The Software Architecture of DBLEARN

Colin Carter, Howard J. Hamilton* and Nick Cercone
Department of Computer Science
University of Regina
Regina, Sask., Canada S4S 0A2
{carter,hamilton,nick}@cs.uregina.ca

January 31, 1994

Abstract

The structure of DBLEARN is described and some improvements to its design and functionality are proposed. DBLEARN is a knowledge discovery from databases program that uses attribute oriented generalization guided by concept hierarchies. DBLEARN consists of five modules: the user interface, the command module, the database module, the concept hierarchy module, and the learning module. Each of these are described as to the function they perform, what input they require and output they produce. Detailed suggestions for improvements to four of the modules are also presented.

1 Introduction

Knowledge discovery from databases is the automated extraction of useful and interesting, implicit information from large bodies of diverse data stored in databases. This information is implicit in that it is contained in the data, but it is hidden by a preponderance of overly specific data. One primary method of extracting the information is by generalizing specific data values into more general concepts. The primary objectives of such discovery is to produce new non-trivial information that is understandable, accurate and useful [Frawley et al., 1992].

DBLEARN is a machine learning program which implements attribute oriented generalization using concept hierarchies [Han et al., 1992, Han et al., 1993]. A prototype implementation was made available to us by Jiawei Han of Simon Fraser University. We modified and to some extent redesigned this prototype to produce the version of DBLEARN described in this paper. For details on changes made to improve efficiency, see [Carter and Hamilton, 1994].

Attribute oriented generalization seeks to transform diverse data stored in database relations into more general and useful information on an attribute by attribute basis

*Supported by Natural Sciences Engineering Research Council of Canada (NSERC) Research Grant OPG0121504,

[Cai et al., 1991]. Generalization is accomplished by replacing the specific attribute data values of each tuple with more and more general concepts. In this way, the number of distinct values of each attribute is reduced as related values are grouped into more general concepts. Many tuples then become redundant as the information they contain becomes identical to other tuples. These redundancies are eliminated by removing all but one of the identical tuples. This generalization continues until an acceptable level of information is attained.

The generalization is guided by concept hierarchies associated with each attribute of the relation to be generalized. A *concept hierarchy* is a tree structure defined by a domain expert that provides a hierarchical taxonomy of concepts ranging from a single most general concept at the root of the tree to some representation of all possible attribute values at the leaves. Concepts are formed by grouping related attribute values together and representing all members of this set or range by a single symbol. These symbols in turn may be grouped into more general concepts which are again represented by another unique symbol. This grouping continues until the most general concept, called *ANY*, is reached.

The generalization process is limited by the specification of two tunable threshold values, the attribute threshold and the table threshold. The *attribute threshold* specifies the maximum number of distinct values of any attribute that may exist in the final generalized relation. Generalization proceeds first by reducing the number of distinct values of attributes to less than or equal to the attribute threshold. Duplicate tuples are then removed. The resulting relation is called the *prime relation*. The *table threshold* specifies the maximum number of tuples that may exist in the final generalized relation. If the number of tuples in the prime relation exceeds the table threshold, then further generalization must take place. This is accomplished by choosing an attribute by some method, generalizing it another level, and then removing duplicates. This process iterates until the number of tuples is no more than the table threshold.

The result of generalization, therefore, is a relation with a limited number of tuples containing much more general, summarized information than was in the original ungeneralized relation. Each tuple in this relation represents a *class* of the original tuples. Any of the tuples that were generalized into this final tuple are specific members of this class.

This generalization procedure is used to accomplish two types of learning tasks, characteristic learning tasks and classification or discriminate learning tasks. *Characteristic learning* tasks operate on one set of data and seek to discover interesting relationships between attributes of a relation. An example might be the relationship between the amount of money a researcher is granted and the area of study in which the researcher is working. Another example might be the relationship between observed symptoms and a disease. Classification or *discriminate learning* tasks operate on two sets of data and seek to determine interesting distinctions between the characteristics of the concepts represented in each. An example might be the amounts of money granted to researcher in one field compared to the amounts of money granted researchers in a different field. Another example might be symptoms that distinguish one disease from another. These two types of learning tasks are the primary functions of DBLEARN.

