Adaptive Indexing in Very Large Databases

Andrew Johnson, Farshad Fotouhi

Wayne State University

ABSTRACT: Very large databases contain large amounts of interrelated information. This information is often stored in relational databases with hundreds of tables and thousands of rows per table. Clustering is an effective way to reduce the information-overhead associated with finding information among these tables, allowing the user to browse through the clusters as well as the individual tables. In this paper, we compare the use of two adaptive algorithms (genetic algorithms, and neural networks) in clustering the tables of a very large database. These clusters allow the user to index into this overwhelming number of tables and find the needed information quickly. We cluster the tables based on the user's queries and not on the content of the tables, thus the clustering reflects the unique relationships each user sees among the tables. The original database remains untouched, however each user will now have a personalized index into this database.

1 Introduction

Relational databases are designed to deal with limited ranges of data on specific topics. The form of the data is known ahead of time and the database tables and their relationships are clearly defined before the data is entered. Very large databases can contain a much larger amount of data on many different, but related topics. The form of the data is often not known ahead of time as the data is collected from a wide range of sources.

Very large databases are used in many areas. They store statistics such as economic data and census data. They store inventory data. They store data from scientific experiments and simulations such as climate modelling and human genome mapping. They are accessed by users from a wide range of disciplines, mostly unfamiliar with databases and their associated query languages. These users need to search for specific pieces of data quickly, and browse through related information to see if it is of value to them. They need to relate information from different tables in the database.

Much of the difficulty in accessing data in very large databases comes from the enormous amount of data that is involved; but the organization of this data is also a major problem (French et al., 1990.) Users from various communities see different relationships between sets of data. Certain information is important to certain users and certain information is not. As time passes, the data in the database will be important to different users. Typically a generic interface is provided to the data. This gives users from all backgrounds a way to access the data, but each of the users must conform to this generic structuring of the data.

When an interface to a very large database is designed, its creator typically imposes a generic structure on the database - a hierarchical menu system allowing the user to move through an ordering of the tables. This gives users from all backgrounds a way to access the data, but each of the users must conform to this generic structuring of the data. This approach has several shortcomings: 1) The menu system does not provide enough flexibility for a wide range of researchers, 2) The users may not know enough about the domain to make appropriate choices, 3) It does not help the user with ill-defined queries.

Graphical query languages have been proposed to simplify the interface (Kuntz and Melchert, 1989, Ozsoyoglu and Wang, 1993.) Graphical query languages make the database schema more visible, reduce typing,

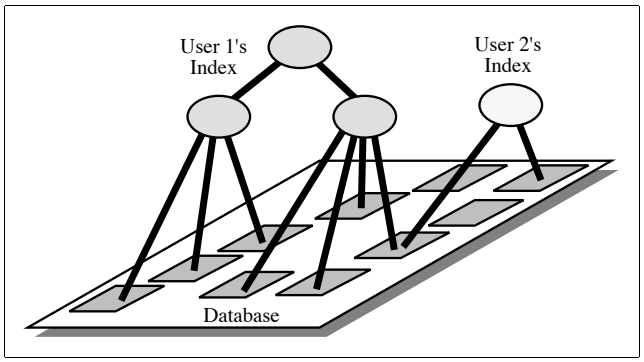

Figure 1: Hierarchical Index into the Database

and allow users to rely on recognition rather than memorization. This approach has several shortcomings: 1) The schema of large databases are so large, and complicated that the user rapidly runs out of screen real-estate, 2) The graphical metaphor quickly becomes cumbersome for complicated queries.

Ioannidis, et all (Ioannidis et al., 1992, Ioannidis et al., 1993) developed a graphical interface for the management of scientific experiments and data using the Object-Oriented data model MOOSE. The user interacts with the database through the schema. The system makes large schemas more manageable by allowing the user to hide parts of the schema, collapse sections of the schema into nodes, and use reference nodes to eliminate long arcs. While useful for people involved in the original experiment, this approach has several shortcomings for users less familiar with the original experiment: 1) The users may not know what data is available, 2) The users may not know enough about the domain to make appropriate choices, 3) It give users a variety of choices without sufficient descriptive material to make that choice, 4) The original schema may not match the relationships seen by all users.

Instead of forcing the users to conform to the structure of the database, we can mold the structure of the database to the needs of the individual users and thereby reduce their confusion when interacting with the database. Our solution is to cluster the tables of a very large database based on the user's queries. The user of the database needs the benefits of clustering so we should involve the user in the clustering process. The content of the tables is irrelevant, only the fact that the user sees a relationship between the tables is important. This flexibility will be increasingly important as the size, breadth, and accessibility of very large databases increases.

We collect data on the user's queries. This information is then given to a genetic algorithm or to a neural network which partitions the tables of the database into a hierarchy of clusters. The genetic algorithm and neural network generate very good clusterings much faster than a deterministic exhaustive search algorithm. Each user now has the choice of browsing through the existing generic interface, or using a hierarchy of clusters as an index to move quickly to the appropriate tables in the database. The original database remains untouched, however each user now has a personal index into this document. See Figure 1.

