Pattern-Based Clustering for Database Attribute Values *

Matthew Merzbacher   Wesley W. Chu

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

We present a method for automatically clustering similar attribute values in a database system spanning mulitple domains. The method constructs an *attribute abstraction hierarchy* for each attribute using rules that are derived from the database instance. The rules have a confidence and popularity that combine to express the "usefullness" of the rule.

Attribute values are clustered if they are used as the premise for rules with the same consequence. By iteratively applying the algorithm, a hierarchy of clusters can be found. The algorithm can be improved by allowing domain expert supervision during the clustering process. An example as well as experimental results from a large transportation database are included.

1 Introduction

In a conventional database system, queries are answered with absolute certainty. If a query has no exact answer, then the user's needs remain unsatisfied. A cooperative query answering (CQA) system behaves like a conventional system when the query can be answered normally, but tries to find an approximate answer when the original query condition cannot be matched exactly [5, 6]. When a query has no answer, the query conditions are relaxed to obtain an approximate answer. For example, consider a query about C–21 cargo aircraft in a transportation application. If no such aircraft is available, the query can be approximated by replacing C–21 with a nearby approximation, such as C–12 or C–20.

To restrict the search space of relaxation to a small set of possible answers, we employ an abstraction mechanism that can discriminate likely candidates efficiently. The *attribute abstraction hierarchy (AAH)* groups like values for each attribute. Attributes are relaxed by finding their value in the hierarchy and replacing it with other values in the same cluster. If searching one cluster in the hierarchy is unsuccessful in yielding an answer, then the condition can be further relaxed by traversing up another level in the hierarchy. Eventually, the query will be sufficiently relaxed to generate an answer[3]. There is an AAH for each attribute domain. Figure 1 shows a classification of aircraft types for the aircraft example. Each level of the hierarchy represents a level of abstraction. For example, C–21 is clustered with C–12 and C–20 at the bottom level and with C–23 and C–130 aircraft at the next level, and so on. The numbers in the hierarchy represent the "nearness" of the values below each node, ranging from zero to one.

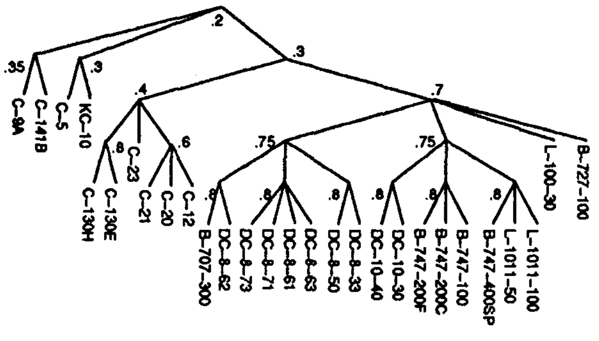

Figure 1: AAH for Aircraft Type

For databases with many attributes having a large number of values, manaully constructing the AAHs is time consuming and prone to error. Thus the hierarchies must be induced automatically, but previous knowledge discovery techniques are inadequate for CQA. Attribute Oriented Induction [1] provides summary information and charactarizes tuples in the database, but is inappropriate for attribute values and focuses too closely on a spe-

