Abstraction of High Level Concepts from Numerical Values in Databases *

Wesley W. Chu and Kuorong Chiang †
Computer Science Department
University of California, Los Angeles

Abstract

A conceptual clustering method is proposed for discovering high level concepts of numerical attribute values from databases. The method considers both *frequency* and *value* distributions of data, thus is able to discover relevant concepts from numerical attributes. The discovered knowledge can be used for representing data semantically and for providing approximate answers when exact ones are not available.

Our knowledge discovery approach is to partition the data set of one or more attributes into clusters that minimize the *relaxation error*. An algorithm is developed which finds the best binary partition in $O(n)$ time and generates a concept hierarchy in $O(n^2)$ time where n is the number of distinct values of the attribute. The effectiveness of our clustering method is demonstrated by applying it to a large transportation database for approximate query answering.

Key words: approximate query answering, type abstraction hierarchy, conceptual clustering, discretization, data summarization, knowledge discovery in databases.

1 Introduction

Knowledge discovered from databases can be used to abstract the data into high level concepts. The discovered concept, or abstraction, usually implies a certain context, thus providing more information than the raw data. As a result, the abstraction can be used to characterize databases and process queries intelligently [6, 3]. In particular, abstraction can be used to derive approximate answers – when the objects requested by a query are not available, the query conditions can be relaxed to their corresponding abstraction where "neighborhood objects" can be found and returned as the approximate answers.

For clustering objects into "neighborhoods," two groups of methods can be used: statistical clustering and numerical taxonomy [12, 1, 7, 17], and conceptual clustering [15, 10, 14]. In statistical clustering and numerical taxonomy, most similarity metrics are defined between pairs of objects. Objects are pair-wise clustered in a bottom up manner until all objects are in a single cluster. In Conceptual clustering, the "goodness measures" are usually defined for the overall *partitioning* of objects instead of for pairs of objects. Clustering methods are designed that maximize the goodness measure. For approximate query answering [3, 5], the goal is to minimizes the *relaxation error* of the approximate answers where the relaxation error is defined based on

*This work is supported in part by DARPA contract N00174-91-C-0107

†The authors may be reached at {wwc,kuorong}@cs.ucla.edu

the overall partitioning of objects. This is closer to the similarity metrics in conceptual clustering methods.

Most current *conceptual clustering* systems use exact match when comparing two attribute values. That is, the values are categorical and can only be equal or not equal. These systems only consider *frequency distribution* of data because they are concerned with the *grouping* of values rather than the *values* themselves. Such clustering, however, is inadequate for providing approximate answers as illustrated in the following example.

Consider two clusters $C_1 = \{0, 100\}$ and $C_2 = \{0, 1\}$. C_1 and C_2 are equivalent based on frequency distribution alone because they both have two distinct values. For representing abstraction, however, they are very different because the values in C_2 are much closer to each other than those in C_1. That is, C_2 defines a much coherent abstraction than C_1 does. Therefore, for discovering abstraction, a clustering method must consider both the frequency and the *value* distributions of data. In this paper, we propose a DIstribution Sensitive Clustering (DISC) method that considers both the frequency and the value distributions in discovering high level concepts from data.

The rest of the paper is organized as follows. After a brief discussion of prior related work, we develop the notion of *relaxation error* as a quality measure for clusters. Then we present the DISC algorithm for generating concept hierarchies called Type Abstraction Hierarchies (TAH) for a single attribute as well as multiple attributes that minimize the relaxation error. Next, we present application of TAH for providing approximate query answers. Finally, we compare the performance of the TAH generated by DISC with the traditional index tree for approximate query answering.

2 Related Work

Prior work in discretization aims at decreasing cardinality of data by maximizing/minimizing certain heuristic measures. A commonly used measure is the information entropy [16]. It can be shown that the entropy is maximum when the data is partitioned most evenly. (We call this the ME method [2, 18].) However, no semantic meaning is attached to the resultant clusters because the discretization is concerned with the frequency distribution rather than the value distribution in the cluster. Therefore, the ME method is not suitable for abstracting numerical data.