Clustering has been found to be a very effective means of reducing information overhead in other large information systems with related components such as hypertext systems Botafogo and Schneiderman, 1991, Hara et al., 1991.) Genetic algorithms have been found to be useful in improving document retrieval accuracy by integrating usage patterns from various users (Gordon, 1988.)

As new users begin to work with this database, they can choose from the existing clusterings. A new user can choose to look at the database from various points of view. This gives each user a starting point

nearer their own needs than the generic interface. Experienced users will also be able to access their data faster and more conveniently because the database will adapt to them. We have tested this approach using users from several disciplines who found it to be useful in accessing a very large database.

Section 2 discusses how we collect information from the user. Section 3 discusses clustering in very large databases. Sections 4 and 5 respectively discuss how we use a genetic algorithm and neural network to create the hierarchy of clusters. Section 6 discusses our implementation of both clustering methods on a very large database. Section 7 discusses uses for this clustering. Finally, Section 8 gives our conclusions and plans for future work.

2 Information Collection

Users retrieve data from a relational database using a query language such as SQL. We monitor the user's queries, and store information about the tables accessed in a list.

Each line of the list contains information on a single table in the database. This line lists all of the columns that have been displayed for that table, and all the columns that have been used to connect this table to other tables in the database. The format of each line of the list is shown below:

table_i : {selectCol_i,}*:{joinCol_i-joinCol_j@table_j,}*.

where:

- table_i is the table we are interested in.
- selectCol_is are the columns in table_i that have been selected.
- joinCol_is are the columns in table_i that have been joined with columns joinColumn_j of table table_j

Three small sample SQL queries are converted into their list form as shown in Figure 2.

When a user begins to search for information in the database they either choose to work with an existing list or create a new empty list. The user can give this new empty list an appropriate name. As the user works with the database, the list keeps track of the tables and columns the user is accessing through their queries.

The mechanism for creating the query (straight query commands, a graphical query language, etc.) is unimportant. Eventually the query in the form of tables and columns is given to the database and this query updates the listing.

A user can create one list for all of their queries, or create multiple lists where each concept the user is interested in is given its own list. A user can keep their lists private or share them with other users of the database. The lists can be unioned together. A list can therefore be one of a user's lists, the union of a user's lists, the union of a group's lists, the union of a site's lists, or the union of all the lists available on the system.

We have used a similar technique to collect information on a user's browsing through a hypertext (Nielsen, 1990a) document to cluster the nodes based on the user's usage patterns (Johnson and Fotouhi, 1993.) The scientific or statistical database user, like a hypertext user, is faced with an overwhelming amount of interrelated information presented in a generic way. Clustering allows us to add a personal interface layer on top of the generic interface reducing information overhead and speeding up access to the underlying data.

Query 1	List 1	
SELECT $A.\phi$, $B.\chi$, $C.\epsilon$ FROM A, B, C WHERE $A.\alpha = B.\beta$ AND $A.\alpha = C.\gamma$;	A:ϕ, B:χ, C:ϵ,	:α-β@B, α-γ@C,. :β-α@A,. :γ-α@A,.
Query 2	List 2	
SELECT $C.\rho$, $C.\omega$, $D.\lambda$ FROM C, D WHERE $C.\rho = D.\sigma$;	C:ρ, ω, D:λ,	:ρ-σ@D,. :σ-ρ@C,.
Query 3	List 3	
SELECT $A.\alpha$ FROM A, B, C WHERE $A.\delta = B.\beta$ AND $B.\beta = C.\rho$;	A:α, B: C:	:δ-β@B,. :β-δ@A, β-ρ@C,. :ρ-β@B,.

Figure 2: Converting Queries into Lists

3 Clustering

As databases grow larger it becomes more and more important to cluster their tables. Clustering reduces information overhead. It allows for higher level concepts (groups of tables), allows for the breaking of a single large database into appropriate modules, allows for views over the database, and allows the user to make changes at the cluster level without affecting the database itself.

Unlike databases where the relationships between the tables are defined before any data is inserted, the tables of statistical and scientific databases are often created by different individuals. Few relationships are defined beforehand so there is little overall structure to the database. Each user imposes their own structure on the database through their queries. A table in a statistical or scientific database is often the results from a single experiment, or information collected over a certain period of time, or over a certain area.

By monitoring the user's progress through the database we can cluster the tables of the database based on the connections the user makes between the tables. Those tables that the user has found important are made more visible, and more easily accessible. In commonly used databases this will give the user quick access to important blocks of information, and several starting points for starting new searches. The clusters can be shared between users. This gives new users a choice of several ways to view an unfamiliar database, and gives regular users a way to see the database from different perspectives.

We use a genetic algorithm, and a neural network to delineate the clusters. Finding the optimal clustering of the tables is equivalent to checking every partition of the set of tables. This is an exponential problem so it is too costly to find the optimal clustering for a database with more than a few tables. Using an adaptive algorithm such as genetic algorithms or neural networks we can find a very good clustering within a reasonable amount of time.

The lists that have been created by the information gathering process described in Section 2 are used to cluster the tables of the database. The number of times each table is used in each list, divided by the total over all the lists is used to generate usage percentages. For each table, every joined table in each of the lists is added onto the roster of neighbouring tables.