*Supported by DARPA contract N00174-91-C-0107

Aircraft	Length		Runway	Turning	Taxi	Weight		Storage
Type	Takeoff	Landing	Width	Radius	Space	Max.	Empty	Space
C–5	13600	5000	90	150	60	769000	665000	21508
C–141B	9000	5000	90	137	50	343000	239000	10454
KC–10	11800	5400	90	148	60	590000	414000	11600
C–130E	6250	3000	60	74	40	173700	NULL	5173
C–130H	6250	3000	60	74	40	173700	NULL	5173
C–9A	8100	3200	10	72	50	108000	90000	4325
C–12	4500	4500	75	43	40	12500	11000	915
C–21	6000	3400	75	53	40	18300	13500	747
C–20	5000	5000	75	52	40	69700	44000	2518
C–23	4000	3000	60	54	30	22900	16200	1690
B–707–300	10800	7000	90	124	50	336600	230000	5600
DC–8–33	9400	5500	90	98	50	315000	224000	10000
DC–8–50	9400	5500	90	98	50	315000	224000	10000
DC–8–61	10400	6100	90	113	50	325000	261000	8100
DC–8–62	11500	6100	90	115	50	350000	230000	8825
DC–8–63	10400	6100	90	132	50	355000	261000	10800
DC–8–71	8850	6100	90	113	50	355000	261000	10800
DC–8–73	9800	6100	90	132	50	355000	261000	10800
DC–10–10	9500	5800	90	144	75	440000	335000	11000
DC–10–30	11850	6100	90	150	75	572000	391000	11600
DC–10–40	10600	5800	90	150	75	572000	391000	11600
B–747SP	7500	5600	90	122	75	696000	410000	14100
B–747–100	9500	6600	90	142	75	750000	526400	17500
B–747–200C	11800	6600	90	142	75	775000	590000	17500
B–747–200F	10500	6600	90	142	75	775000	590000	17500
L–1011–100	10640	7300	90	142	75	466000	320000	10800
L–1011–500	9760	7300	90	127	75	496000	338000	10800
B–727–100	9200	4800	90	80	60	207500	160000	5600
L–100–30	6000	4900	75	85	50	155000	122000	5800

Table 1: The database table *AIRCRAFT*

cific target. Conceptual Clustering [8, 7] is a top-down method, iteratively subdividing the tuple-space into smaller sets. The top-down approach does not yield clusters the best correlation near the bottom of the hierarchy. CQA operates from the bottom of the hierarchy, so better clustering near the bottom is desirable. Further, we require a nearness measure for the clustered attribute values. To remedy these shortcomings, we present a bottom up approach for constructing attribute abstraction hierarchies called *Pattern-Based Knowledge Induction* (PKI) that includes a nearness measure for the clusters.

PKI can cluster attribute values with or without expert direction, and thus is well-suited to non-numeric domains. However, it also works for numerical domains, particularly when pre-clustered sub-ranges of the numerical domain are identified in advance. PKI also works well when there are NULL values in the data. Our experimental results confirm that the method is scalable to large systems.

PKI determines clusters by deriving rules from the instance of the current database. The rules are not 100% certain; instead, they are rules-of-thumb about the database, such as

If AIRCRAFT is in the class of B–747 then
  RUNWAY TAKEOFF LENGTH >= 6600

Each rule has a *popularity* that measures how often the rule applies, and a *confidence* indicating how well the rule "fits" the database. In certain cases, a more sophisticated rule with high confidence can be derived by combining simpler rules.

The PKI approach generates a set of useful rules which can then be used to construct the AAH by clustering the premises of rules sharing a similar consequence. For example, if the following three rules:

If AIRCRAFT = B–747–200C then
RUNWAY TAKEOFF LENGTH <= 6600 feet
If AIRCRAFT = B–747–200F then
RUNWAY TAKEOFF LENGTH <= 6600 feet
If AIRCRAFT = B–747–100 then
RUNWAY TAKEOFF LENGTH <= 6600 feet

have high confidence, then this indicates that these three types of 747 should be clustered together. Supporting and contradicting evidence from rules for other attributes is gathered and, based on weights assigned by an expert, PKI builds an initial set of clusters. Each invocation of the clustering algorithm adds a layer of abstraction to the hierarchy. The attribute abstraction hierarchy is constructed by applying the algorithm iteratively.

This paper is organized as follows. We first introduce the example domain in detail. Then, after reviewing terms and definitions, we present the Pattern-Based Knowledge Induction (PKI) algorithm for classifying attribute values into clusters. We then present the experimental results of clustering a large transportation database. An example is also given to illustrate the principle using the relation shown in Table 1.

2 Pattern–Based Knowledge Induction (PKI)

In this section, we show how pattern-based knowledge induction systematically acquires rules from database instances. The induced knowledge is then used to cluster semantically similar values.

2.1 Definitions

Patterns are conditions on attributes in the database, such as:

RUNWAY WIDTH > 80, *or*
TURNING RADIUS = 144 feet

The objects that satisfy the pattern are said to *match the pattern.*

Cardinality The cardinality of a pattern P, denoted $|P|$, is the number of distinct objects that match P.

