Clustering Association Rules*

Brian Lent† Arun Swami§ Jennifer Widom†

† Department of Computer Science
Stanford University
Stanford, CA 94305
`http://www-db.stanford.edu`
{`lent, widom`}`@cs.stanford.edu`

§ Internet Profiles Corporation
San Francisco, CA
`arun@cs.stanford.edu`

Abstract

We consider the problem of clustering two-dimensional association rules in large databases. We present a geometric-based algorithm, BitOp, for performing the clustering, embedded within an association rule clustering system, ARCS. Association rule clustering is useful when the user desires to *segment* the data. We measure the quality of the segmentation generated by ARCS using the Minimum Description Length (MDL) principle of encoding the clusters on several databases including noise and errors. Scale-up experiments show that ARCS, using the BitOp algorithm, scales linearly with the amount of data.

1 Introduction

Data mining, or the efficient discovery of interesting patterns from large collections of data, has been recognized as an important area of database research. The most commonly sought patterns are *association rules* as introduced in [AIS93b]. Intuitively, an association rule identifies a frequently occuring pattern of information in a database. Consider a supermarket database where the set of items purchased by a single customer at the check-stand is recorded as a *transaction*. The supermarket owners may be interested in finding "associations" among the items purchased together at the check-stand. An example of such an association is that if a customer buys bread and butter then it is likely he will buy milk. Given a set of transactions, where each transaction is a set of items, an association rule is an expression $\mathcal{X} \Longrightarrow \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are sets of items. The meaning of this rule is: transactions that contain the items in $\mathcal{X}$ also tend to contain the items in $\mathcal{Y}$.

Generally, association rules are used in conjunction with transaction (or *basket*) type data, but their use is not limited to this domain. When dealing with customer demographic data, for example, the database

*This research was initiated by the first two authors while at Silicon Graphics Computer Systems, Inc. Continuing support for the first author has been provided by a graduate fellowship of the Department of Defense Office of Naval Research.

schema defines a fixed set of attributes for each record, and each customer is represented by one record. Each record contains a value for each attribute, i.e., (attribute = value). By replacing the sets of items in the traditional definition of association rules with conjunctions of (attribute = value) equalities, we can generalize the above definition of association rules to include this type of non-transactional data. For example, (age = 40) $\wedge$ (Salary = \$50,000) $\Longrightarrow$ (own_home = yes).

When mining association rules from this type of non-transactional data, we may find hundreds or thousands of rules [KMRT94] corresponding to specific attribute values. We therefore introduce a *clustered association rule* as a rule that is formed by combining similar, "adjacent" association rules to form a few general rules. Instead of a set of (attribute = value) equalities, for clustered rules we have a set of value ranges using inequalities. For example, the clustered rule $(40 \leq age < 42) \Longrightarrow (own\_home = yes)$ could be formed from the two association rules $(age = 40) \Longrightarrow (own\_home = yes)$ and $(age = 41) \Longrightarrow (own\_home = yes)$. The problem we consider is how to efficiently mine clustered association rules from large databases, using an association rule mining algorithm as one step in the overall process.

In practice it is very important that rules mined from a given database are understandable and useful to the user. Clustered association rules are helpful in reducing the large number of association rules that are typically computed by existing algorithms, thereby rendering the clustered rules much easier to interpret and visualize. A practical use of these clusters is to perform *segmentation* on large customer-oriented databases. As an example, consider a marketing company that sends a direct mail catalog to its current customer base promoting its products. Orders are taken for each customer and placed in a transactional database, recording a list of those items purchased along with some demographic information about the purchaser. At some point the company decides to expand its current customer base. By grouping its existing customers by total sales, for instance, into groups of "excellent", "above average", and "average" profitability, the company could use clustered association rules on the attributes of the demographic database to segment its customers. The company can then use this segmentation in selecting the new customers it should target for future mailings who are most likely to respond. Hence, we would like a segmentation that defines a specific criterion (e.g., customers rated "excellent") as a function of the other attribute values. This corresponds to considering clustered association rules of the form

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Longrightarrow G$$

where G is one of "excellent", "above average", or "average", and the A_i are attribute ranges such as $(40 \leq age < 42)$.

In this paper, we consider association rule clustering in the two-dimensional space, where each axis represents one attribute from the database used on the left-hand side (LHS) of a rule. (In Section 5 we discuss possibilities for handling higher dimensionality data.) Before processing by our algorithms, the two

LHS attributes A_x and A_y are chosen by the user. Statistical techniques for identifying the most influential attributes for a given dataset, such as *factor analysis* [Kru78] and *principal component analysis* [Law71], have been well-studied and could also be used. The user also selects a value from the third attribute as a criterion to be used for segmentation. We use an adjacency definition that ensures that the clusters are rectangular regions of the data. This helps with readability of the final clustered association rules.

Clustering in general is a difficult problem and as a research topic has been studied for quite some time [Rom84, Spa80]. Even under the two-dimensional attribute assumptions we make, the problem of identifying the fewest clusters in a two-dimensional grid is a specific instance of the *k-decision set-covering problem*, which has been shown to be NP-complete [CLR90].

Our clustering approach begins by taking source data in tuple form and partitioning those attributes that take values from a continuous domain. We then perform a single pass over the data using an association rule engine to derive a set of association rules. Next, we cluster all those two-attribute association rules where the right-hand side of the rules satisfies our segmentation criteria. Our approach to the clustering problem is heuristic, based on the geometric properties of a two-dimensional grid, and produces an efficient linear time approximation to an optimal solution. This clustering produces the desired segmentation. We test the segmentation for accuracy, and if necessary modify certain system parameters to produce a better segmentation and repeat the process.

1.1 Related Work

The problem of taking sets of attribute-value pairs, such as (Age=x, Salary=y), as points in two-dimensional space and then finding optimal regions, with respect to some specified criteria, within this space was introduced in [FMMT96]. The authors in [FMMT96] considered two types of regions, rectangles and *connected x-monotone*, but focus on the latter. We focus on rectangular regions because they usually are more understandable. Unlike [FMMT96] we do not require the user to choose any of the parameters necessary for generating association rules, but instead provide a fully automated system for identifying clustered association rules.

In [SA96] the authors consider the problem of mining quantitative attributes, in contrast to previous algorithms designed for transaction-type items. The authors in [SA96] partition attributes using *equi-depth* bins (where each bin contains roughly the same number of tuples), and introduce a new measure to help the user to decide how many partitions there should be for each attribute. They address issues of combining adjacent intervals and use a "greater-than-expected-value" interest measure for the rules. One component of our work also requires partitioning quantitative attributes, but differs in that we are using the partitioned quantitative attributes as a means to form clusters that ultimately define a segmentation of the data. Further, our system is fully automated and does not require any user-specified thresholds as does the work in [SA96].

In addition, we have provided an alternative definition of an "interesting" association rule that gives an intuitive means of segmenting the data. Finally, whereas [SA96] uses user-specified parameters and a partial completeness measure to determine attribute partitioning, we use geometric properties of the attribute space to form clusters.

A related problem is *classification*, where data is categorized into disjoint groups based upon common characteristics of its attributes. Techniques for classification include instance-based classifiers, decision trees, artificial neural networks, genetic algorithms, and various statistical techniques [Qui93]. Classification has been studied primarily in the AI community, and the computational complexity of these algorithms generally inhibits the performance efficiency necessary when mining large databases [Lub89]; furthermore, the algorithms do not scale well with increasing database size. In the database community, the work in [AGI+92, MAR96, SAM96] has focused on designing efficient classification algorithms for large databases. The goal of classification is to compute a *predictive model* for each of the groups. The emphasis is to produce the most accurate model that can predict the group to which an unseen instance belongs. An understanding of the model that the classification algorithm computes is usually of secondary importance. By contrast, in our work we want to segment the space of attribute values, based on some criterion grouping attribute, into manageable segments that are understandable to the user.