COBWEB [8], a conceptual clustering system, uses *category utility* (CU) [9] as a quality measure to classify the objects described by a set of attributes into a classification tree. Formally, for a partition from a class C to m mutually exclusive classes $C_1, ..., C_m$, the category utility (CU) is defined as the increase in the average class *goodness*[1]. That is,

$$CU(C_1, ..., C_m) = \frac{\sum_{k=1}^{m} P(C_k)G(C_k) - G(C)}{m} \tag{1}$$

where $P(C_k)$ is the occurrence probability of C_k in C, and $G(C_k)$ and $G(C)$ are the goodness functions for C_k and C, respectively. That is,

$$G(C_k) = \sum_{a \in A} \sum_{x_i^a \in X_k^a} P(x_i^a)^2 \tag{2}$$

$$G(C) = \sum_{a \in A} \sum_{x_i^a \in X^a} P(x_i^a)^2 \tag{3}$$

where A is the set of all the attributes, and X_k^a and X^a are the distinct values of attribute a in C_k and C respectively.

[1] COBWEB does not use the term *goodness*. We use it for ease of presentation.

COBWEB cannot be used for abstracting numerical data because it only deals with categorical data. Moreover, its classification tree serves as *the database* for the instances and requires a large storage space. Furthermore, matching of objects with existing classes is time-consuming. For providing approximate answers, we want to build a classification tree *in addition to* the original database. The tree can be viewed as an index, thus making efficient storage and retrieval important.

3 Relaxation Error - A Goodness Measure for Clustering Numerical Values

To deal with numerical values, we need to generalize category utility. To simplify our presentation, let us consider the single-attribute case. For a class $C = \{x_1, ..., x_n\}$, (3) reduces to

$$G(C) = \sum_{i=1}^{n} P(x_i)^2 \tag{4}$$

where x_i is the i-th distinct attribute value and $P(x_i)$ is the occurring probability of x_i in C.

Consider the following experiment. The class C in (4) is represented by an urn with balls, each of them representing an attribute value. The number of balls equals the occurring frequency of the corresponding value. We randomly draw two balls from the urn with replacement. If the two balls have the same value, we score 1. Otherwise, we score 0. If we do this experiments a large number of times, the expected score we have will be $G(C)$. Let $s(x_i, x_j)$ be the score for the drawn values x_i and x_j, then $s(x_i, x_j) = 1$ if $x_i = x_j$ and 0 otherwise. Thus, (4) becomes

$$G(C) = \sum_{i=1}^{n} \sum_{j=1}^{n} P(x_i)P(x_j)s(x_i, x_j). \tag{5}$$

Equation (5) cannot serve as a quality measure for numerical values as illustrated in the following. Consider the two clusters $C_1 = \{0, 100\}$ and $C_2 = \{0, 1\}$ as mentioned in section 1. According to (5), C_1 is as good as C_2: $G(C_1) = G(C_2) = 0.5$. Intuitively, this is unsatisfying because the values in C_2 are much closer to each other than those in C_1. As a result, a label such as *small* can be attached to C_2 but not to C_1.

In order for (5) to be a goodness measure for numerical values, we need to change the scoring rule. Let $S(x_i, x_j) = 1 - \frac{|x_i - x_j|}{\Delta}$ where Δ is the maximum difference between two values.[2] $S(x_i, x_j) = 1$ if $x_i = x_j$. $S(x_i, x_j)$ decreases when $|x_i - x_j|$ increases. And $S(x_i, x_j) = 0$ when $|x_i - x_j| = \Delta$. Replacing $s(x_i, x_j)$ by $S(x_i, x_j)$ in (5), we have

$$G(C) = \sum_{i=1}^{n} \sum_{j=1}^{n} P(x_i)P(x_j)(1 - \frac{|x_i - x_j|}{\Delta})$$

After rearrangement by noticing $\sum_{i=1}^{n} \sum_{j=1}^{n} P(x_i)P(x_j) = 1$, we have

$$G(C) = 1 - \sum_{i=1}^{n} \sum_{j=1}^{n} P(x_i)P(x_j)\frac{|x_i - x_j|}{\Delta}. \tag{6}$$

[2] The same scoring rule is independently proposed in [11] for matching two numerical values.

