mining algorithms. For completeness and as a reference point, the classical Apriori (Agrawal et al 1994) is also included in the evaluation suite.

The experiments cover a range of databases and mining workloads and the only difference with the existing algorithms is that the database sizes
were considered significantly larger than the available main memory. The performance metric in all the experiments is the total execution time taken by the mining operation.

The databases used in the experiments were synthetically generated using the technique described in (Agrawal et al. 1994) and attempt to mimic the customer purchase behaviour seen in retailing environments. The parameters used in the synthetic generator and their default values are described in the Table 4.1. In particular, the databases with parameters T10.I4, T20.I2 and T40.I8 with 10 million tuples in each of them were considered.

Table 4.1 Parameters used for Data Generation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of Items</td>
<td>1000</td>
</tr>
<tr>
<td>T</td>
<td>Mean Transaction Length</td>
<td>10,20,40</td>
</tr>
<tr>
<td>L</td>
<td>Number of potentially frequent itemsets</td>
<td>2000</td>
</tr>
<tr>
<td>I</td>
<td>Mean length of potentially frequent itemsets</td>
<td>4,8,12</td>
</tr>
<tr>
<td>D</td>
<td>Number of transactions in the database</td>
<td>10M</td>
</tr>
</tbody>
</table>

The rule support threshold values are set to as low as was feasible with the available main memory. At these low support values, the number of frequent itemsets exceeded twenty five thousand. Beyond this, it was felt that the number of rules generated would be enormous and the purpose of mining to find interesting patterns would not be served. In particular, the rule support threshold values for the T10.I4, T20.I2 and T40.I8 databases were set to the ranges (0.1% - 2%), (0.4%-2%) and (1.15% - 5%) respectively.
The experiments were conducted on a 700MHz Pentium III workstation running Red Hat Linux 6.2, configured with a 512MB main memory and a local 18GB SCSI 10000 rpm disk. For the T10.I4, T20.I2 and T40.I8 databases, the associated database sizes were approximately 500 MB, 900MB and 1.7GB respectively. All the algorithms in the evaluation suite were written in C++. The basic version of FP-growth algorithm was implemented wherein it was assumed that the entire FP-tree data structure fits in main memory. Finally, the partition size in the proposed G-MAR model was fixed to 20K tuples.

**4.6.2 Experimental Results for the Current Mining Algorithms**

Two experiments were conducted to evaluate the performance of current mining algorithms in generating rules with respect to the proposed G-MAR model. The first experiment was run on large (10M tuples) databases, while the second experiment was run on small (100K tuples) databases.

**Experiment 1: Performance of current algorithms**

In the first experiment, the performance of Apriori, Viper and the G-MAR model were evaluated for the T10.I4, T20.I2 and T40.I8 databases each containing 10M transactions and the results are shown in figures (Figures 4.7 a-c). The x-axis in these graphs represents the support threshold values while the y-axis represents the response times of the algorithms being evaluated.

In these graphs, it is seen that the response times of all algorithms increase exponentially as the support threshold is reduced. This is only to be expected since the number of itemsets in the output increases exponentially with the decrease in support.
Figure 4.7(a) Performances of Algorithms on T10.I4.D10M datasets

Figure 4.7(b) Performances of Algorithms on T20.I12.D10M datasets
It is also seen that there is a considerable gap in the performance of both Apriori and Viper with respect to the model. For example, in Figure 4.7(a), at a support threshold of 0.1%, the response time of Viper is more than 6 times that of the G-MAR model whereas the response time of Apriori is more than 26 times. In this experiment, the performance of FP-growth could not be evaluated because it did not complete in any of the runs on large databases due to its heavy and database size dependent utilization of main memory. The reason for this is that FP-growth stores the database itself in a condensed representation in a data structure called FP-tree. In (Han et al 2000), the authors briefly discuss the issue of constructing disk resident FP-trees.

**Experiment 2: Small Databases**

Since, as mentioned above, it was not possible to evaluate the performance of FP-growth on large databases due to its heavy utilization of main memory, the performance of FP-growth and other current algorithms...
were evaluated on small databases consisting of 100K transactions. The results of this experiment are shown in figures 4.8 (a-c) which corresponds to T10.I4, T20.I12 and T40.I8 databases respectively.