For example, from the data, the pattern 140 <= TURN RADIUS <= 150 is matched by nine tuples and thus has a cardinality of nine.

Rule A rule is an inferential relationship between two patterns A and B, represented by $A \rightarrow B$, indicating that when A is true, B also holds. A is the *premise* and B is the *consequence* of the rule.

The usefulness of a rule can be measured in terms of how often and how well the rule applies to the database. To provide quantitative values for these properties, we introduce the notions of confidence and popularity, each defined in terms of cardinality. The *confidence* of a rule is an indication of how often that rule holds true.

Confidence The confidence of a rule $A \rightarrow B$, denoted by $\xi(A \rightarrow B)$, is

$$\xi(A \rightarrow B) = \frac{|A \cap B|}{|A|} \qquad (1)$$

Confidence ranges from 0 to 1. If the confidence of a rule is 1 then the rule is *deterministic.*

For example, assuming $|A| = 100$, $|B| = 320$, and $|A \cap B| = 80$, then $\xi(A \rightarrow B) = 80\%$, however, $\xi(B \rightarrow A) = 25\%$. Note that the confidences of rules for two opposite inference directions are neither symmetric nor complementary.

More sophisticated boolean combinations of rules can be derived from the basic rules [2].

Popularity is another important measure of a rule, indicating "how common" the rule is. Consider the rule

TYPE = C–5 → RUNWAY WIDTH = 90 ft.

that has confidence of 100% but only applies to one tuple in *AIRCRAFT*. This rule has less semantic importance than another rule with lower confidence, but which applies to more of the tuples in the database, such as

AIRCRAFT is in the class B–747 →
RUNWAY TAKEOFF LENGTH = 6600 feet

which applies to four tuples (although it only holds for three of the four).

The *popularity* of a rule measures how often a rule can be applied. Thus, popularity is solely dependent on the premise of the rule and independent of the consequence.

Popularity Let A→B be a rule that applies to a relation R, and $|R|$ be the number of tuples in R (cardinality of R). The popularity of the rule over R is defined as

$$\eta(A \rightarrow B) = \frac{|A|}{|R|} \qquad (2)$$

2.2 Knowledge Induction

We now show how to gather a basic set of inferential rules which we will use for clustering. The premise and consequence of the gathered rules will be comprised entirely of *atomic patterns*, which are the most basic kind of pattern.

Atomic Patterns Atomic patterns are the patterns whose *conditions* are on a single attribute, such as "RUNWAY TAKEOFF LENGTH = 5000" or "WEIGHT = 100 tons".

The atomic pattern set for RUNWAY WIDTH attribute in Table 1 is:

RUNWAY WIDTH = 10
RUNWAY WIDTH = 60
RUNWAY WIDTH = 75
RUNWAY WIDTH = 90

The algorithm to induce knowledge from a relation via atomic patterns follows directly:

Algorithm: Induce Rules

```
Derive atomic patterns for each attribute
For each pair of atomic patterns $(I,J)$
    consider $I \rightarrow J$ as a candidate rule
    calculate the popularity and
        confidence of $I \rightarrow J$
```

For storage and maintanence reasons, rules with popularity or confidence below certain threshold values may be discarded, though in this paper we retain all rules, as our clustering algorithm discounts rules with low popularity and confidence. The *AIRCRAFT* relation alone has 1790 atomic rules.

3 The Clustering Algorithm

We now show how to use the rules generated by PKI to construct a hierarchy. The cluster algorithm groups attribute values which appear as premises in rules with the same consequence based on the following rule:

Rule of Shared Consequence If two rules share a consequence and have the same attribute as a premise (but different values), then those values are candidates for clustering.

For example, the two rules

$$\begin{array}{ll} \text{A} = a_1 \rightarrow \text{B} = b & (\text{ confidence } = \xi_1 \text{ }) \\ \text{A} = a_2 \rightarrow \text{B} = b & (\text{ confidence } = \xi_2 \text{ }) \end{array} \qquad (3)$$

have identical consequences (B $= b$) and have the same attribute (A) as a premise with different values (a_1 and a_2).