1.2 Paper Organization

In Section 2 we present formal definitions and an overview of the work. Section 3 describes our Association Rule Clustering System, ARCS, and describes the BitOp algorithm for clustering. We also discuss a preprocessing step to smooth the grid, and dynamic pruning that removes uninteresting clusters. Statistical measures for setting the initial threshold values are also described in Section 3. In Section 4, we present experimental results on synthetic data and show the scalability of ARCS for large databases. Using a classifier as an alternative way of segmenting the data, we compared a well-known classifier to our approach that instead uses clustered association rules for segmentation. Section 5 concludes and presents our ideas for future work.

2 Preliminaries

In this section we present formal definitions and terminology for the problem of mining clustered association rules, and then we give an overview of our approach.

2.1 Terminology

An attribute can be either *categorical* or *non-categorical*. Categorical attributes are those that have a finite number of possible values with no ordering amongst themselves. Examples of categorical attributes include "zip code", "hair color", "make of car", etc. Non-categorical attributes, which we will call *quantitative* attributes, do have an implicit ordering and can assume continuous values usually within a specified range. Examples of quantitative attributes include "salary", "age", "interest rate", etc.

Let $\mathcal{D}$ be a database of tuples where each tuple is a set of attribute values, called *items*, of the form ($\text{attribute}_i = \text{value}_i$). An *association rule* is an expression of the form $\mathcal{X} \Longrightarrow \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are sets of (attribute=value) items such that no attribute appears more than once in $\mathcal{X} \cup \mathcal{Y}$. We will refer to $\mathcal{X}$ as the left-hand side (LHS) of the rule, and $\mathcal{Y}$ as the right-hand side (RHS) of the rule.

Two common numeric measures assigned to each association rule are *support* and *confidence*. Support quantifies how often the items in $\mathcal{X}$ and $\mathcal{Y}$ occur together in the same tuple as a fraction of the total number of tuples, or $\frac{|\mathcal{X}\mathcal{Y}|}{|D|}$ where $|D|$ denotes the total number of tuples. Confidence quantifies how often $\mathcal{X}$ and $\mathcal{Y}$ occur together as a fraction of the number of tuples in which $\mathcal{X}$ occurs, or $\frac{|\mathcal{X}\mathcal{Y}|}{|\mathcal{X}|}$. When generating a set of association rules using any of the known algorithms, the user must specify minimum threshold values for both support and confidence.

The following is an example of an association rule:

$$(\text{Age} = 32) \wedge (\text{Salary} = \$38,500) \Longrightarrow (\text{Customer\_rating} = \text{good})$$

Because quantitative attributes will typically assume a wide range of values from their respective domains, we partition these attributes into intervals, called *bins*. In this paper, we consider only *equi-width* bins (the interval size of each bin is the same). Other choices are possible, such as *equi-depth* bins (where each bin contains roughly the same number of tuples), or *homogeneity-based* bins (each bin is sized so that the tuples in the bin are uniformly distributed) [MD88, WKW94]. Our approach can easily be extended to handle these cases as well. When a quantitative attribute is partitioned into bins, the bins are mapped to consecutive integers; when attribute values are mapped to bins their value is replaced with the corresponding integer for that bin. For categorical attributes we also map the attribute values to a set of consecutive integers and use these integers in place of the categorical values. Since this mapping happens prior to running an association rule mining algorithm, the binning process is transparent to the association rule engine. This binning and mapping approach is also used in [SA96] and [PS91].

Clustering, as defined here, is the combination of adjacent attributes values, or adjacent bins of attribute values. For example, clustering (Age=40) and (Age=41) results in ($40 \leq \text{Age} < 42$). A *clustered association rule* is an expression of the form $\mathcal{X}_C \Longrightarrow \mathcal{Y}_C$. $\mathcal{X}_C$ and $\mathcal{Y}_C$ are items of the form (Attribute = value) or ($\text{bin}_i \leq \text{Attribute} < \text{bin}_{i+1}$), where bin_i denotes the lower bound for values in the i^{th} bin.

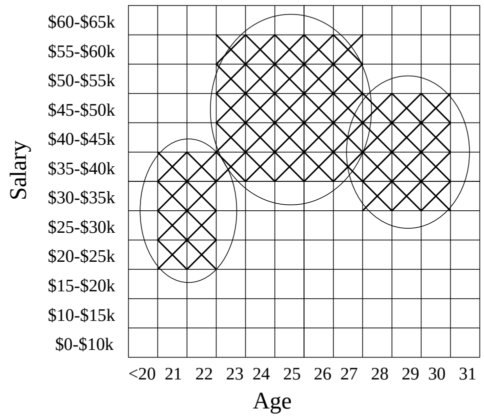


Figure 1: Sample grid with clustered association rules.

2.2 Overview of Approach

In this paper, we consider the problem of clustering association rules of the form $A \wedge B \Longrightarrow C$ where the LHS attributes (A and B) are quantitative and the RHS attribute (C) is categorical.[1] We define a *segmentation* as the collection of all the clustered association rules for a specific value C of the criterion attribute.

Given a set of two-attribute association rules over binned data, we form a two-dimensional grid where each axis corresponds to one of the LHS attributes. On this grid we will plot, for a specific value of the RHS attribute, all of the corresponding association rules. An example of such a grid is shown in Figure 1. Our goal is to find the fewest number of clusters, shown as circles in the figure, that cover the association rules within this grid. These clusters represent our clustered association rules and define the segmentation.

The algorithm we introduce to cluster association rules using a 2D grid solves only a part of the overall problem. Recall that in order to mine a collection of association rules we must first define minimum thresholds for both support and confidence. Further, for the quantitative attributes we must determine the number of bins over which to partition the attribute domain. Finding the values for each of these parameters that gives us the best segmentation is a difficult combinatorial optimization problem. We present a unifying framework that uses heuristics for searching this space efficiently and has very good results. Figure 2 shows a high-level view of our entire system to compute the clustered association rules, which we now describe.

While the source data is read, the attribute values are partitioned (by the *binner*) as described earlier. While the number of bins can be changed by the user, doing so restarts the system. The association rule engine is a special-purpose algorithm that operates on the binned data. The minimum support is used along with the minimum confidence to generate the association rules.

Once the association rules are discovered for a particular level of support and confidence, we then form a grid of only those rules that give us information about the group (RHS) we are segmenting. We apply

[1] We consider only quantitative LHS attributes because the lack of ordering in categorical attributes introduces additional complexity. We are currently extending our work to handle categorical LHS attributes.