Table	List 1 Percent	List 2 Percent	List 3 Percent	Neighbouring Tables
A	50	0	50	BC
B	50	0	50	AC
C	33	33	33	ABD
D	0	100	0	C

Figure 3: Converting lists into Percentages

Table	List 1 Percent	List 2 Percent	List 3 Percent	Neighbouring Tables
A	63	0	63	BC
B	63	0	63	AC
C	50	50	50	ABD
D	0	100	0	C

Figure 4: Final Inputs to the Clustering Algorithms

The converted versions of the lists created in Figure 2 are shown in Figure 3. Since Table A was used once in List 1, and once in List 3, Lists 1 and 3 have usage percentages of 50% while List 2 has a usage percentage of 0%. Table A was joined with tables B and C in the queries so tables B, and C are neighbours of table A. As the number of lists increases, the usage percentages decrease towards zero for the commonly used tables. This decreases the effectiveness of the clustering algorithms as the amount of difference in the usage patterns decreases. To avoid this pitfall, the percentages are scaled so that the average usage percentage is set to 50% rather than (100/number of lists)%. The inputs to the two clustering algorithms are shown in Figure 4. The usage percentages and the neighbour roster are used to cluster the tables.

We have used a similar technique to cluster the tables of a scientific database in (Johnson et al., 1994) and these techniques are further elaborated upon in (Johnson, 1994.)

4 Genetic Algorithm Clustering

Genetic algorithms accept their input coded as a finite length string (or chromosome) over a finite alphabet (the set of allele values.) Each of the elements in the string is a gene, and each gene has an allele value. Several of these chromosomes form a population. A payoff function determines the fitness of each chromosome based on its allele values.

The population changes from one generation to the next through reproduction, crossover, and mutation. The more fit a chromosome is, the more likely it is to reproduce and survive until the next generation. The number of chromosomes in each iteration remains constant, but their fitness should generally improve.

Each of the chromosomes in the population is initially given random allele values. The more fit chromosomes of the previous generation reproduce to produce the chromosomes of the next generation. Sometimes when two chromosomes reproduce, they crossover. When chromosomes W and X crossover to form chromosomes Y and Z, a random crossover percentage is generated. That percentage of chromosome W's genes are copied into identical positions in Y. The remaining genes in chromosome W are copied into Z. Genes from chromosome X fill in the remaining positions in Y and Z. The result of the crossover may be a more fit chromosome, or a less fit chromosome.

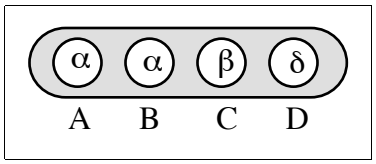

Figure 5: Sample Chromosome

Crossover only mixes existing allele values at each gene. Mutation causes random changes in the allele values, allowing for more (uncontrolled) variation. The result of the mutation may be a more fit chromosome, or a less fit chromosome. For a more thorough introduction to genetic algorithms see (Goldberg, 1989.)

In our approach, we use a population of 10 chromosomes. Each chromosome represents a set of clusters; each gene represents a table in the database; each allele value represents a cluster. Genes with the same allele value are in the same cluster. Each table is a member of at most one cluster, and the number of clusters varies from 1 to n where n is the total number of tables being clustered. Each table can form its own unique cluster in the chromosome, if necessary.

Figure 5 shows a sample chromosome that would be used to cluster the data in Figure 2. The four genes (A, B, C, D) in the chromosome represent each of the four tables involved in the clustering. Tables A and B are in the same cluster as they both have an allele value of α. Table C forms its own cluster with an allele value of β, and table D forms its own cluster with an allele value of δ. As there are four tables (A, B, C, D), there are four possible clusters (α, β, γ, δ.)

The payoff function is composed of two separate partial payoff functions: weight similarity, and neighbourlyness. Each partial payoff function's values range from 0.0 to 10.0.

Weight similarity, computed by Equation 1, promotes clusters containing tables with similar weight percentages, indicating similar usage patterns. For every cluster, the difference between each table percentage and the average percentage of the cluster is computed and scaled into the range 0 to 10. The closer the percentages are, the smaller this ratio will be, and the larger the overall payoff will be. Weight similarity pushes the genetic algorithm to create smaller clusters.

$$10 \times (1 - min(\frac{\sum_{tables} \sum_{lists} \mid table\% - aveCluster\% \mid}{\#\ of\ tables}, 1)) \tag{1}$$

Neighbourlyness, computed by Equation 2, promotes clusters containing neighbouring tables. The roster of neighbours shows which tables have been joined. For each pair of tables that are connected by a join, at least one user must feel these two tables are related because that user joined them. Each connection in the input lists either connects two tables in the same cluster, or two tables in different clusters. Neighbourlyness sums up the number of connections between tables in the same cluster and divides the total by the total number of connections in the lists. Neighbourlyness pushes the genetic algorithm to create bigger clusters.

$$10 \times \frac{\sum_{clusters} \#\ of\ joins\ between\ tables\ in\ cluster}{\#\ of\ joins} \tag{2}$$

Neighbourlyness tries to bring all the connected tables together. Weight similarity tries to isolate tables with similar usage patterns. Together, this pulling together and pushing apart generates the clusters. Equation 3 shows how the two partial payoff functions are combined.

$$payoff = neighbourlyness \times weightSimilarity^2 \tag{3}$$

This results in an overall payoff function value that ranges from 0 to 1000.