Let us define the *relaxation error* of C, denoted by $RE(C)$, as the normalized expected difference between any two values in C. Formally,

$$RE(C) = \sum_{i=1}^{n} \sum_{j=1}^{n} P(x_i) P(x_j) \frac{|x_i - x_j|}{\Delta} \tag{7}$$

Thus,

$$G(C) = 1 - RE(C). \tag{8}$$

Using the new goodness measure for the above example where $\Delta = 100$, we have $G(C_1) = 0.5$ and $G(C_2) = 0.995$. Clearly, C_2 is better than C_1.

From the standpoint of query relaxation, let us define the *relaxation error of* x_i, $RE(x_i)$, as the average difference from x_i to x_j, j=1,...,n. Formally,

$$RE(x_i) = \sum_{j=1}^{n} P(x_j) \frac{|x_i - x_j|}{\Delta} \tag{9}$$

where $P(x_j)$ is the occurring probability of x_j in C. $RE(x_i)$ can be used to measure the quality of an approximate answer where x_i in a query is relaxed to x_j, j=1,...,n. Summing $RE(x_i)$ over all values x_i in C, we have

$$RE(C) = \sum_{i=1}^{n} P(x_i) RE(x_i). \tag{10}$$

Thus, $RE(C)$ is the expected error of relaxing any value in C.

To express the category utility in terms of the relaxation error, we substitute (8) into (1):

$$\begin{aligned} CU &= \frac{\sum_{k=1}^{m} P(C_k)[1 - RE(C_k)] - [1 - RE(C)]}{m} \\ &= \frac{RE(C) - \sum_{k=1}^{m} P(C_k) RE(C_k)}{m}. \end{aligned} \tag{11}$$

From (11) notice that $RE(C)$ is a constant for a given C, so CU is maximum (i.e., the best partition) when the term $\sum_{k=1}^{m} P(C_k)RE(C_k)$ (i.e., the average relaxation error of the partition from C to $C_1, ..., C_m$) is minimum. Thus in the next section, we shall develop the DISC algorithm that minimizes the relaxation error of the partition.

4 The DISC Algorithm

We shall now present the DISC algorithm for clustering numerical values that minimizes the relaxation error. We start from one cluster consisting of all the values of an attribute, and then we find "cuts"[3] to recursively partition the cluster. The partitioning is terminated when the number of distinct values in the resultant cluster is less than a pre-specified threshold.

Given a cluster with n distinct values, the number of partitions to examine is exponential with respect to n. To reduce computation complexity, we shall only develop binary cut algorithm which partitions a cluster into two sub-clusters. The clustering algorithm DISC is given in Table 1.

[3] A cut c is a value that separates a cluster of numbers $\{x | a \le x \le b\}$ into two sub-clusters $\{x | a \le x \le c\}$ and $\{x | c < x \le b\}$.

Algorithm DISC(C)

if the number of distinct values in $C < T$, return /* T is a threshold */
call **BinaryCut**(C) to find the best cut, *cut*, for C such that relaxation error is minimized
partition values in C based on *cut*
let the resultant sub-clusters be C_1 and C_2
  call DISC(C_1) and DISC(C_2)

Algorithm BinaryCut(C)
/* input cluster $C = \{x_1, ..., x_n\}$ */
for $h = 1$ to $n - 1$ /* evaluate each cut */
  Let P be the partition with clusters $C_1 = \{x_1, ..., x_h\}$ and $C_2 = \{x_{h+1}, ..., x_n\}$
  compute relaxation error $RE(P)$ for P
  if $RE(P) < MinRE$ then
    $MinRE = RE(P)$, $cut = h$ /* the best cut */
Return *cut* as the best cut

Table 1: The algorithms DISC and BinaryCut

4.1 Computation Complexity of DISC

Let us consider the time complexity of computing the relaxation error of a cluster. Let n be the number of distinct values in C, f_i be the frequency of the value x_i, and N be the total number of values in C which equals $\sum_{i=1}^{n} f_i$. Based on [13], equation (7) can be transformed to

$$RE(C) = \frac{2}{N^2} \sum_{h=1}^{n-1} F(h)(N - F(h))(x_{h+1} - x_h) \tag{12}$$

where $F(h)$ is the accumulated frequency $\sum_{i=1}^{h} f_i$. Note that in (12), both frequency and value distributions are considered – $RE(C)$ equals the sum over all gaps (the value distribution) weighted by the accumulated frequencies on both sides of each gap (the frequency distribution).