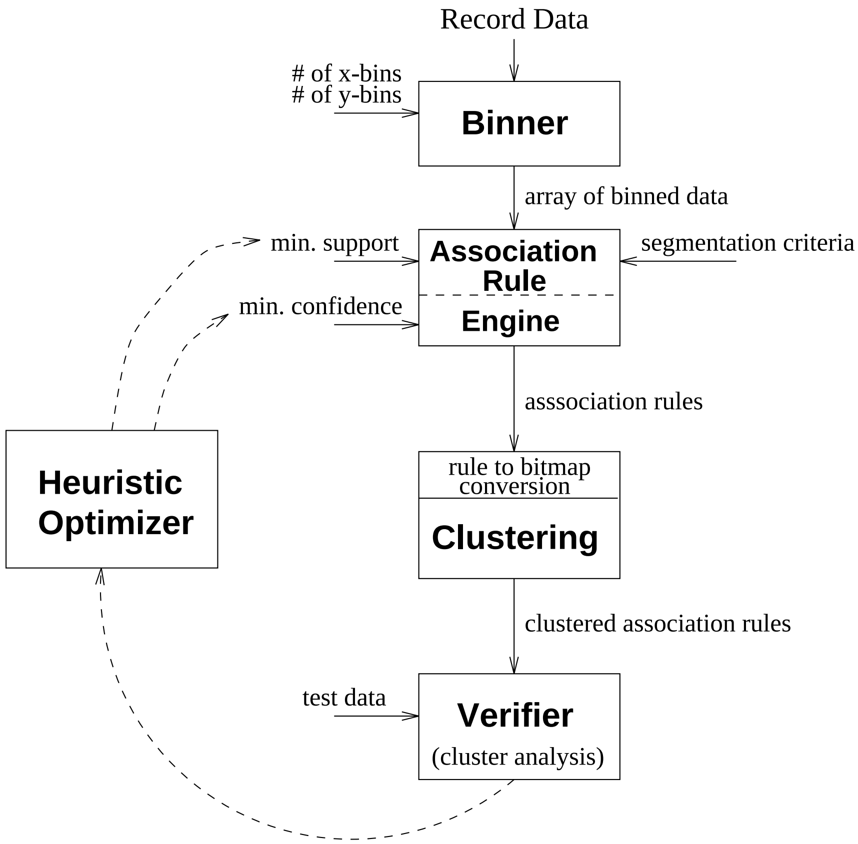


Figure 2: Architecture of the Association Rule Clustering System

our BitOp algorithm (Section 3.3) to this grid, along with certain other techniques described in Sections 3.4 and 3.5, to form clusters of adjacent association rules in the grid. These clustered association rules are then tested for their accuracy (by the *verifier*) against a sample of tuples from the source database. The accuracy is supplied to a *heuristic optimizer* (Section 3.7) that adjusts the minimum support threshold and/or the minimum confidence threshold, and restarts the mining process at the association rule engine. These heuristic adjustments continue until the verifier detects no significant improvement in the resulting clustered association rules, or the verifier determines that the budgeted time has expired.

3 Association Rule Clustering System (ARCS)

The ARCS framework was shown in Figure 2. We now detail components of the system.

3.1 Binning Data

The binner reads in tuples from the database and replaces the tuples' attribute values with their corresponding bin number, as previously described. We first determine the bin numbers for each of the two (LHS) attributes, A_x and A_y. Using the corresponding bin numbers, bin_x and bin_y, we index into a 2D array where, for each

```
Input:
   I1:  Tuples of the form (A_x, A_y, A_seg), where A_seg is the segmentation criterion attribute
        to appear on the RHS of the association rules.
   I2:  n_x, the number x-bins.
   I3:  n_y, the number y-bins.
Output:
   O1:  The structure BinArray[x,y,seg], containing aggregated tuple counts for the binned data.

Procedure Binner()
   h_x = dom(A_x)/n_x; /* width of the x-bins */
   h_y = dom(A_y)/n_y; /* width of the y-bins */

   /* Perform the binning of the attributes */
   Repeat get next tuple (a_x, a_y, a_seg)
      bin_x = a_x/h_x;
      bin_y = a_y/h_y;
      BinArray[bin_x, bin_y, a_seg] = BinArray[bin_x, bin_y, a_seg] + 1;
      BinArray[bin_x, bin_y,TOTAL] = BinArray[bin_x, bin_y,TOTAL] + 1;
   Until no remaining source data
```

Figure 3: The binning algorithm

bin_x,bin_y pair, we maintain the number of bin_x,bin_y tuples having each possible RHS attribute value, as well as the total number of bin_x,bin_y tuples. The size of the 2D array is $n_x * n_y * (n_{seg}+1)$ where n_x is the number of x-bins, n_y is the number of y-bins, and n_{seg} is the cardinality of the (RHS) segmentation attribute. In our system we assume this array can fit in main memory.

Although we are typically interested in only one value of the segmentation criteria at a time (e.g., "customer_rating = excellent"), by maintaining this data structure in memory we can compute an entirely new segmentation for a different value of the segmentation criteria without the need to re-bin the original data. If memory space is at a premium, however, we can set $n_{seg} = 1$ and maintain tuple counts for only the one value of the segmentation criteria we are interested in. Figure 3 gives the algorithm for the binning component.

3.2 The Association Rule Engine

While it is possible to use any of the existing association rule mining algorithms to mine the binned data, we describe a more efficient algorithm for the special case of mining two-dimensional association rules using the data structure constructed by our binning process. Deriving association rules from the binned data in the BinArray is straightforward. Let G_k be our RHS criterion attribute. Every cell in the BinArray can be represented by an association rule whose LHS values are the two bins that define the BinArray cell, and whose RHS value is G_k:

$$(X = i) \wedge (Y = j) \Longrightarrow G_k$$

```
Input:
   I1:  The BinArray computed from the binning component.
   I2:  The value G_k we are using as the criterion for segmentation.
   I3:  The min_support (%) threshold.
   I4:  The min_confidence (%) threshold.
   I5:  N, the total number of tuples in the source data.
   I6:  n_x, the number of x-bins.
   I7:  n_y, the number of y-bins.
Output:
   O1:  A set of pairs of bin numbers, (i,j), representing association rules of the form
        (X = i) ∧ (Y = j) ⟹ G_k.

Procedure GenAssocationRules()
   min_support_count = N * min_support;
   /* Perform association rule generation from the binned data */
   for(i=1; i≤ n_x; i++) do begin
      for(j=1; j≤ n_y; j++) do begin
         if ((BinArray[i,j,G_k] ≥ min_support_count) and
             (BinArray[i,j,G_k]/BinArray[i,j,TOTAL] ≥ min_confidence))
            Output (i,j)
      end-for
   end-for
```

Figure 4: The association rule mining algorithm

where $(X = i)$ represents the range $(bin_x^i \leq X < bin_x^{i+1})$, and $(Y = j)$ represents the range $(bin_y^j \leq Y < bin_y^{j+1})$, and bin_x^i and bin_y^j are the lower bounds of the i^{th} x-attribute bin and the j^{th} y-attribute bin, respectively. The support for this rule is $\frac{|(i,j,G_k)|}{N}$ and the confidence is $\frac{|(i,j,G_k)|}{|(i,j)|}$, where N is the total number of tuples in the source data, $|(i,j)|$ is the total number of tuples mapped into the BinArray at location (i,j), and $|(i,j,G_k)|$ is the number of tuples mapped into the BinArray at location (i,j) with criterion attribute value G_k. To derive all the association rules for a given support and confidence threshold we need only check each of the occupied cells in the BinArray to see if the above conditions hold. If the thresholds are met, we output the pair (i,j) corresponding to the association rule on binned data as shown above. The (i,j) pairs are then used to create a bitmap grid that is used by the BitOp algorithm, described in the following section, to locate clusters of association rules.

Our algorithm, shown in Figure 4, requires only a single pass through the data, in contrast to using some existing algorithms that may need to make several passes to find all association rules. It is also very important to note that by maintaining the BinArray data structure, we can apply different support or confidence thresholds without reexamining the data, making the "re-mining" process dramatically faster than with previous algorithms. In our system, changing thresholds is nearly instantaneous.