Each table is initially put into its own cluster. The genetic algorithm is started. After each new generation is created the most fit chromosome of the previous generation replaces the least fit chromosome of the new generation. This preserves the best set of clusters from one generation to the next. The genetic algorithm stops when the value of the most fit chromosome remains constant for 50 generations, or the number of generations reaches a set maximum value.

If there is more than one resulting cluster, these clusters are given back to the genetic algorithm. The weight percentages, and neighbours of the components of each cluster are used to generate the weight percentage, and neighbours for the cluster. Each cluster is now treated like a table. The clusters are clustered forming a hierarchy until one global cluster remains.

5 Neural Network Clustering

A neural network is a distributed network of small processing elements. This large number of simple processing units is connected together forming a miniature version of the human brain. Neural networks have had a good deal of success in pattern recognition and learning by example.

We are using a self organizing feature map, a neural network developed by Teuvo Kohonen (Kohonen, 1990.) A self organizing feature map is a two dimensional array of processing elements. Each element contains a weight vector. The input to the self organizing feature map is a set of weight vectors. The self organizing feature map forms itself into a topological ordering of the input data through unsupervised learning.

The weight vector W_{ij} of each element of the N by N array is initialized with random values. An input vector I is chosen at random. The neuron whose weight vector W_{ij} is closest to I is found ($min \mid W_{ij} - I \mid$). This neuron and all the neurons within a certain neighbourhood (N_c by N_c) of this neuron have their weights adjusted, bringing the weights closer to I. Random inputs are repeatedly provided while the amount of adjustment (α), and the neighbourhood shrink in a form of simulated annealing. The amount of adjustment and the size of the neighbourhood eventually become negligible and the learning ceases.

When the learning stops, the network has organized the inputs into a two dimensional array. For each input there is one processing element with the closest matching weight vector. Labeling each neuron with the table that it matches most closely gives the topological ordering of the inputs. For a more thorough introduction to neural networks see (Nielsen, 1990b.)

In our clustering algorithm the weight vectors contain the usage percentages for the various usage lists being used to cluster the database. If there are k lists involved in the clustering, the weight vector will contain k values. The size of the array N = $\lceil \sqrt{number\ of\ tables} \rceil$. Each table can form its own unique cluster in the array, if necessary.

If we used a self organizing feature map to cluster the data in Figure 2, we would use a 2 by 2 array. Each element of the array would be a weight vector of length 3 (one for each list.) The set of input vectors would consist of four weight vectors (one for each table) of length three (0.63, 0.0, 0.63), (0.63, 0.0, 0.63), (0.5, 0.5, 0.5), (0.0, 1.0, 0.0).

The amount of adjustment, and the neighbourhood are adjusted as follows:

$$\alpha_0 = 1$$

$$\alpha_{t+1} = \begin{cases} \alpha_t - 0.001 & \text{if } \alpha_t \geq 0.7 \\ \alpha_t - 0.0001 & \text{otherwise} \end{cases}$$

Abbreviated Table Name	Table Description
AEROLOG	measurements from surface flux group
AIR_FLUX	aircraft flux from flights over the konza
AMS_87	ave from ncar's pams, army corp's dcps
AMS_89	ave from ncar's pams, army corp's dcps
AMS_STAT	# of reports from each ams station
BIOMASS	plant biomass weight, nitrogen content
BRUT	actual radiosonde data observations
CLOUD	cloud estimates from liverpool cameras
FIFE_SITE	reference info. on the collection sites
GRAV	soil moisture readings at 25,75,150mm
NEUTRON	soil moisture with 200cm neutron probe
RAD_FLUX	measurements from surface flux group
RAIN_DAY	daily rainfall data by site and date
SOIL_PROP	soil properties measured historically
SOIL_GAS	no2 flux, co2 from soil respiration
VEG_SPEC	species composition data by site, date
WIND_PRO	noaa lidar wind profile data

Figure 6: FIFE Table Descriptions

$$N_{c_t} = N \times \alpha_t$$

The weigh vectors are adjusted as follows:

$$W_{ij_{t+1}} = \begin{cases} W_{ij_t} + \alpha_t \times (I - W_{ij_t}) & \text{if node is within } N_{c_t} \\ W_{ij_t} & \text{otherwise} \end{cases}$$

For each iteration in the learning process, one table is chosen at random. The neuron with the most similar set of weights is identified. That neuron and all the neurons within the current neighbourhood have their weights adjusted bringing them closer to the weights of the chosen table. Each neighbour of the chosen table is taken in turn. The neuron with the most similar set of weights to the neighbouring table is identified. Its weights, and those of its neighbours are adjusted to bring it closer to the original table chosen. The self organizing feature map orders itself with regards to similarity in the weight percentages and the neighbouring tables.

6 Case Study

Scientific databses (Michalewicz, 1991) are typical very large databass. Scientific and statistical data is colected by investigators from a wide range of disciplines. This data is then often stored using relational databases. This huge amount of interrelated data is then forced into the rigid table structure of a relational database which can not adequately model the necessary relationships (Kim, 1990.) We tested our approach on a subset of NASA's FIFE scientific database.