Using (12), $RE(C)$ can be computed in $O(n)$ time. Such computation is executed $n - 1$ times in the algorithm BinaryCut, thus the overall time complexity of BinaryCut is $O(n^2)$.

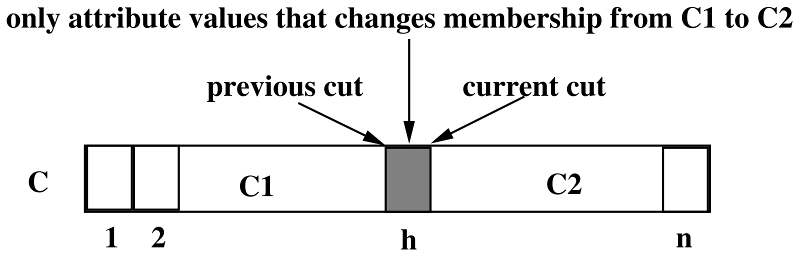

Figure 1: Membership change

Further improvement of BinaryCut is possible by the following observation. For each iteration in BinaryCut, only a single value changes its membership from one sub-cluster to another (see Figure 1). In general, for a cluster $C = \{x_u, ..., x_v\}$, only two cases of membership change can happen: (1) C is changed to $C' = \{x_u, ..., x_{v+1}\}$ where a new value x_{v+1} larger than any existing values is inserted and (2) C is changed to $C'' = (x_{u+1}, ..., x_v\}$ where an old value x_u smaller than

Algorithm BinaryCut(C)
/* The input is $C = \{x_1, ..., x_n\}$. $N = \sum_{i=1}^{n} f_i$. */
/* Initially, C_1 is empty and $C_2 = C$. Each iteration moves x_h from C_2 to C_1 */
Initialization:
$T = N_1 = S = 0, N_2 = N, MinRE = BiggestNumber$
$W = \sum_{h=1}^{n-1}[N - F(h)](x_{h+1} - x_h), S' = \sum_{h=1}^{n-1} F(h)[N - F(h)](x_{h+1} - x_h)$
for h=1 to $n - 1$ do /* evaluate each possible cut h */
$T = T + N_1(x_h - x_{h-1})$ /* x_h changes membership from C_2 to C_1 */
$S = S + f_h T$
$N_1 = N_1 + f_h$
$RE(C_1) = 2S/N_1^2$
$S' = S' - f_h W$
$N_2 = N_2 - f_h$
$W = W - N_2(x_{h+1} - x_h)$
$RE(C_2) = 2S'/N_2^2$
$AveRE = \frac{N_1}{N} RE(C_1) + \frac{N_2}{N} RE(C_2)$
if $AveRE < MinRE$ then
$MinRE = AveRE$, $cut = h$ /* remember the best cut */
Return cut as the best cut.

Table 2: Improved BinaryCut algorithm

any existing values is deleted. Since the modification to the partition resulting from these changes is only limited to one member, we shall derive equations for computing $RE(C')$ and $RE(C'')$ recursively from $RE(C)$.