3.3 Clustering

We begin by presenting a very simple example of the clustering problem to illustrate the idea. Consider the following four association rules where the RHS attribute "Group_label" has value "A":

$$
\begin{array}{c}
(\text{Age} = 40) \wedge (\text{Salary} = \$42,350) \Longrightarrow (\text{Group\_label} = A) \\
(\text{Age} = 41) \wedge (\text{Salary} = \$57,000) \Longrightarrow (\text{Group\_label} = A) \\
(\text{Age} = 41) \wedge (\text{Salary} = \$48,750) \Longrightarrow (\text{Group\_label} = A) \\
(\text{Age} = 40) \wedge (\text{Salary} = \$52,600) \Longrightarrow (\text{Group\_label} = A)
\end{array}
$$

If the LHS Age bins are $a_1, a_2, \ldots$, and the LHS Salary bins are $s_1, s_2, \ldots$, then these rules are binned to form the corresponding binned association rules:

$$
\begin{array}{c}
(\text{Age} = a_3) \wedge (\text{Salary} = s_5) \Longrightarrow (\text{Group\_label} = A) \\
(\text{Age} = a_4) \wedge (\text{Salary} = s_6) \Longrightarrow (\text{Group\_label} = A) \\
(\text{Age} = a_4) \wedge (\text{Salary} = s_5) \Longrightarrow (\text{Group\_label} = A) \\
(\text{Age} = a_3) \wedge (\text{Salary} = s_6) \Longrightarrow (\text{Group\_label} = A)
\end{array}
$$

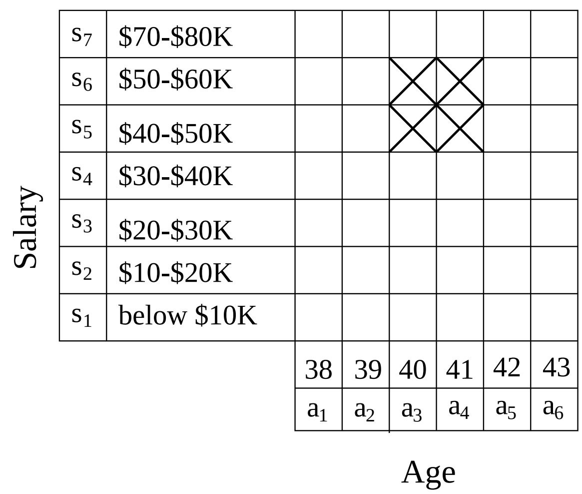


Figure 5: Grid showing the four association rules

We represent these four rules with the grid in Figure 5.[2] Clearly linear, adjacent cells can be combined to form a line segment, and we can naturally extend this idea to rectangular regions. In doing so, all four of the original association rules are clustered together and subsumed by one rule:

$$(a_3 \leq \text{Age} < a_4) \wedge (s_5 \leq \text{Salary} < s_6) \Longrightarrow (\text{Group\_label} = A)$$

Assuming the bin mappings shown in Figure 5, the final clustered rule output to the user is:

$$(40 \leq \text{Age} < 42) \wedge (\$40,000 \leq \text{Salary} < \$60,000) \Longrightarrow (\text{Group\_label} = A)$$

Intuitively, starting with a grid as in Figure 5, we are searching for the rectangular clusters that best cover all of the occupied cells. For example, given the grid in Figure 6, our algorithm would select the two clusters shown if we are interested in finding the fewest number of clusters.

[2] The grid is constructed directly from the (a_i, s_j) pairs output by the association rule engine.

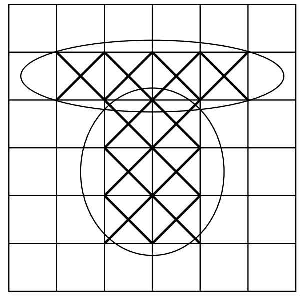


Figure 6:

The problem we investigate is how to find these non-overlapping clusters, or more precisely, how to find a good clustering of rules given some criterion. An example of a simple criterion is minimizing the total number of clusters; in Section 3.6 we will describe the criterion we use. The process of clustering is made difficult by *noise* and *outliers*. Noise occurs in the data from those tuples that belong to other groups than the group we are currently clustering. For example, a customer database could contain data about customers whose group label (RHS attribute) would be one of several values (customer_rating is "average", "above average", or "excellent"), of which "excellent" may be our criterion value for segmenting the data using clusters. Tuples having any other value will be considered noise that will affect both the support and confidence of the clusters we compute. Outliers are those tuples that belong to the same group but lie outside of any existing clusters for that group. We will give examples of these occurrences and show how our clustering system can mitigate their effects in the following sections.

3.3.1 The BitOp Algorithm

The algorithm we present enumerates clusters from the grid. We select the largest cluster from the enumerated list, iteratively applying the algorithm until no clusters remain. It has been shown in [CLR90] that such a greedy approach produces near optimal clusters in $O(\sum_{S \in \mathcal{C}} |S|)$ time, where $\mathcal{C}$ is the final set of clusters found. We experimented with pure greedy cluster selection and complete search of all possible clusters, and we found the greedy approach produced similar results to the non-greedy approach while saving a significant amount of computation. Our greedy approach is outlined in Figure 7.

Our BitOp algorithm, shown in Figure 8, does the cluster enumeration. It performs bitwise operations to locate clusters within the bitmap grid. We examine each row of bits from the bitmap in turn, bitwise ANDing them with the remaining rows. At each step, we can analyze the row of ANDed bits to determine the location of any possible clusters. Consider as an example the following row of bits:

Input:
  I1: A set of (x, y) pairs representing association rules over the binned data.
Output:
  O1: The answer set, Ans, consisting of all the clustered association rules.

for each input (i,j) BM[i,j] $\Leftarrow$ true; /* First, construct the bitmap */
$\mathcal{U} \leftarrow \mathcal{BM}$
$\mathcal{A}ns \leftarrow \emptyset$
While $\mathcal{U} \neq \emptyset$ **do begin**
  /* Greedily choose the cluster $\mathcal{S}$ that covers the most remaining unclustered cells */
  Call BitOp (Figure 8) on $\mathcal{U}$ to generate a set of candidate clusters $\mathcal{P}$
  Select an $S \in \mathcal{P}$ that maximizes $|S \cap \mathcal{U}|$
  $\mathcal{U} \leftarrow \mathcal{U} - \mathcal{S}$
  $\mathcal{A}ns \leftarrow \mathcal{A}ns \cup S$
end-while
Return $\mathcal{A}ns$

Figure 7: Greedy cluster selection

mask	0	1	0	1	1	1	0	0
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*

Just this row alone tells us there is a cluster of size one at bit-position 2 and a cluster of size three at bit-positions 4 through 6. Now consider if the "mask" shown above is the bitwise AND of the first k rows starting at row j of the bitmap, denoted as mask_j^k where $1 \leq j \leq N$, $k \leq N - j + 1$, and N is the number of rows in the bitmap. Then, by definition, we would have a cluster of dimensions 1-by-k and 3-by-k, respectively, since those same bit positions would have to be set in all k rows.

The algorithm begins by defining the first mask, mask_1^1, to be the first row of the bitmap, and then bitwise ANDing it with the second row producing the second mask, mask_1^2. We continue bitwise ANDing this mask with each successor row until we reach the last row or the mask is zero, containing no set bits. We then consider the second row as the starting row, apply the algorithm to successor rows, and so on. If at any step the mask changes, indicating that some bits were cleared, then we know the end of a cluster has been found so we report it. For example, consider this simple figure:

Bitmap

row 3	1	0	0	0
row 2	1	0	1	0
row 1	0	1	1	0

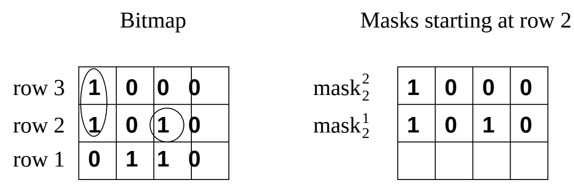
Masks starting at row 1

mask_1^3	0	0	0	0
mask_1^2	0	0	1	0
mask_1^1	0	1	1	0

The first mask, mask_1^1, is defined to be the same as the first row, while the second mask, mask_1^2, is the first mask bitwise ANDed with the second row. Since mask_1^2 is not identical to mask_1^1, we evaluate mask_1^1 for clusters finding a 2-by-1 cluster, indicated by the solid circle. We see that mask_1^2 identifies a 1-by-2 cluster:

1 since a single bit is set and 2 since it is in the second mask meaning the cluster extends two rows. This cluster is indicated by the dashed circle. Finally, mask_1^3 identifies no clusters since the mask contains no set bits, signifying there are no clusters that begin at row 1 and extend through row 3.