3.1 The Clustering Correlation

Using the rule of shared consequence, the *clustering correlation* between two values is the product of the confidences of the two rules, $\xi_1 \times \xi_2$. To account for two values sharing consequences from several different attributes, the overall clustering correlation between two attribute values, γ, is the sum of the individual correlations. That is, if a_1 and a_2 are values of attribute A, and there are m attributes $\text{B}_1, \text{B}_2, \ldots, \text{B}_m$ in the relation, then

$$\gamma(a_1, a_2) = \sum_{i=1}^{m} \xi(\text{A} = a_1 \rightarrow \text{B}_i = b_i) \times \xi(\text{A} = a_2 \rightarrow \text{B}_i = b_i) \qquad (4)$$

We normalize by dividing by $m - 1$.

$$\overline{\gamma}(a_1, a_2) = \frac{1}{m-1}\gamma(a_1, a_2) \qquad (5)$$

By inspection, $\gamma(a_1, a_2) = \gamma(a_2, a_1)$ and thus $\overline{\gamma}(a_1, a_2) = \overline{\gamma}(a_2, a_1)$. $\overline{\gamma}$ measures the "nearness" of two attribute values, and can thus be used for binary clustering.

3.2 Binary Clustering

We now present a binary clustering algorithm that uses $\overline{\gamma}$ values to identify clusters.

Algorithm: Binary Cluster

```
repeat
    INDUCE RULES and determine $\overline{\gamma}$
    sort $\overline{\gamma}$ in descending order
    for each $\overline{\gamma}(a_i, a_j)$
        if $a_i$ and $a_j$ are unclustered
            cluster $a_i$ and $a_j$
            replace $a_i$ and $a_j$ in DB
                with joint value $J_{i,j}$
until fully clustered
```

The binary clustering algorithm is a "greedy" algorithm, putting the two attribute values with the highest overall $\overline{\gamma}$ together. Then, the two values with the next highest $\overline{\gamma}$ are clustered, but only if those values have not already been clustered, and so on, until $\overline{\gamma}$ falls below a specified threshold value or all values are clustered.

To construct the next level of the hierarchy, we substitute a joint value $J_{i,j}$ for each set of values a_i and a_j that are clustered together. We repeat the algorithm using the new values, generating new rules, recalculating $\overline{\gamma}$, and clustering the highest new values. The process repeats until the hierarchy is complete.

3.3 n-ary Clustering

The binary clustering algorithm yields a binary hierarchy insted of a more general n-ary hierarchy. The obvious extension to achieve n-ary clustering is to use a generalized $\overline{\gamma}$ that measures the correlation between an arbitrary n values instead of just two. However, evaluating such a formula would require considering all possible combinations of values instead of all possible pairs, and would thus be too costly for a large number of attribute values.

Instead of modifying $\overline{\gamma}$ to allow n-ary clustering, we approximate the n-ary $\overline{\gamma}$ using a combination of the binary values. Multiple values are clustered together if the $\overline{\gamma}$ between each possible pair of them is above a prespecified threshold; that is, a_1, a_2 and a_3 are clustered together if $\overline{\gamma}(a_1, a_2)$, $\overline{\gamma}(a_1, a_3)$, and $\overline{\gamma}(a_2, a_3)$ are all above the threshold.

After each iteration, the threshold is reduced by a pre-specified amount to allow more clustering and create a hierarchy. Eventually, the threshold is low enough to include all values. We note that the user may select different threshold values for the correlation coefficient to control the depth of the n-ary AAH. These threshold values are based on application context and requirements.

3.4 Weighted Clustering

Thusfar, each attribute has had an equal weighting on the clustering over every other attribute. In general, different attributes should have varying impact on the final clustering. To allow control over the selection of attributes for clustering, we allow the expert to assign weights to each attribute. Let $\overline{\gamma_w}$ be the normalized weighted sum of the rules, and $w_{\mathrm{B}_i}(\mathrm{A})$ be the weight assigned to attribute B_i when clustering attribute A, then,

$$\begin{aligned}\overline{\gamma_w}(a_1, a_2) &= \frac{1}{W(\mathrm{A})}\sum_{i=1}^{m} w_{\mathrm{B}_i}(\mathrm{A}) \times \\ &\quad \xi(\mathrm{A} = a_1 \rightarrow \mathrm{B}_i = b) \times \\ &\quad \xi(\mathrm{A} = a_2 \rightarrow \mathrm{B}_i = b) \qquad (6)\end{aligned}$$

where

$$W(\mathrm{A}) = \sum_{i=1}^{m} w_{\mathrm{B}_i}(\mathrm{A}) \qquad (7)$$

Thus, different clusters can be constructed by assigning weights for each attribute based on domain context and user profile.