Based on equation (12), for the cluster C', we have

$$RE(C') = \frac{2}{[N + f_{v+1}]^2} \sum_{h=u}^{v} F(h)[N + f_{v+1} - F(h)](x_{h+1} - x_h).$$

where $N = \sum_{h=u}^{v} f_h$. After manipulation, we have

$$RE(C') = \frac{2}{[N + f_{v+1}]^2}\{S(v) + f_{v+1}[T(v) + N(x_{v+1} - x_v)]\} \tag{13}$$

where $S(v) = \sum_{h=u}^{v-1} F(h)[N - F(h)](x_{h+1} - x_h)$ and $T(v) = \sum_{h=u}^{v-1} F(h)(x_{h+1} - x_h)$. Note that (13) can be computed in constant time for a given $S(v)$ and $T(v)$. From the definitions of $S(v)$ and $T(v)$, we observe

$$T(v + 1) = T(v) + N(x_{v+1} - x_v) \tag{14}$$

$$S(v + 1) = S(v) + f_{v+1} T(v + 1) \tag{15}$$

Substituting (14) and (15) into (13), we have $RE(C') = 2S(v + 1)/[N + f_{v+1}]^2$.

Similar to (13), we obtain

$$RE(C'') = \frac{2}{[N - f_u]^2}[S'(u) - f_u W(u)] \tag{16}$$

where $S'(u) = \sum_{h=u}^{v-1} F(h)[N - F(h)](x_{h+1} - x_h)$ and $W(u) = \sum_{h=u}^{v-1}[N - F(h)](x_{h+1} - x_h)$. Similar to (14) and (15), we have

$$S'(u+1) = S'(u) - f_u W(u) \tag{17}$$

$$W(u+1) = W(u) - (N - f_u)(x_{u+1} - x_u) \tag{18}$$

And $RE(C'') = 2S'(u+1)/[N - f_u]^2$.

Based on equations (14), (15), (17), and (18), the improved algorithm for BinaryCut is shown in Table 2.

For each iteration in the improved BinaryCut, we can now calculate the relaxation error of the partition in constant time. Since there are $n-1$ iterations, the loop takes $O(n)$ time to complete. Because initializing the variables W and S' also takes $O(n)$ time, the time complexity of BinaryCut is therefore linear. Since the algorithm DISC calls BinaryCut at most $n-1$ times, the worst case time complexity of DISC is $O(n^2)$.

5 Application to Approximate Query Answering

We shall now use an example to show how the concept hierarchies generated by DISC, called the Type Abstraction Hierarchy (TAH), can be used for providing approximate answers.

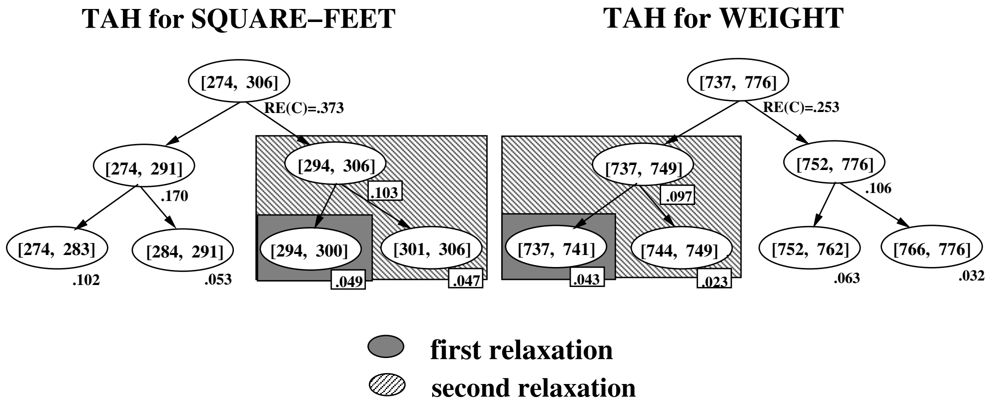

Figure 2: TAHs for SQUARE-FEET and WEIGHT generated by DISC from a transportation database. Higher nodes have higher relaxation errors.

Example. Consider the query "find a cargo with size 300 square feet and weight 740 kg."

The corresponding SQL query is

select CARGO-ID
from CARGOS
where SQUARE-FEET = 300 and WEIGHT = 740.

The query conditions are too specific; no answers are found for the query. To obtain approximate answers, the query is modified by relaxing cargo size and weight according to the TAHs for SQUARE-FEET and WEIGHT (see Figure 2). (These TAHs are generated by DISC from a transportation database for the CARGOS relation.) As a result, the query is modified to

select CARGO-ID
from CARGOS
where $294 \leq$ SQUARE-FEET ≤ 300 and $737 \leq$ WEIGHT ≤ 741.