We now repeat this process, beginning with the second row of the bitmap, producing the two clusters as shown:



The process ends when the mask for the last row is computed. Our algorithm can be implemented efficiently since it only uses arithmetic registers, bitwise AND, and bit-shift machine instructions. We assume that the size of the bitmap is such that it fits in memory, which is easily the case even for a 1000x1000 bitmap.

3.4 Grid Smoothing

As a preprocessing step to clustering, we apply a 2D *smoothing* function to the bitmap grid. In practice, we have often found that the grids contain jagged edges or small "holes" of missing values where no association rule was found. A typical example is shown in Figure 9(a). These features inhibit our ability to find large, complete clusters. To reduce the effects caused by such anomalies, we use an image processing technique known as a *low-pass filter* to smooth out the grid prior to processing [GW92]. Essentially, a low-pass filter used on a two-dimensional grid replaces a value with the average value of its adjoining neighbors, thereby "smoothing" out large variations or inconsistencies in the grid. The use of smoothing filters to reduce noise is well known in other domains such as communications and computer graphics [GW92]. Details of our filtering algorithm are omitted for brevity, but Figure 9(b) nicely illustrates the results.

3.5 Cluster Pruning

Clusters that are found by the BitOp algorithm but that do not meet certain criteria are dynamically pruned from the set of final candidate clusters. Typically, we have found that clusters smaller than 1% of the overall graph are not useful in creating a generalized segmentation. Pruning these smaller clusters also aids in reducing "outliers" and effects from noise not eliminated by the smoothing step. In the case where the BitOp algorithm finds all of the clusters to be sufficiently large, no pruning is performed. Likewise, if the algorithm cannot locate a sufficiently large cluster, it terminates. The idea of pruning to reduce error and to reduce the size of the result has been used in the AI community, especially for *decision trees* [Qui93].

```
Input:
   I1:  R, the number of bins for attribute X.
   I2:  C, the number of bins for attribute Y.
   I3:  BM, the bitmap representation of the grid.
         (BM[i] is the i^th row of bits in the bitmap)
Output:
   O1:  clusters of association rules

row = 1;
while (row < R) do begin
   RowMask = set all bits to '1';
   PriorMask = BM[row];
   height=0;
   for (r=row; r<R; r++) do begin
      RowMask = RowMask & BM[row];
      if (RowMask == 0) do begin
         process_row(PriorMask,height); /* Output all clusters in given mask */
         break;
      end-if
      if (RowMask != PriorMask) do begin
         process_row(PriorMask,height);
         PriorMask = RowMask;
      end-if
      height = height + 1; /* extend height of possible clusters */
   end-for
   process_row(PriorMask,height);
end-while
```

Figure 8: The BitOp algorithm

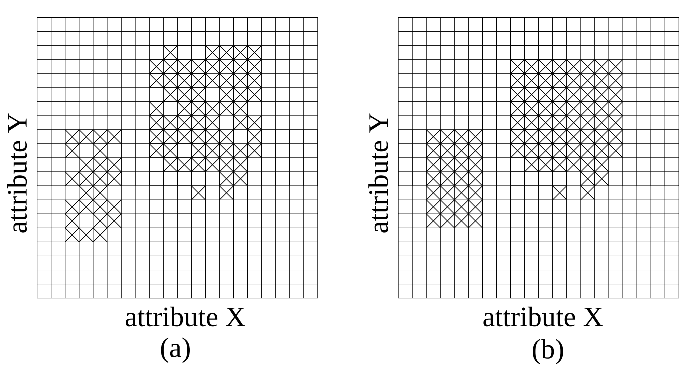


Figure 9: A typical grid (a) prior to smoothing; (b) after smoothing.

Function 2

$$((age < 40) \wedge (50K \leq salary \leq 100K)) \implies \text{Group A} \quad (1)$$

$$((40 \leq age < 60) \wedge (75K \leq salary \leq 125K)) \implies \text{Group A} \quad (2)$$

$$((age \geq 60) \wedge (25K \leq salary \leq 75K)) \implies \text{Group A} \quad (3)$$

$$\text{else} \implies \text{Group other}$$

Figure 10: The function used to generate synthetic data

3.6 Cluster Accuracy Analysis

To determine the quality of a segmentation by a set of clustered association rules we measure two quantities: (i) the number of rules computed, and (ii) the summed error rate of the rules based on a sample of the data. The two quantities are combined using the *minimum description length* (MDL) principle [Ris89] to arrive at a quantitative measure for determining the quality compared to an "optimal" segmentation. We first describe our experiments, then we describe our error measure for a given rule, then we describe our application of the MDL principle.

In [AIS93a], a set of six quantitative attributes (*salary, commission, age, hvalue, hyears, loan*) and three categorical attributes (*education_level, car, zip code*) for a test database are defined. Using these attributes, 10 functions of various complexity were listed. We used Function 2, shown in Figure 10, to generate the synthetic data used in our experiments. The optimal segmentation for this data would be three clustered association rules, each of which represents one of the three disjuncts of the function. The clustering process is made difficult when we introduce noise, random perturbations of attribute values, and error due to binning of the input attributes **age** and **salary**.

An intuitive measure of the accuracy of the resulting clustered rules would be to see how well the rectangular clusters overlap the three precise disjuncts of the function in Figure 10. We define the notion of *false-positives* and *false-negatives* graphically, as shown in Figure 11, and seek to minimize both sources of error. In Figure 11, the light-grey rectangle represents an actual cluster according to the function, and the dark-grey rectangle represents a computed cluster. The false-positive results are when the computed cluster incorrectly identifies tuples outside of the optimal cluster as belonging to the specified group, whereas the false-negatives are tuples that should belong to the group but are not identified as such by the computed cluster. The total summed error for a particular cluster is the total (false-positives + false-negatives). However, unless the optimal clustering is known beforehand, such as here where a function is used to generate the data, this exact measure of error is not possible. Because we are interested in real-world data where the

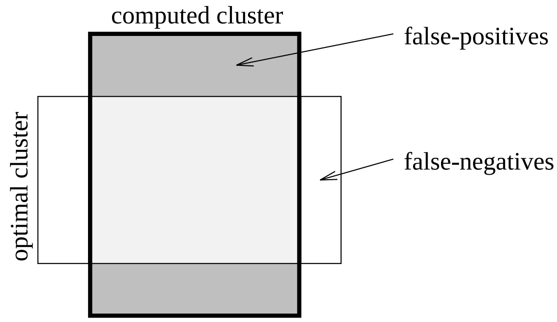


Figure 11: Error between overlapping regions

optimal clustering is not known, we instead select a random sample of tuples from the database and use these samples to determine the *relative* error of the computed clusters. The relative error is only an approximation to the exact error since we are counting the number of false-negatives and false-positives based only on a sample of the original database. In order to get a good approximation to the actual error, we use repeated k out of n sampling, a stronger statistical technique.

The strategy we use to measure the quality of a segmentation given a set of clustered association rules is based on the MDL principle. The MDL principle states that the best model for encoding data is the one that minimizes the sum of the cost of describing the model and the cost of describing the data using that model. The goal is to find a model that results in the lowest overall cost, with cost typically measured in bits.