4 A Transportation Example

Consider a transportation database used to simulate scenarios of troop and supply movement. The database is used by strategists to plan a time phased force deployment and contains specifications of aircraft, ships and the cargo they carry. As an example, we have selected the *AIRCRAFT* relation, which includes various requirements about the aircraft used in the database as specified in the nine attributes in the example relation shown in Table 1. In this example, we will cluster the attribute AIRCRAFT TYPE based on the other eight attributes.

We first calculate $\overline{\gamma}$ for each pair of values in AIRCRAFT TYPE, shown in Table 2. Due to space considerations, only pairs with $\overline{\gamma}$ over .75 are listed.

a_1	a_2	$\overline{\gamma}(a_1, a_2)$
C-130E	C-130H	1.00
DC-8-33	DC-8-50	1.00
DC-8-63	DC-8-73	.97
DC-10-30	DC-10-40	.95
B-747-200C	B-747-200F	.95
DC-8-61	DC-8-71	.92
B-747-200C	B-747-100	.92
L-1011-100	L-1011-500	.89
DC-8-61	DC-8-63	.88
B-747-100	B-747-200F	.86
DC-8-61	DC-8-73	.81

Table 2: *AIRCRAFT* types with $\overline{\gamma} > .75$

The binary algorithm clusters the two values with the highest $\overline{\gamma}$, C-130E and C-130H. Then, the next highest values, DC-8-33 and DC-8-50 are clustered, because those values have not already been clustered. The algorithm continues, clustering each pair of unclustered values until $\overline{\gamma}$ falls below the specified threshold value or all values are clustered. For example, B-747-200C is *not* clustered with B-747-100 because it has already been clustered with B-747-200F, which has a higher $\overline{\gamma}$ (.95 instead of .86). The algorithm creates a forest of binary clusters, shown in Figure 2.

We now substitute a joint value ($J_{i,j}$) for each clustered pair and repeat the algorithm. We generate new rules, recalculate $\overline{\gamma}$, and cluster the highest

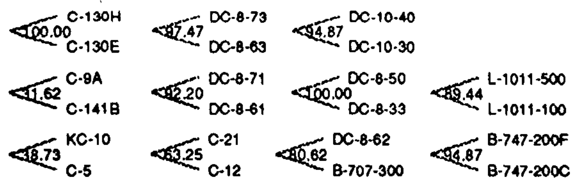

Figure 2: Clusters after one iteration

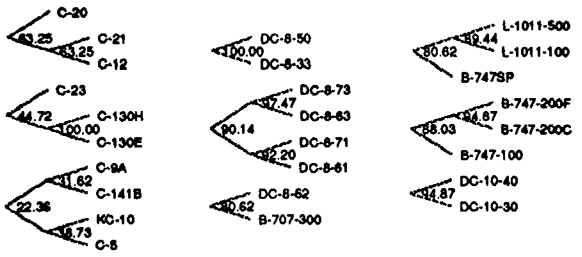

Figure 3: Clusters after two iterations

new values, creating Figure 3. The process repeats until the hierarchy is complete as shown in Figure 4.

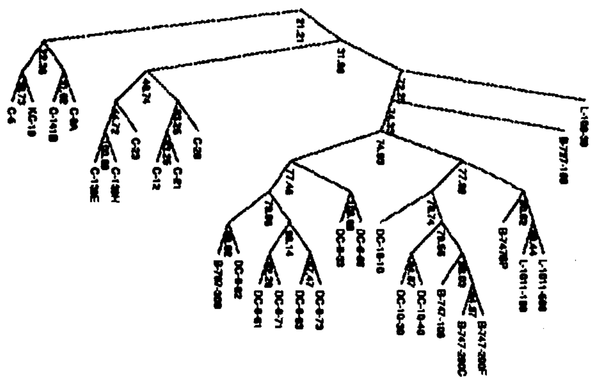

Figure 4: Final Binary AAH

The n-ary clustering algorithm is straightforward. Using an initial threshold of .80 yields the initial cluster shown in Figure 5. Note that a different threshold would yield a different initial cluster.