The modified query is submitted to the database and the following cargo is returned:

CARGO-ID	SQUARE-FEET	WEIGHT
10	296	740

The quality of the answer can be measured by equation (9). From the CARGOS relation, we obtain Δ(SQUARE-FEET) = 32 and Δ(WEIGHT) = 39. Thus, RE(SQUARE-FEET = 300) = 0.125 and RE(WEIGHT = 740) = 0. Assuming the two attributes SQUARE-FEET and WEIGHT are equally important, then the quality of the answer can be measure by the sum of individual relaxation errors, which equals 0.125.

If more answers are needed, the conditions can be further relaxed by moving one more step up the TAHs:

> **select** CARGO-ID
> **from** CARGOS
> **where** 294 $\leq$ SQUARE-FEET $\leq$ 306 and 737 $\leq$ WEIGHT $\leq$ 749.

The following four cargos are returned for this query:

CARGO-ID	SQUARE-FEET	WEIGHT
10	296	740
21	301	737
30	304	746
44	306	745

For the above answer, RE(SQUARE-FEET = 300) = 0.117 and RE(WEIGHT = 740) = 0.090 with a sum of 0.207. This answer yields more relaxation error than the previous one because the values deviate more from those of the requested cargo.

Thus from the above example, we see that the TAHs generated by DISC can be used for relaxing query conditions to obtain "neighborhood answers" and their quality measure when the requested answer is not available.

6 Multi-dimensional Type Abstraction Hierarchy (MTAH)

In many applications, abstraction needs to be characterized by multiple attributes, e.g., *nearness* of geographical locations. Given two locations represented by longitudes and latitudes, they are near to each other only if their longitudes *and* latitudes are close to each other. To cluster objects of multiple attributes, DISC can be extended to M-DISC (Table 3). The generated multi-dimensional TAHs are called MTAHs. The algorithm DISC is a special case of M-DISC and TAH is a special case of MTAH.

MTAHs can be used to modify queries for providing approximate answers as shown in section 5. They can also be used as "semantic indices" where each internal node of an MTAH stores an attribute name and a *cut* value, and the leaf nodes contain pointers to physical data blocks. To locate an object, the query conditions are compared to the *cuts* of the nodes: if they are less than or equal to the *cut*, then the comparison continues to the left child, otherwise it continues to the right. When a leaf node is reached, the stored pointer is used to retrieve the object. If the object is not available in the database, a traversal up the MTAH enlarges the "neighborhood" of the requested object. The best n objects covered in this neighborhood can then be returned as the approximate answers where n is a threshold.

Algorithm M-DISC(C)

if the number of values in $C < T$, return /* T is a threshold */
for each dimension d
    call BinaryCut(C) to find the best cut h
    compute relaxation error reduction ΔRE based on h
    if $\Delta RE > MaxReduction$ then /* remember the best cut */
        $MaxReduction = \Delta RE$, $BestDimension = d$, $cut = h$
partition C based on cut of the dimension $BestDimension$
let the resultant sub-clusters be C_1 and C_2
    call M-DISC(C_1) and M-DISC(C_2)