In the context of clustering, the models are the descriptions of the clusters and the data is the sampled data described above. The greater the number of clusters used for segmentation, the higher the cost necessary to describe those clusters. The cost of encoding the sampled data using a given set of clusters (the model) is defined to be the sum of all errors for the clusters. The intuition is that if a tuple is not an error, then it is identified by a particular cluster and hence its cost is included with the description of the cluster. Otherwise, if the tuple is an error, then we must specifically identify it as such and this incurs a cost. We use the following equation to determine the cost of a given set of clustered association rules:

$$\text{cost} = w_c \log_2(|C|) + w_e \log_2(errors)$$

where $|C|$ is the number of clusters and $errors$ is the sum of (false-positives + false-negatives) for the clusters C. The constants w_c and w_e allow the user to impart a bias towards "optimal" cluster selection. If w_c is large, segmentations that have many clusters will be penalized more heavily since they will have a higher associated cost, and segmentations of the data that have fewer clusters will have a greater probability of being "optimal". Likewise, if w_e is large, the system will favor segmentations where the error rate is lowest. If both constants are equal, $w_c = w_e = 1$, as in the default case, neither term will bias the cost.

Our heuristic optimizer (recall Figure 2), by means described in the following section, seeks to minimize the MDL cost.

3.7 Parameter Heuristics

In this section we describe the algorithm our overall system uses to adjust the minimum support and confidence thresholds based upon the accuracy analysis from the previous section. (currently the number of bins for each attribute is preset at 50. We discuss this issue more in the following section.) We desire values for support and confidence that will result in a segmentation of the data that optimizes our MDL cost function. The search process involves successive iterations through the feedback loop shown in Figure 2. We identify the actual support and confidence values that appear in the binned data, and use only these values when adjusting the ARCS parameters. We begin by enumerating all unique support thresholds from the binned data with one pass, and then all of the unique confidence thresholds for each of these support thresholds with a second pass. A data structure similar to that shown in Figure 12 is used to maintain these values. Note that as support increases, there become fewer and fewer cells that can support an association rule, and we have found a similar decrease in the variability of the confidence values of such cells.

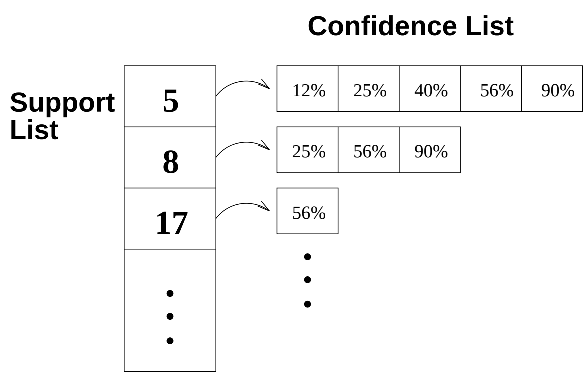


Figure 12: Ordered lists of confidence and support thresholds

Given a choice to either begin with a low support threshold and search upwards, or begin with a high support threshold and search downwards, we chose the former since we found most "optimal" segmentations were derived from grids with lower support thresholds. If we were using a previous association rule mining algorithm, for efficiency it might be preferable to start at a high support threshold and work downwards, but our efficient mining algorithm allows us to discover segmentations by starting at a low support threshold and working upwards. Our search starts with a low minimum support threshold so that we consider a larger number of association rules initially, allowing the dynamic pruning performed by the clustering algorithm to remove unnecessary rules. The support is gradually increased to remove background noise and outliers until there is no improvement of the clustered association rules (within some ϵ). The search algorithm is shown in Figure 13.

```
Procedure ParameterizedSearch()
   /* The main loop of the system */
   set bin_x = 50; set bin_y = 50;
   set support_lvl = 0;
   set old_min_cost = +Infinity;
   Compute Support List and Confidence Lists
   set maxlevels = number of elements in the Support List;
   Do Begin
      set min_support = SupportList[support_lvl]; /* next lowest support level */
      set min_cost = +Infinity;
      for (conf_lvl=0; conf_lvl < ConfidenceList[support_lvl].len; conf_lvl++) do begin
         set min_confidence = confidenceList[support_lvl].value[conf_lvl];
         Perform cluster mining
         Compute MDL cost from cluster accuracy analysis phase
         if (cost < min_cost) then min_cost = cost;
      end-for
      delta = old_min_cost - min_cost;
      old_min_cost = min_cost;
      support_lvl = support_lvl + 1;
   While ((delta > -ε) && (support_lvl < maxlevels))
```

Figure 13: ARCS search algorithm

4 Experimental Results

The ARCS system and the BitOp algorithm have been implemented in C++ (comprising approximately 6,300 lines of code). To assess the performance and results of the algorithms in the system, we performed several experiments on an Intel Pentium workstation with a CPU clock rate of 120 MHz and 32MB of main memory, running Linux 1.3.48. We first describe the synthetic rules used in generating data and present our initial results. We then briefly compare our results with those using a well-known classifier, C4.5, to perform the segmentation task. Finally, we show performance results as the sizes of the databases scale.

4.1 Generation of Synthetic Data

We generated synthetic tuples using the rules of Function 2 from Figure 10. Several parameters affect the distribution of the synthetic data. These include the fraction of the overall number of tuples that are assigned to each value of the criterion attribute, a perturbation factor to model fuzzy boundaries between disjuncts, and an outlier percentage that defines how many tuples will be assigned to a given group label but do not match any of the defining rules for that group. These parameters are shown in Table 1.

Attribute	Value
salary	uniformly distributed from \$20,000 to \$150,000
age	uniformly distributed from 20 to 80
$\|D\|$	Number of tuples, 20,000 to 10 million
frac_{GroupA}	Fraction of tuples for "Group A", 40%
$\text{frac}_{Groupother}$	Fraction of tuples for "Group other", 60%
ρ	Perturbation factor, 5%
σ	Outlier percentage, 0% and 10%

Table 1: Synthetic data parameters

4.2 Accuracy and Performance Results

Figure 14 shows a plot of the generated data for Function 2 with $|D| = 50{,}000$ and 5% perturbation. The second plot, shown in Figure 15, represents the function but with 10% outliers, i.e., 10% of the data are outliers that do not obey the generating rules. In every experimental run we performed, ARCS always produced three clustered association rules, each very similar to the generating rules, and effectively removed all noise and outliers from the database. The following clustered association rules were generated from the data of Figure 14 with a minimum support threshold of 0.01% and a minimum confidence threshold of 39.0%:

$$(20 \leq \text{Age} \leq 39) \ \wedge \ (\$48601 \leq \text{Salary} \leq \$100600) \Longrightarrow \text{Group A}$$
$$(40 \leq \text{Age} \leq 59) \ \wedge \ (\$74601 \leq \text{Salary} \leq \$124000) \Longrightarrow \text{Group A}$$
$$(60 \leq \text{Age} \leq 80) \ \wedge \ (\$25201 \leq \text{Salary} \leq \$74600) \Longrightarrow \text{Group A}$$

The reader can compare the similarity of these rules with those used to generate the synthetic data in Figure 10.

We measured the error rate of ARCS on these databases and compared it to rules from C4.5. C4.5 is well known for building highly accurate decision trees that are used for classifying new data, and from these trees a routine called C4.5RULES constructs generalized rules [Qui93]. These rules have a form similar to our clustered association rules, and we use them for comparison, both in accuracy and in speed of generation. Figures 16 and 17 graph the error of both systems as the number of tuples scale, using the data from Figures 14 and 15, respectively. The missing bars for C4.5 on larger database sizes are due to the depletion of virtual memory for those experiments, resulting in our inability to obtain results (clearly C4.5 is not suited to large-scale data sets.) From Figure 16, we see that C4.5 rules generally have a slightly lower error rate than ARCS clustered association rules when there are no outliers in the data. However, with 10% of the data appearing as outliers the error rate of C4.5 is slightly higher than ARCS, as shown in Figure 17. C4.5 also comes at a cost of producing significantly more rules, as shown in Figures 18 and 19. As mentioned earlier, we are targeting environments where the rules will be processed by end users, so keeping the number of rules small is very important.