The objective of the ISLSCP (International Satellite Land Surface Climatology Project) is to develop techniques to determine surface climatology from satellite observations. FIFE (First ISLSCP Field Experiment) was undertaken at a 15km by 15km square site near Manhattan, Kansas in 1987 and 1989. Its purpose

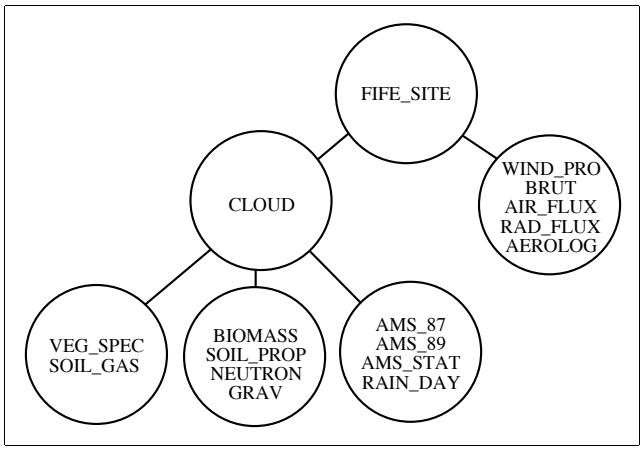

Figure 7: Clusters produced by Genetic Algorithm

was to gather enough data to allow the creation and testing of models to develop these techniques (Strebel et al., 1989.) 120 gigabytes of data was collected (300 megabytes textual data, the rest images.) The textual data fills 100 tables in a relational database, and each of those tables has 10 to 50 attributes.

Five queries utilizing 17 tables were made to the database showing the kind of queries that researchers in different, but related fields would create. Each of these queries was stored in its own appropriately named list. The tables which made up those queries are shown in Figure 6. The tables used in each of the five queries are shown below:

1. Rainfall: FIFE_SITE, AMS_87, AMS_89, AMS_STAT
2. Meteorological: FIFE_SITE, AMS_87, AMS_89, AMS_STAT, RAIN_DAY, CLOUD
3. Atmospheric: FIFE_SITE, WIND_PRO, BRUT, AIR_FLUX, RAIN_DAY, RAD_FLUX, AEROLOG
4. Surface Biophysical: FIFE_SITE, BIOMASS, SOIL_PROP, SOIL_GAS, CLOUD, VEG_SPEC
5. Soil Moisture: FIFE_SITE, CLOUD, NEUTRON, GRAV, BIOMASS, SOIL_PROP

The genetic algorithm takes 25 seconds to return the clusters shown in Figure 7. The neural network takes 30 seconds to return the clusters shown in Figure 8.

The genetic algorithm and neural network create similar clusterings of the database tables. The genetic algorithm clusters the database into 3 clusters, which are clustered into 2 clusters which are then clustered into one global cluster. The neural network clusters the databases into 8 clusters which are clustered into one cluster. The three clusters at the top of the neural network form the central cluster in the genetic algorithm. The two clusters at the bottom right of the neural network form the rightmost cluster in the genetic algorithm. The three clusters in the center and lower left of the neural network form the leftmost cluster in the genetic algorithm.

The genetic algorithm generates a strict hierarchy with an arbitrary number of levels where the relationships between the clusters are shown by their relative positions in the hierarchy. The neural network generates only two levels of clustering but the overview shows more subtle relationships between the clusters. The genetic algorithm generates deeper, more structured indices than the neural network. The neural network gives a better overall picture of how clusters are related.

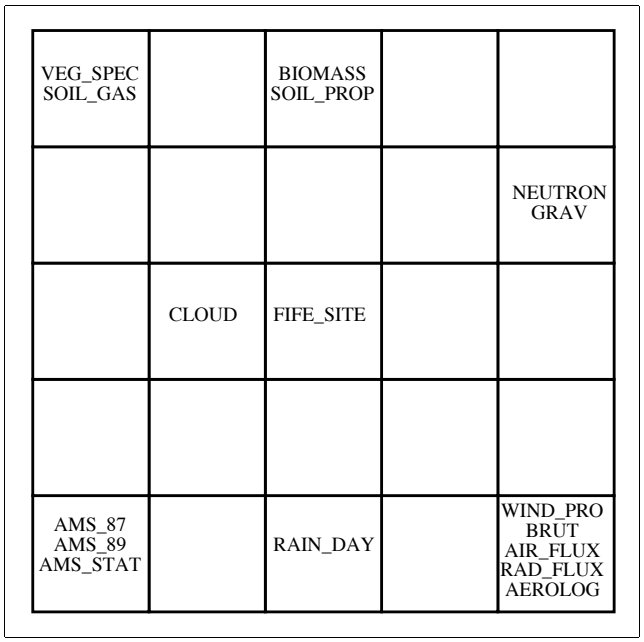

Figure 8: Clusters produced by Neural Network

The neural network clustering can help a new user see a global picture of the database. With the neural network clustering, tables used most generally will be found near the center of the array. Tables with more specific uses will be found towards the corners. This can be seen in Figure 8 where the FIFE_SITE table used in all five queries was placed in the center of the array while the VEG_SPEC table used in only one query was placed in the upper left corner.