Table 3: Multi-dimensional DISC algorithm

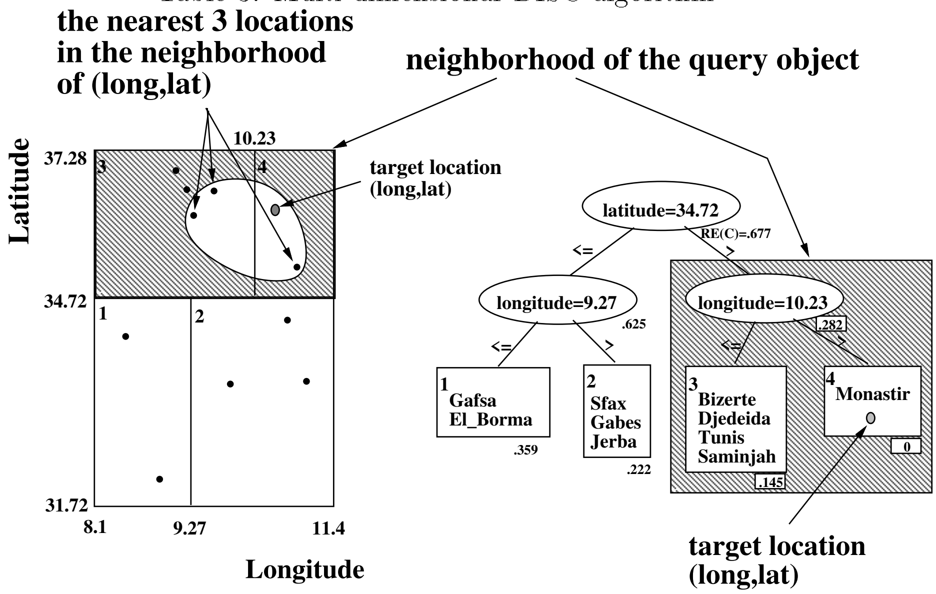

Figure 3: A map of cities in Tunisia and its corresponding MTAH generated by M-DISC

For example, Figure 3 shows a simplified map of Tunisia and the corresponding MTAH generated by M-DISC. The MTAH can provide the neighborhood region of the target location (*long,lat*). In this case, Monastir is the city in the smallest neighborhood ($10.23 < long < 11.4, 34.72 < lat < 37.28$) of the target location. The next larger neighborhood ($8.1 < long < 11.4, 34.72 < lat < 37.28$) includes 4 more cities: Bizerte, Djedeida, Tunis, and Saminjah (shown in the shaded region in Figure 3).

Also shown in Figure 3 are the relaxation errors at each node of the MTAH, $RE(C)$, which are computed from equation (7). Note that the relaxation error of a region increases as the dispersion of objects in that region increases. For example, the cities in region 1 ($RE(C_1) = .359$) are more dispersed than those in region 3 ($RE(C_3) = .145$).

7 Performance Comparison of DISC with ME

7.1 Single Attribute

Empirical results based on a large transportation database show that clusters discovered by DISC have less relaxation error than those by the Maximum Entropy method (ME). It can be shown that only when the data distribution is symmetrical at the median, then the ME method and DISC perform equally well:

Theorem. Let D and M be the optimal binary cuts by DISC and ME respectively. If the distribution of data is symmetrical at the median, then D = M (i.e., the cuts determined by DISC and ME are the same).

Outline of the proof. The best binary cut determined by the ME method is at the median of the distribution since the resulting partition is the most even. Since the category utility CU is a function of the cut, the best cut c determined by DISC can be found by solving the equation $CU'(c) = 0$. For symmetrical distributions, the best cut is at the median. Therefore, ME and DISC find the same binary cut. (For a formal proof, the interested readers shall refer to [4].)

For skewed distributions, which holds for most data in the transportation database, DISC performs better than ME. Empirical results show that the performance improvement of DISC over ME increases as the skewness increases [4].

7.2 Multiple Attributes

In addition to the relaxation error, we shall introduce two additional performance measures, *accuracy* of the answer and *efficiency* of the retrieval[4], to compare the performance of DISC with ME:

$$\text{accuracy of the answer} = \frac{\text{retrieved relevant answers}}{\text{all relevant answers}}$$

$$\text{efficiency of the retrieval} = \frac{\text{retrieved relevant answers}}{\text{all retrieved answers}}$$

where "all relevant answers" are the best n answers determined by exhaustive search. In general, there is a trade-off between the two measures: the higher the accuracy of the answer, the lower the efficiency of the retrieval.