At least four versions of DBLEARN are currently being used or developed, three in academic institutions and one in a commercial setting. The version running at the University of Regina has been ported to an IBM compatible 80386 microcomputer running on OS/2

Figure 1: DBLEARN Components

version 2.1. It uses IBM's DB2/2 database, a port of their mainframe relational database product. It is written in C using IBM's CSetPlus compiler. It has two user interfaces, a command line interface developed with UNIX's lex and yacc compilers, and a prototypical OS/2 Presentation Manager graphical user interface.

The remainder of this paper is organized as follows. In Section 2, we describe the architecture of DBLEARN in some detail as it is and as it should be. In Section 3, we suggest some enhancements that might be incorporated into its general design and implementation. Finally in Section 4, we present conclusions. In the Endnotes section at the end of the paper, we identify specific discrepancies between our software architecture for DBLEARN and the current implementation.

2 The Architecture of DBLEARN

As shown in Figure 1, DBLEARN consists of the following components:

1. User Interface Module
2. Command Module
3. Database Access Module
4. Concept Hierarchy Module
5. Learning Module

The user interface module is the primary communication medium between the user and the program as a whole. It handles all input, passing user requests to the command module

Figure 2: Interaction with the User Interface Module

and receiving back information to be displayed. The command module in turn controls the majority of internal program direction. It receives its direction from the user interface and routes the requests to the appropriate module for processing. The command module communicates with the database module to gain information about available database objects, and it initializes the learning task query to the database. It also directs the concept hierarchy module to build the concept hierarchies and directs the translation of high level learning requests into low level SQL queries. When all relevant task information has been set by the user interface, the command module starts the generalization process with a call to the learning module. This module is the heart of the generalization process. After activation by the command module, the learning module interacts with both the database module to retrieve data and the concept hierarchy module to gain generalization information to reduce the database relations to informational size. These results are passed back through the command module and then to the user through the user interface module.

An attempt has been made in this design to make these five units as distinct and separate from each other as possible. The current version of DBLEARN achieves this modularity to a great extent. For the most part, this paper will describe the design as it should be, noting in the Endnotes where discrepancies occur in the current actual version. Each of the modules will now be described in greater detail.

2.1 The User Interface

The primary purpose of the user interface is to provide a way for the user to easily initiate learning tasks and set and access information about all aspects of this process. It should also allow interaction between the user and the various modules of the program to correct error conditions and give direction as needed. These and other features are described in this section.

The interaction between the user interface module and the command module are shown in Figure 2. The user communicates with the interface through the keyboard at least. If a graphical interface is available, the mouse will also be a primary input tool. In addition the user may use I/O redirection to obtain input from a file. All user requests are routed to the command module for direction to other modules for handling. These other modules will carry out the request from the user and return some sort of structure or error condition to the command module. The command module will return these to the user interface. It is the user interface's responsibility to handle all display or output of results and interaction with the user to correct error conditions.[1]

There are two interfaces available at the current time, a command line interface and a prototypical graphical user interface developed under OS/2's Presentation Manager. These two interfaces will now be described and analyzed to some extent.

2.1.1 Command Line Interface

The primary current interface in use is a command line style interactive interface. It has been created using UNIX's lex and yacc programs. All information and feedback is entered in a simple text based screen, and therefore is available to any computer with an ASCII interface. The yacc interface has been described in [Huang and Cai, 1993]; here, we simply mention some modifications.

In the original program, all information for a learning task is defined in one large syntactic unit. That is, the individual units of information needed for defining the task are specified in a predefined and restricted order as a part of one large task definition paragraph. If the user accidentally enters a syntactic error in a latter part of the task definition entry, such as misspelling a keyword, the whole process must be started again. This is very frustrating to the user.