The genetic algorithm clustering can help an administrator categorize the clusters into a hierarchical relationship. General concepts are found in the interior nodes at the top of the hierarchy while more specific concepts are found at the bottom of the hierarchy, near the actual tables being clustered. This generates a hierarchy similar to the generic hierarchy provided by most statistical and scientific databases, but this one is based on actual user patterns instead of presumed user patterns.

The strict partitioning of the genetic algorithm, like a generic hierarchy, will not completely satisfy each user. There is no half-way point between clusters. The neural network clustering does provide these half-way clusters by generating more, smaller, clusters and using their positions in the matrix to show which clusters are 'close.'

The two algorithms have comparable execution times. Compared to a deterministic algorithm clustering the tables using an identical payoff function, the genetic algorithm, on average, found a clustering better than 99% of the possible clusterings with inputs of less than 14 tables. It was impractical to do comparisons with more than 13 input tables due to the running time of the deterministic program. Thus the heuristically based genetic algorithm does the same job as a deterministic algorithm in a small fraction of the time. For a comparison of running times of the deterministic, genetic algorithm, and neural network clustering algorithms see Figure 9. The time for running the neural network is directly related to the size of the array. As the array size is based on the $\sqrt{number\ of\ tables}$, the time complexity of the neural network is a step function.

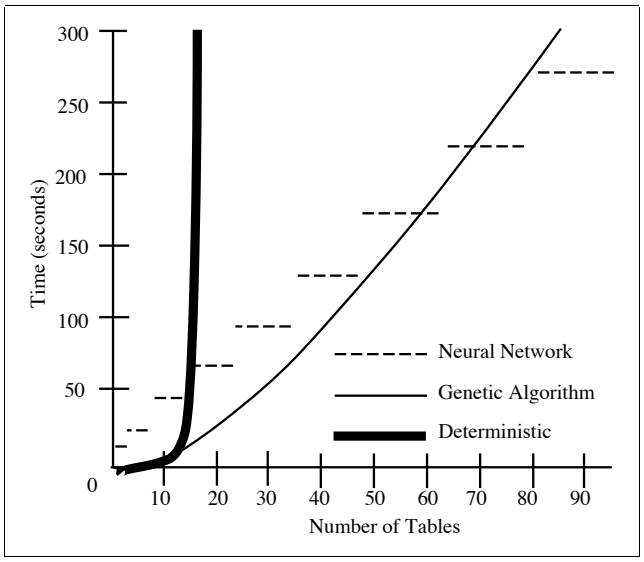

Figure 9: Comparison of Different Clustering Algorithms

Clustering will be an ongoing process in the database. New users will begin using the database, current users will return to search for new data. Perhaps even new tables will be added to the database. More and more relationships will connect the tables together, and more tables will be brought into the clustering. These new relationships may be very different from the older relationships. Over time the data in the database will be used for different purposes, so the clustering must adapt. The clustering should dynamically support both the new and current users.

Totally re-clustering the tables will allow the index to show the most effective clustering based on all of the users' previous experiences. This will be very useful to new users, but may be confusing to current users when the clustering patterns change, possibly dramatically. Instead the clustering can be performed incrementally. The current clusters can be used as an input to both the genetic algorithm and the neural network, instead of starting the clusters from scratch. This will allow the clusters to change slowly, giving the current users a common framework and adding in each new user's contributions.

More than one set of indices can be maintained in the system. New users can start with the most up-to-date indexing information, while current users can access the database using their familiar clusters. Since the clusters are derived from the input lists, different combinations of input lists will yield different orientations. Individuals, groups, and sites can maintain their own clusters as well as the global clustering for the entire database. The global clustering can be updated at regular intervals. Individuals, groups, and sites can choose how often to update their clusters. They can also decide to include all the available lists no matter how old, or only cluster based on recent usage patterns. However the clusters are maintained, the underlying database remains untouched.

7 Uses

The clusters that have been generated can be used for several purposes. They can aid both the user and the database administrator.