In these graphs, it can be seen that there continues to be a considerable gap in the performance of current mining algorithms with respect to G-MAR. For example, for the T40.I8 database shown in Figure 4.8(c), the response time of FP-growth is more than 8 times that of G-MAR for the entire support threshold range. Second, although FP-growth does well at low supports, its performance is worse than Apriori for high supports. These results are inconsistent with those shown in (Han et al 2000) where it was shown that FP-growth consistently performs better than Apriori for the entire support range. While this could possibly be due to the differences between the respective implementations of FP-growth and / or Apriori, it is felt that there are logical reasons for this behaviour as explained below.

At high supports Apriori typically performs only 2 passes over the data since with these supports there are usually no frequent itemsets of length greater than two. In these cases, the first pass of Apriori is identical to the preprocessing pass in FP-growth in which all frequent singletons are obtained. The second pass of Apriori is quite efficient since the counts of the candidate 2-itemsets are maintained in a 2-dimensional lookup array. FP-growth, on the other hand, constructs a FP-tree during the second pass. The FP-tree is updated on a tuple-by-tuple basis. Each node in the FP-tree contains an item-name field. A critical operation during FP-tree construction is to find the child of a node given a key item-name. If these keys are stored in look-up arrays, the memory requirements of FP-tree would be still worse. The alternative is to use an indexing data structure such as a red-black tree or a skip-list that requires O(log n) time to perform the find operation, but this would make the FP-tree construction slow. Even assuming that the cost FP-tree construction is
equal to the second pass of Apriori, FP-growth still needs to mine the FP-tree. Hence FP-growth loses out at high supports.

Figure 4.8 (a) Performances of Algorithms on T10.I4.D100K datasets

Figure 4.8(b) Performances of Algorithms on T20.I12.D100K datasets
Figure 4.8(c) Performances of Algorithms on T40.I8.D100K datasets

The guiding principle in the design of G-MAR is that an attempt is made to extend the applicability of Association rules without modifying the core process. This is marked contrast to earlier approaches.

The construction of factions is important in the construction of the Coalescent Dataset. The factions should keep as much demographic information as possible, that is, the people in each faction should have similar behaviour. In G-MAR, the database is conceptually partitioned into ‘n’ disjoint partitions $P_1, P_2 \ldots P_n$. At most 2 passes are made over the database. In the first pass, a set of candidate itemsets are formed that is guaranteed to be a superset of the set of frequent itemsets. During the first pass, the counts of candidates in G are determined over each partition by maintaining the candidates in a data structure. Generation and removal of candidates is done simultaneously while computing counts.
4.6.3 Memory Utilization in G-MAR

In the design and implementation of G-MAR, speed has been opted in most decisions that involve a space-speed trade-off. Therefore, the main memory utilization in G-MAR is certainly more as compared to algorithms such as Apriori. However, in the following discussion, it is shown that the memory usage of G-MAR is well within the reaches of the current machine configurations, which is also experimentally confirmed.

The main memory consumption of G-MAR comes from the following sources:

1. The arrays for storing counters of itemsets and pairs respectively
2. The structure for storing counters of large itemsets including the tidlists of those itemsets
3. The current partition
4. The number of factions generated
5. The size of the Coalescent Dataset

Since the environment that is considered is the one where the pattern lengths are small, the number of candidates will typically be comparable to or well within the available memory. Xiao et al (1999) discusses alternative approaches when this assumption does not hold. The main memory consumed by the current partition is small compared to the space occupied by the tidlist of itemsets. For example, if each transaction occupies 1 KB, a partition size 20K would require only 20K of memory.
Hence, the total memory consumption of the model is widely acceptable on current machines.

### 4.6.3.1 Experimental Results for the G-MAR model

The performance of G-MAR with respect to Intra-Inter transaction mining was studied on a variety of databases and support characteristics. The experimental result for the same performance model discussed above is reported. Since, Apriori, FP-growth and Viper have already been compared with the proposed G-MAR model, the experimental results were compared with the Intra-Inter transaction mining algorithm.