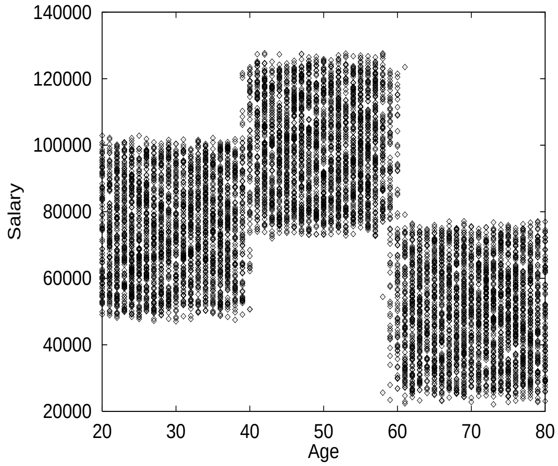


Figure 14: Synthetic data for $\rho = 5\%$ and $\sigma = 0\%$

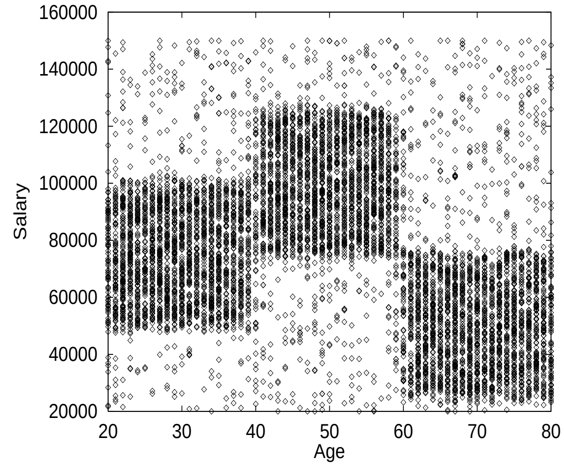


Figure 15: Synthetic data for $\rho = 5\%$ and $\sigma = 10\%$

The primary cause of error in the ARCS rules is due to the granularity of binning. The coarser the granularity, the less likely it will be that the computed rules will have the same boundary as the generating rules. To test this hypothesis, we performed a separate set of identical experiments using between 10 to 60 bins for each attribute. We found a general trend towards more "optimal" clusters as the number of bins increases, as shown in Figure 20. This figure shows the lowest MDL cost, as defined in Section 3.6, where a lower cost indicates a better segmentation.

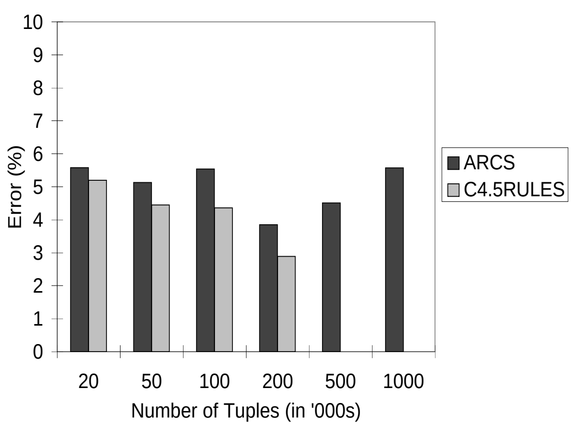


Figure 16: Error rate with $\sigma = 0\%$

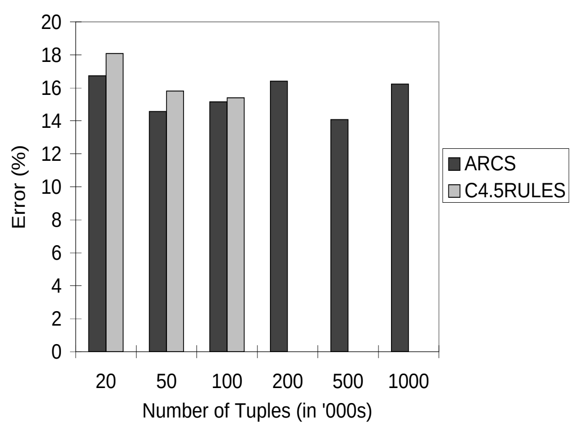


Figure 17: Error rate with $\sigma = 10\%$

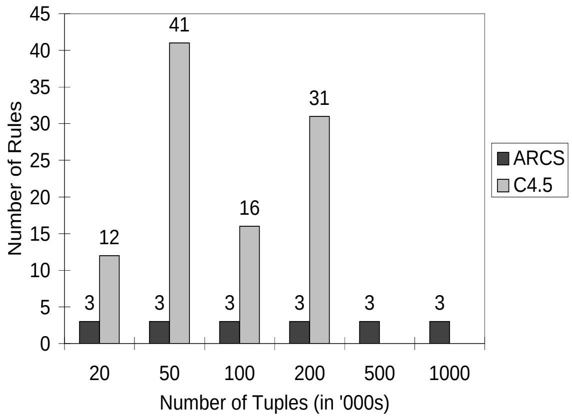


Figure 18: Number of rules produced, $\sigma = 0\%$

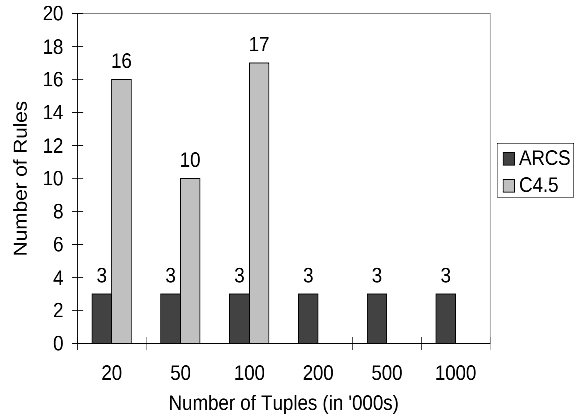


Figure 19: Number of rules produced, $\sigma = 10\%$

4.3 Scaleup Experiments

To test scalability, we ran ARCS on several databases with increasing numbers of tuples. Figure 21 shows that execution time increases at most linearly with the size of the database. Because ARCS maintains only the BinArray and the bitmap grid, ARCS requires only a constant amount of main memory regardless of the size of the database (assuming the same number of bins). This actually gives our system significantly better than linear performance, as can be seen by close inspection of Figure 21, since some overhead exists initially but the data is streamed in faster from the I/O device with larger requests. For example, when the number of tuples scales from 100,000 to 10 million (a factor of 100), the execution time increases from 42 seconds to 420 seconds (a factor of 10). In comparison, C4.5 requires the entire database, times some factor, to fit entirely in main memory. This results in paging and eventual depletion of virtual memory (which prevented us from obtaining execution times or accuracy results for C4.5 rules from databases greater than 100,000 tuples). Both C4.5 alone and C4.5 together with C4.5RULES take exponentially higher execution times than ARCS, as shown in Table 2.

5 Conclusions and Future Work

In this paper we have investigated clustering two-attribute association rules to identify generalized segments in large databases. The contributions of this paper are summarized here:

- We have presented an automated system to compute a clustering of the two-attribute space in large databases.