After each iteration, the threshold is reduced by a pre-specified amount to allow more clustering. The second layer, with a threshold of .75, is shown in Figure 6. Eventually, the threshold is low enough to include all values, yielding the n-ary cluster shown in Figure 1. Compare this with the binary cluster shown in Figure 4.

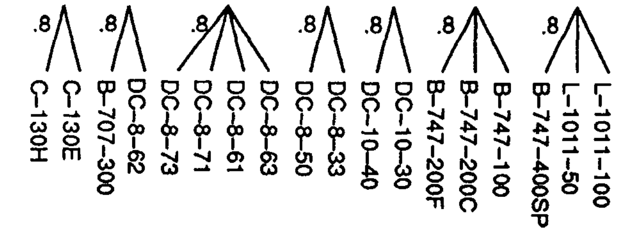

Figure 5: Clusters after one iteration (threshold = .80)

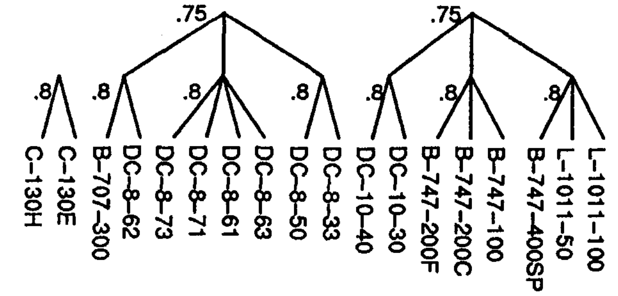

Figure 6: Clusters after second iteration (threshold = .75)

5 Experimental Results

PKI generates hierarchies for cooperative query answering with the correlation coefficients. As we go up the hierarchy, the correlation coefficients values are reduced. Thus, these numbers can be used for determining an appropriate cutoff point for terminating query relaxation. Further, since the correlation coefficients are normalized between zero and one, they can be combined from different hierarchies when more than one attribute is relaxed. This allows control of relaxation over several hierarchies simultaneously. The n-ary algorithm retains the feature that the correlations monotonically decrease as we go up the hierarchy.

The PKI clustering algorithm works well for discrete values, but is not as suitable for continuous domains, such as numeric values. Clustering of continuous numeric domains is more effectively handled by an algorithm which takes advantage of the ordering and range of the data, such as DISC [4]. These methods can be used to pre-cluster the numeric domain into ranges which can then be used as input to PKI, thus significantly improving the performance of the algorithm.

The PKI algorithm for rule detection is exponential on the number of attribute values. For the

			RUNTIME (mins.)	
RELATION	ATTRIBUTES	TUPLES	ORIGINAL	IMPROVED
CARGO_DETAILS	12	195598	280	55
GEOLOC	16	52686	144	43
CARGO_CATEGORIES	8	56416	65	27
UNIT_CHARACTERISTICS	23	12740	12	6
TP_MOVEMENT_REQS	84	8464	8	3
SHIPS	41	153	7	4
CHSTR_SHIP	16	806	4	2
FM_ELEMENTS	2	1228	4	2
ULNS	12	1232	3	2
ASSETS_SHIPDAYS	6	4130	3	2
...				
remainder (84 relations)	< 50	< 1000	< 1	< 1

Table 3: Relations in Transportation Database with a RUNTIME exceeding 1 minute

algorithm to be used in very large domains, pre-clustering of attributes, as described above, will decrease the number of values in numeric domains and thus reduce the number of atomic patterns and potential rules. Pre-clustering can also help in non-numeric domains, if the domain expert makes an initial pass through the database and selects values which are closely related. The domain expert can also be consulted to dictate which pairs of attributes are most likely to generate "useful" rules and further direct the PKI algorithm and reduce the computational complexity.