**Experiment 3: Performance of G-MAR**

![Figure 4.9(a) Performance on Synthetic Datasets (T10.I4.D10M)](image)
In this experiment, the response time performance of the G-MAR was evaluated with Intra-Inter transaction mining algorithms for the T10.I4,
T20.I12 and T40.I8 databases each containing 10M transactions and these results were shown in Figures 4.9(a-c).

In these graphs, it is first seen that G-MAR’s performance is close to that of Inter-Inter transaction mining for high supports. This is because of the following reasons: The density of the frequent itemset distribution is sparse at high supports resulting in only a few frequent itemsets with supports close to minsup. Hence, frequent itemsets are likely to be locally frequent within most partitions. Even if they are not locally frequent in a few partitions, it is very likely they are still the frequent over these partitions. Hence, their counters are updated even over these partitions. Therefore, the complete counts of most candidates would be available at the end of the first pass resulting in a second pass. Hence, it is expected that the performance of G-MAR will be close to that of Inter–Inter Transaction mining at high supports.

Since the frequent itemset distribution becomes dense at low supports, the above argument does not hold in this support region. Hence, it is seen that G-MAR’s performance relative to Intra-inter transaction mining decreases at low supports. This is shown in the Table 4.2, where the ratios of performance of G-MAR to that of Intra-Inter transaction mining for the lowest support values were considered for each of the databases.

<table>
<thead>
<tr>
<th>Database</th>
<th>Minsup(%)</th>
<th>G-MAR(seconds)</th>
<th>FITI(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T10.I4.D10M</td>
<td>0.1</td>
<td>372.45</td>
<td>225.98</td>
</tr>
<tr>
<td>T20.I12.D10M</td>
<td>0.4</td>
<td>1151.31</td>
<td>811.02</td>
</tr>
<tr>
<td>T40.I8.D10M</td>
<td>1.15</td>
<td>2701.44</td>
<td>2247.20</td>
</tr>
</tbody>
</table>

Table 4.2 Worst-case Efficiency of G-MAR with respect to FITI
The previous experiments were conducted with the total number of items in the Coalescent Dataset, N, being set to 1K. In this experiment, the value was set to 20K items for the T10.I4 database where this environment represents an extremely stressful situation for G-MAR with regard to memory utilization due to the very large items. Figure 4.10 shows the memory utilization of G-MAR as a function of support for the N= 1K and N=20K cases. It is seen that the main memory utilization of G-MAR scales well with the number of items. For example, at the 0.1% support threshold, the memory consumption of G-MAR for N=1K items was 104 MB while for N=20K items, it was 143 MB. The reason for this is that the main memory utilization of G-MAR does not depend directly on the number of items, but only on the size of the output.

Figure 4.10 Memory Utilization in G-MAR
Experiment 4: Real Datasets

Performance of G-MAR on Real Datasets

The performance of the model was done on the two datasets – BMS-Webview-1, a click stream data from Blue Martini software (Zheng
et al 2001) and EachMovie, a movie database from Compaq Equipment Corporation (com97), which has been transformed to the format of market basket data. The resulting databases had 59,602 and 61,202 transactions respectively with 870 and 1648 distinct items.

The support rule threshold values for the BMS-WebView-1 and EachMovie databases were set to the ranges (0.06%-0.1%) and (3%-10%), respectively. The results of these experiments are shown in Figures 4.11(a) and 4.11(b). It is seen in these graphs that the performance of G-MAR continues to be within twice that of FITI. The ratio of G-MAR’s performance at the lowest support value of 0.06% for the BMS-WebView-1 database was 1.83 whereas at the lowest support value of 3% for EachMovie databases it was 1.73.

4.7 CONCLUSION

The proposed model for the generation of association rules for a new setting has been compared with the existing approaches and has been found that the model performs quite well in prediction of rules for a new setting. A clustering based approach is adopted for the generation of rules which thereby is used for comparing the performance of the proposed model with the other algorithms. The memory utilization of the model was also discussed and the experimental results have shown that the model better performs in identifying rules for a new situation.