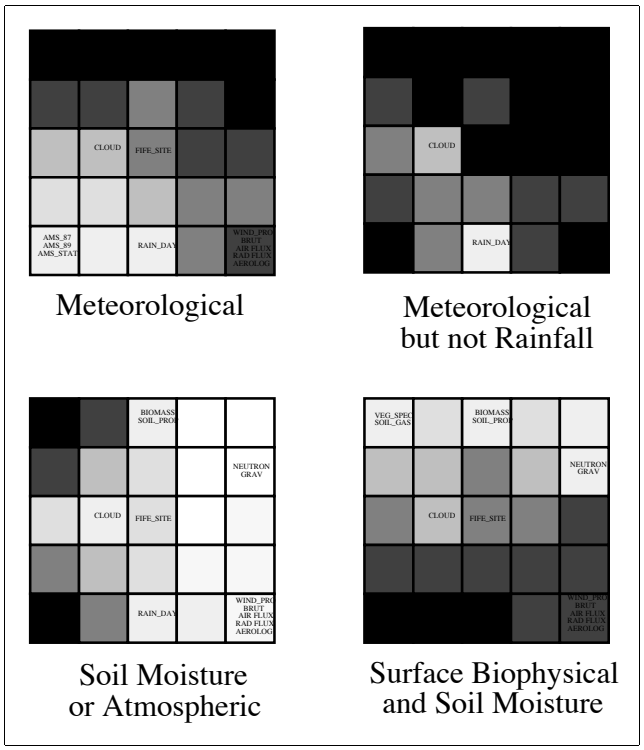

Figure 10: Sample Index Queries using Neural Network

When a user wishes to start a new query, they can look through the existing set of topics that have been queried. One of these topics may match up with their needs giving them an excellent starting point for their queries. They may find that their area of interest straddles two existing queries. Using the existing lists the user can reduce their search time in the database as shown in Figures 10 and 13.

Figure 10 shows how the clusters in Figure 8 can be highlighted. Each of the arrays in Figure 10 is in the same orientation as the larger (and more readable) Figure 8. The user can choose to see the tables important to someone interested in Meteorological information, or someone interested in Meteorological data but not Rainfall data. The user might want information related to Soil Moisture or Atmospheric conditions, or they may want information that combines Surface Biophysical data and Soil Moisture data.

By selecting the appropriate queries the clusters highlight the appropriate areas of the database. Important areas are shown in white; unimportant areas are shown in black. The brightness of the region shows how relevant the tables in that region are to the suggested query areas. Since the neural network forms a topological map of the input lists, these input lists allow us to find the literal peaks and valleys of interest. See Figure 11.

As well as highlighting the important tables, the clustering diagram also shows relationships between the highlighted tables, and how general or specific the highlighted tables are by their location in the array. Thus it gives much more information to the user than simply highlighting table names based on the usage percentages directly, or using a general overview diagram of the database.

It is unlikely that any of the existing topics will exactly match the needs of a new user so the ability to find 'nearby' tables is very important. This can only be provided by integrating the experiences of many

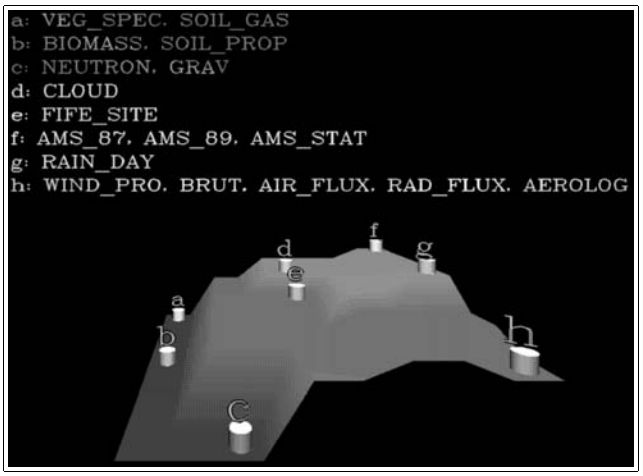

Figure 11: Visualizing the Clusters Using 3D Graphics

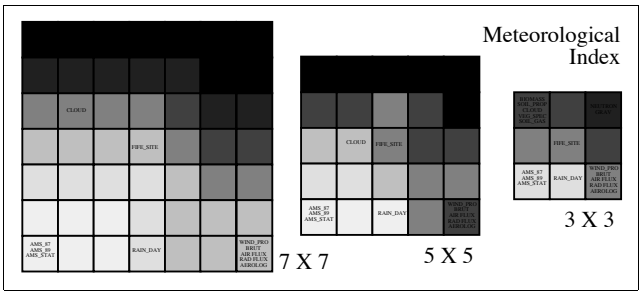

Figure 12: Zooming In and Out of the Array

users. This information can be especially important if the user needs to find intermediate tables to link together the values from tables of interest. The ability to see 'nearby' tables is also important if the user has an inexact query. The existing topic lists can give the user a good starting point to begin browsing through the 'nearby' tables to see if they are of interest.

Since there are complete weight vector values for every element in the array, the computer can interpolate these values to change the size of the array without recomputing the weight vectors. This allows the user to 'zoom in' to the array to see an area in more detail, or 'zoom out' to see less detail. As the user 'zooms out' the individual clusters form into clusters of clusters; as the user 'zooms in' the clusters break apart. See Figure 12.

Figure 13 shows how the clusters in Figure 7 can be highlighted. As with the neural network version, the user can choose combinations of the existing queries to see the tables that are of personal importance.

Using the clustering information provided by the users, the database administrator can find which tables are in common usage among which groups. For example, all the investigators would require access to the most commonly used tables but perhaps only the geologists are accessing certain tables, and the climatologists are the exclusive users of another set of tables. Using the clusters the administrator can determine which tables an investigator will need based on their areas of interest, because those are the tables that investigators with similar interests have used in the past.