We have implemented PKI using C on a Sun SPARCstation IPC and used it to generate hierarchies for 94 different relations in the transportation domain. The relations have a variety of attributes including numbers, dates and text strings. The size of these relations range from 84 attributes and 8464 tuples to 12 attributes and 195,000 tuples. PKI takes less than a minute to generate hierarchies for all but the very largest relations. Table 3 shows the size of each relation (in attibutes and tuples), and the runtime required to generate hierarchies for all attributes. Of the 94 relations, only the ten requiring more than a minute to complete are listed.

In addition to using preclustering, efficiency can also be improved by eliminating rules with low popularity. By setting a popularity threshold, we require that all rules apply to several tuples. The running time of the algorithm is exponential on the number of rules, so increasing the popularity threshold from one to two improved the algorithm run-time by 80% for the largest relation. Table 3 shows the improvement in runtime if the popularity threshold is increased to two. However, if rules are eliminated, then the values in the premise of those rules can no longer be clustered. Thus, eliminating rules will yield hierarchies that only cluster values that appear more frequently in the database. These incomplete hierarchies still prove useful, as CoBase [3] can use partial hierarchies when full ones are not available. Since incomplete hierarchies include only values which appear frequently in the database, they require much less space than full hierarchies. Thus, they are appropriate for applications where a general overview of the clustering is needed, without all the specific details, as well as applications with strict time or space requirements. Also, setting a popularity threshold should not be used with key attributes, since a key implies that every value is distinct.

For very large databases, statistical sampling methods can be used for rule gathering. Instead of considering all tuples when evaluating a rule, a randomly selected sample of the tuples in the relation can be used to estimate the rule's confidence and popularity.

Both the n-ary and binary algorithms generate hierarchies with no overlapping clusterings. If the domain dictates that values may belong to more than one cluster, the selection criterion must be modified. Instead of taking the pairs with the highest $\overline{\gamma}$, all pairs with $\overline{\gamma}$ above the threshold are clustered.

The threshold value for n-ary clustering must be selected based on domain semantics. In the example, we selected an initial value of .8 and lowered the value by .05 each iteration until the hierarchy was complete. Choosing a higher initial threshold and smaller reduction per iteration will yield a hierarchy more similar to the original binary clustering.

6 Conclusions

In this paper, we have presented a Pattern-Based Knowledge Induction mechanism that generates rules relating attribute values in a relational database. The rules can then be used to cluster attribute values and construct binary AAHs. Such induction provides a correlation measure of nearness between the clustered attributes. Based on the initial binary clustering and specified threshold values, we can construct an n-ary attribute abstraction hierarchy suitable for use with cooperative query answering systems such as CoBase. The algorithm works well for discrete non-numeric domains and can use semantics and domain expert direction for pre-clustering to reduce computational complexity. The expert can also add weights to place emphasis on selected attributes, thus generating an AAH based on the context of a specific application.

References

[1] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI Press/The MIT Press, Menlo Park, CA, 1991.

[2] Q. Chen, W. W. Chu, and R. Lee. Pattern-based knowledge induction from databases. In *Database Systems for Next Generation Applications.* World Science Publishing Co., 1992.

[3] W. W. Chu, Q. Chen, and R. Lee. Cooperative query answering via type abstraction hierarchy. In S.M. Deen, editor, *Cooperating Knowledge Based Systems.* North-Holland, Elsevier Science Publishing Co., Inc., 1991.

[4] W. W. Chu and K. Chiang. A distribution sensitive clustering method for numerical values. Technical Report 93–0006, UCLA Computer Science Department, 1993.

[5] F. Cuppers and R. Demoloube. Cooperative answering: a methodology to provide intelligent access to databases. In *Proc. 2nd International Conference on Expert Database Systems*, Virginia, USA, 1988.

[6] J. Minker, G.A. Wilson, and B.H. Zimmerman. Query expansion by the addition of clustered terms for a document retrieval system. *Information Storage and Retrieval*, 8:329–348, 1972.

[7] J. R. Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 2. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[8] R. E. Stepp III and R. S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 2. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.