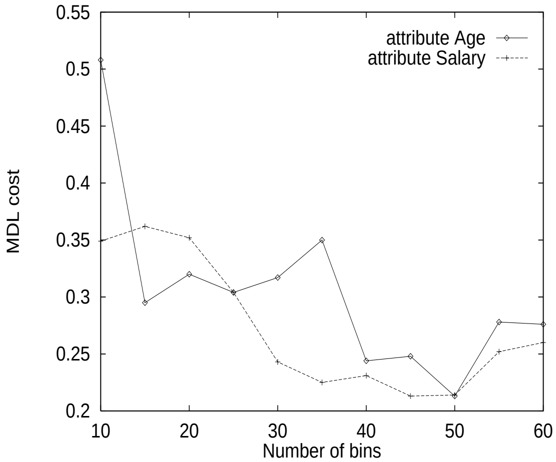


Figure 20: Better results with more bins

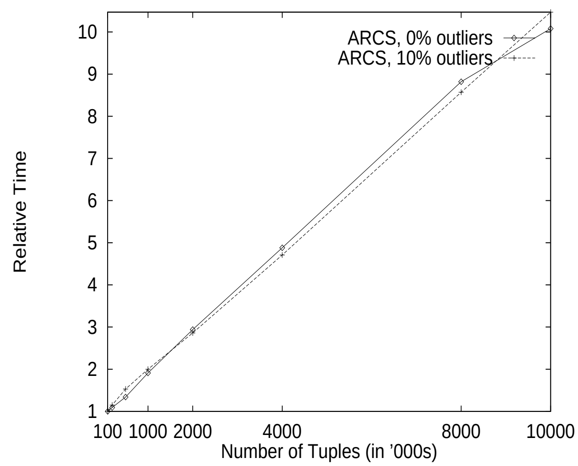


Figure 21: Scalability of ARCS

	ARCS		**C4.5**		**C4.5 + C4.5RULES**	
$\|D\|$	σ=0%	σ=10%	σ=0%	σ=10%	σ=0%	σ=10%
20,000	27	28	14	27	751	4,587
50,000	37	43	81	190	8,260	45,487
100,000	42	41	210	893	39,331	424,353
200,000	45	47	650	2,952	n/a	n/a
500,000	56	62	4,018	19,581	n/a	n/a
1 million	80	82	14,808	85,985	n/a	n/a
2 million	123	117	n/a	n/a	n/a	n/a
4 million	203	192	n/a	n/a	n/a	n/a
8 million	367	349	n/a	n/a	n/a	n/a
10 million	420	426	n/a	n/a	n/a	n/a

Table 2: Comparative execution times (sec)

- We have demonstrated how association rule mining technology can be applied to the clustering problem. We also have proposed a specialized mining algorithm that only makes one pass through the data for a given partitioning of the input attributes, and allows the support or confidence thresholds to change without requiring a new pass through the data.

- A new geometric algorithm for locating clusters in a two-dimensional grid was introduced. Our approach has been shown to run in linear time with the size of the clusters. Further, parallel implementations of the algorithm would be straightforward.

- We apply the Minimum Description Length (MDL) principle as a means of evaluating clusters and use this metric in describing an "optimal" clustering of association rules.

- Experimental results show the usefulness of the clustered association rules and demonstrates how the proposed system scales in better than linear time with the amount of data.

The algorithm and system presented in the paper have been implemented on several platforms, including Intel, DEC, and SGI. So far we have performed tests only using synthetic data, but intend to examine real-world demographic data. We also plan on extending this work in the following areas:

- It may be desirable to find clusters with more than two attributes. One way in which we can extend our proposed system is by iteratively combining overlapping sets of two-attribute clustered association rules to produce clusters that have an arbitrary number of attributes.

- Handle both categorical and quantitative attributes on the LHS of rules. To obtain the best clustering, we will need to consider all feasible orderings of categorical attributes. Our clustering algorithm has been extended to handle the case where one attribute is categorical and the other quantitative and we achieved good results. By using the ordering of the quantitative attribute, we consider only those subsets of the categorical attribute that yield the densest clusters.

- We did not investigate the use of equi-depth or homogeneity-based binning. Furthermore, an incremental equi-width technique that starts with a large number of bins initially could combine similar adjacent bins without the need to reread and re-bin the source data.

- It may be beneficial to apply measures of *information gain* [Qui86], such as entropy, when determining which two attributes to select for segmentation or for the optimal threshold values for support and confidence.

- The technique of *factorial design* by Fisher [Fis60, BHH78] can greatly reduce the number of experiments necessary when searching for "optimal" solutions. This technique can be applied in the heuristic optimizer to reduce the number of runs required to find good values for minimum support and minimum confidence. Other search techniques such as simulated annealing can be also be used in the optimization step.

- Preliminary experiments show that smoothing can be improved if the association rule support values are considered in the low-pass filter, and more advanced filters could be used for purposes of detecting edges and corners of clusters.

- Where taxonomies (*is-a* hierarchies) exist for attributes of the data, we would like to use these in deriving more general clusters. For example, if we know the taxonomy 'raincoat' *is-a* 'jacket', 'parka' *is-a* 'jacket', and 'sportscoat' *is-a* 'jacket', and we have a clustered association rule such as

$$(30 \leq \text{Age} < 40) \text{ and } (\text{Items} \in \{raincoat, parka, sportscoat\}) \Longrightarrow (\text{Frequent\_buyer})$$

then we can use the taxonomy to reduce this rule to

$$(30 \leq \text{Age} < 40) \text{ and } (\text{Items} = \text{jacket}) \Longrightarrow (\text{Frequent\_buyer})$$

Acknowledgements

We are grateful to Dan Liu for his work in extending functionality in the synthetic data generator and for the experiments using C4.5.

References

[AGI+92] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proceedings of the 18th International Conference on Very Large Data Bases*, Vancouver, Canada, 1992.

[AIS93a] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. In *IEEE Transactions on Knowledge and Data Engineering*, volume 5(6), pages 914–925, December 1993.

[AIS93b] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C., 1993.

[BHH78] G. E.P. Box, W.G. Hunter, and J. Stuart Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building.* John Wiley and Sons, 1978.

[CLR90] T.H. Cormen, C.E. Lieserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[Fis60] R. A. Fisher. *The Design of Experiments.* Hafner Publishing Company, 1960.

[FMMT96] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

[GW92] R. C. Gonzalez and R.E. Woods. *The Design of Experiment.* Addison-Wesley, 1992.

[KMRT94] M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. Finding interesting rules from large sets of discovered association rules. In *3rd International Conference on Information and Knowledge Management (CIKM)*, November 1994.

[Kru78] J. B. Kruskal. Factor analysis and principle components: Bilinear methods. In H. Kruskal and J.M. Tanur, editors, *International Encyclopedia of Statistics.* Free Press, 1978.

[Law71] D. N. Lawley. *Factor Analysis as a Statistical Method.* American Elsevier Publishing Company, second edition, 1971.

[Lub89] D. J. Lubinsky. Discovery from databases: A review of ai and statistical techniques. In *IJCAI–89 Workshop on Knowledge Discovery in Databases*, pages 204–218, 1989.

[MAR96] M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.

[MD88] M. Muralikrishna and D. DeWitt. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3), September 1988.

[PS91] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. *Knowledge Discovery in Databases*, 1991.

[Qui86] J.R. Quinlan. Induction of decision trees. In *Machine Learning*, volume 1, pages 81–106, 1986.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[Ris89] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.

[Rom84] H. Charles Romesburg. *Cluster Analysis for Researchers*. Lifetime Learning Publications-Wadsworth Inc., Belmont, California, 1984.

[SA96] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

[SAM96] J. Shafer, R. Agrawal, and M. Mehta. Fast serial and parallel classification of very large data bases. In *Proceedings of the 22nd International Conference on Very Large Databases*, Bombay, India, 1996.

[Spa80] H. Spath. *Cluster Analysis Algorithms for data reduction and classification of objects*. Ellis Horwood Publishers, 1980.

[WKW94] K. Whang, S. Kim, and G. Wiederhold. Dynamic maintenance of data distribution for selectivity estimation. *VLDB Journal*, 3(1), January 1994.