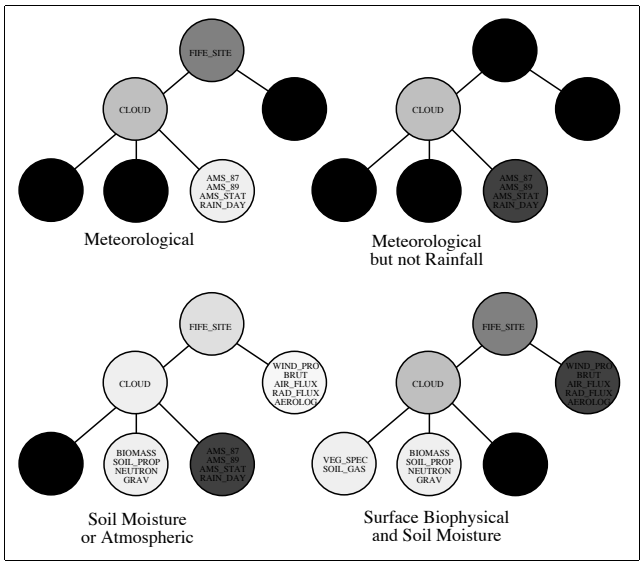

Figure 13: Sample Index Query using Genetic Algorithm

8 Conclusions and Future Work

In this paper we have proposed, implemented, and compared two adaptive indexing algorithms for very large databases. These algorithms hierarchically cluster tables according to each user's view of the database, rather than the database administrator's view of the data. This hierarchy of clusters gives each user a personal index into the database. New and experienced users can use the existing indices to speed up their access to the database.

Both the genetic algorithm and the neural network create clusters within a reasonable amount of time. The genetic algorithm generates a strict multi-level hierarchy of clusters. The neural network generates a two dimensional map of the clusters. Both algorithms allow the user to view the database in a personalized way.

We are currently enhancing our implementation in the following ways: 1. Improving cluster visualization using 3d graphics, 2. Applying this approach to other database systems, 3. Using the stored list information to suggest appropriate join and display columns, 4. Decreasing clustering time through parallelism,.

Acknowledgements

We gratefully acknowledge the assistance of Narendra Goel, John Norman, and Don Strebel for their insight into the FIFE database. We would also like to thank the reviewers for their valuable comments.

References

[French et al., 1990] French, J., Jones, A., and Pfaltz, J. (1990). Summary of the final report of the NSF workshop on scientific database management. *SIGMOD RECORD*, 19(4):32–40.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine learning.* Addison-Wesley, New York.

[Gordon, 1988] Gordon, M. (1988). Probabilistic and genetic algorithms for document retrieval. *Communications of the ACM*, 31(10):1208–1218.

[Hara et al., 1991] Hara, Y., Keller, A., and Wiederhold, G. (1991). Implementing hypertext database relations through aggregations and exceptions. In *Proceedings of the Hypertext '91 Conference*, pages 75–90.

[Ioannidis et al., 1992] Ioannidis, Y., Livny, M., and Haber, E. (1992). Graphical user interfaces for the management of scientific experiments and data. *SIGMOD RECORD*, 21(1):47–53.

[Ioannidis et al., 1993] Ioannidis, Y., Livny, M., Haber, E., Miller, R., Tsatalos, O., and Wiener, J. (1993). Desktop experiment management. *Data Engineering*, 16(1):19–23.

[Johnson and Fotouhi, 1993] Johnson, A. and Fotouhi, F. (1993). Automatic touring in a hypertext system. In *Proceedings of the 12th IEEE International Phoenix Conference on Computers and Communication*, pages 524–530, Tempe, Arizona.

[Johnson et al., 1994] Johnson, A. Fotouhi, F., and Goel, N. (1994). Adaptive clustering of scientific data. In *Proceedings of the 13th IEEE International Phoenix Conference on Computers and Communication*, pages 241–247, Tempe, Arizona.

[Johnson, 1994] Johnson, A. (1994). the SANDBOX: a Virtual Reality Interface to Scientific Databases. Ph.D dissertation (in preparation), Wayne State Univ., Detroit, Michigan.

[Kim, 1990] Kim, W. (1990). Object-oriented approach to managing statistical & scientific databases. In *Proceedings of the Fifth International Conference on Statistical & Scientific Database Management*, Charlotte, North Carolina.

[Kohonen, 1990] Kohonen, T. (1990). The self-organizing feature map. *Proc. IEEE*, 78(9):1464–1480.

[Kuntz and Melchert, 1989] Kuntz, M. and Melchert, R. (1989). Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power. In *Proceedings of 15th International Conference on Very Large Data Bases*, pages 97–105, Amsterdam, Holland.

[Michalewicz, 1991] Michalewicz, Z., editor (1991). *Statistical and Scientific Databases.* Ellis Horwood.

[Nielsen, 1990a] Nielsen, J. (1990a). *Hypertext and Hypermedia.* Academic Press, San Diego.

[Nielsen, 1990b] Nielsen, R. (1990b). *Neurocomputing.* Addison-Wesley, New York.

[Özsoyoğlu and Wang, 1993] Özsoyoğlu, G. and Wang, H. (1993). Example-based graphical database query languages. *IEEE Computer*, 26(5):25–38.

[Strebel et al., 1989] Strebel, D., Newcomer, J., and Ormsby, J. (1989). Data management in the FIFE information system. In *Proceedings of International Geoscience and Remote Sensing Symposium*, pages 42–45, Vancouver, Canada.