We generate an MTAH and an ME tree[5] based on attributes Longitudes and Latitudes of 972 geographical locations from a transportation database. We generate 500 queries with the form: "find the n locations nearest to $(long, lat)$" where n is randomly selected from 1 to 20, and $long$ and lat are generated based on the distributions of Longitudes and Latitudes from the location relation in the database. To answer the query using the MTAH or the ME tree, we first locate the most specific node in the tree that "covers" n' locations where $n' \geq n$. Then we compute the n' distances to $(long, lat)$ and use them to select the nearest n locations (see Figure 3). The accuracy, efficiency, and the average distance from the best n answers to the target location $(long, lat)$, averaged over the 500 queries, are shown in Table 4. Notice that the answers provided by the MTAH not only are closer to the target location, but also are more accurate and efficient than those of the ME tree. Further, the MTAH is far more efficient than the exhaustive search, yet provides answers very close to those generated by the exhaustive search.

[4] These measures are known as *recall* and *precision* in Information Retrieval.

[5] Since the classification tree generated by the ME method is balanced, it can be viewed as an efficient index.

	MTAH	ME-Tree	Exhaustive Search
accuracy	0.7	0.45	1.0
efficiency	0.41	0.29	0.011
ave distance (miles)	55.2	70.4	49

Table 4: Comparison of the MTAH and the ME tree

8 Conclusion

In this paper, we present a DIstribution Sensitive Clustering method (DISC) for numerical attribute values. For an attribute of n distinct values, DISC finds the best binary partition in $O(n)$ time and generates a Type Abstraction Hierarchy (TAH) in $O(n^2)$ time. To demonstrate the efficiency of DISC, we have used DISC to generate TAHs for a large transportation database which consists of 94 relations, the largest one of which has 12 attributes and 195,598 tuples. DISC generates all the numerical TAHs in less than one hour of processing time on a Sun Sparc 10 Workstation.

The generated TAHs are used in the *Cooperative Database System* (CoBase) [5] at UCLA for providing approximate answers. The approximate query answers derived from TAHs are empirically evaluated in terms of accuracy, efficiency, and relaxation error. We show that the approximate query answers derived from the TAH generated by DISC are better than those derived from an index tree generated by the Maxmum Entropy method.

References

[1] M. R. Anderberg. *Cluster Analysis for applications.* Academic Press, New York, 1973.

[2] David K. Y. Chiu, Andrew K. C. Wong, and Benny Cheung. Information discovery through hierarchical maximum entropy discretization and synthesis. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI Press/The MIT Press, 1991.

[3] Wesley W. Chu and Q. Chen. Neighborhood and associative query answering. *Journal of Intelligent Information Systems*, 1(3/4), 1992.

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

[5] Wesley W. Chu, M. A. Merzbacher, and L. Berkovich. The design and implementation of CoBase. In *Proceedings of ACM SIGMOD*, Washington D. C., USA, May 1993.

[6] F. Cuppens and R. Demoloube. Cooperative answering: a methodology to provide intelligent access to databases. In *Proceedings of the 2th International Conference on Expert Database Systems*, Virginia, USA, 1988.

[7] B. Everitt. *Cluster Analysis.* Heinemann Educational Books, London, 1980.

[8] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[9] M. A. Gluck and J. E. Corter. Information, uncertainty, and the unity of categories. In *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, pages 283–287, Irvine, CA, 1985.

[10] Stephen Jose Hanson and Malcolm Bauer. Conceptual clustering, categorization, and polymorphy. *Machine Learning*, 3:343–372, 1989.

[11] Yannis E. Ioannidis, Tomas Saulys, and Andrew J. Whitsitt. Conceptual learning in database design. *ACM Transactions on Information Systems*, 10(3):265–293, 1992.

[12] A. K. Jain and R. C. Dubes. *Algorithms for Cluster Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[13] Maurice G. Kendall and Alan Stuart. *The Advanced Theory of Statistics*, volume 1. Hafner Publishing Company, 1969.

[14] M. Lebowitz. Experiments with incremental conceptual formation. *Machine Learning*, 2(2):103–138, 1987.

[15] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 1. Margan Kaufmann Publishers, Inc., 1983.

[16] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Ill, 1964.

[17] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. W. H. Freeman and Company, San Francisco, 1973.

[18] Andrew K. C. Wong and David K. Y. Chiu. Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):796–805, 1987.