In the current version, the task definition process is broken down into much smaller parts, each of which establish a current default for the various parts until changed. For example, the `set` command sets any of four threshold values; e.g.,

```
set attribute threshold <number>
```

sets the value of the attribute threshold in the task information structure and returns. Similarly, all other individual pieces of information that the command module provides functions for setting can be set in a similar fashion. Those that perform the construction of some data structure, such as the specification of a concept hierarchy file, build the structure right away so that error conditions can be corrected immediately. When the learning task is completed, the values remain set and only those that need changing for a subsequent task need to be specified. For example, after completing a task, the attribute threshold can be adjusted, the start command issued, and the same task would be repeated with these changes.

In the modified program, commands are provided to ask queries about the databases. For example, the command "`what databases`" returns a list of current databases known by the DBMS. Similarly, the command "`what tables`" returns a list of tables that exist in the current connected database, and "`what attributes for <table_name>`" returns the attributes defined for a given table. There is also provision for retrieving information about the schema of a given table.[2]

Finally, the current yacc interface does not use the command module structure described in this paper. The information needed to initiate a learning task is defined and then saved to a file which the learning module opens and reads to gain the information it needs. The learning module itself then directs most of the activity that this paper describes as the responsibility of the command module.

2.1.2 Prototypical Graphical Interface

As an alternate to the yacc interface, a prototypical graphical user interface (GUI) has been developed under OS/2's Presentation Manager. It is an interface which uses many of the standard graphical entry and feedback mechanisms now common in many GUI's such as Microsoft Windows. It is written is C++ using IBM's CSetPlus compiler. The main windows and dialogs were designed using the dialog editor provided with the Programmer's Toolkit that IBM bundles with their compiler.

The primary input structure in this interface is a task definition dialog window which has a number of different input objects used to set various specifications of a learning task. This input structure and the functions it provides are the primary focus of the following discussion.

Dynamically Retrieved Information A significant advantage of the graphical interface is that it dynamically provides much of the information needed for specifying a learning task. This saves the user from having to remember such things as the precise names of databases, relations and attributes. When the GUI is started, the database manager (DBMS) system tables are queried for all defined databases. These are presented to the user in a drop down list box[3] for selection of the desired database. That database is then connected, and the DBMS system tables are again queried for all defined tables and views in this database.

These are presented to the user in a pop-up multi-select list box.[4] The user is then able to select as many tables as are relevant for the learning task. A click on an OK button signals the end of table selection. The selected items are then entered into a plain scrollable text box so the user can see only which tables were selected. If the user selects only one, it is entered into this list in an unmodified form. If more than one table is selected, a correlation name[5] is automatically added to each table name to make the construction of queries easier at a later time.

Next the database's system tables are again queried for all defined attributes of the tables that were selected. These attributes are presented to the user in another pop-up, multi-select list box, and the user chooses a subset of attributes that are to be in the relation to be generalized. These are similarly entered into a plain list box in the main dialog window so the user can see which ones have been selected.

This information is not set permanently for the current learning task, but may be changed at any time by clicking on the drop down symbol of the database list or one of two push buttons[6] associated with the relations or attributes lists.

Setting General Learning Task Parameters When the user has been guided through these steps to select the database, tables and attributes relevant to the learning task, he or she may now specify the rest of the information needed to initiate a learning task.

The task type, characteristic or discriminate, may be selected from a two item drop down list box, with the user's choice being the only one to remain visible after selection. The choice of this task type also determines what further options are available to the user.

If a discriminate task is chosen, then the area of the main task definition dialog window for defining a contrast predicate is activated and made available for input. If a characteristic task is chosen, these areas are deactivated so that input is not possible. This saves the inexperienced user from specifying unneeded information.

The concept hierarchy files may be selected from a multi-select, browsable file list[7] when a "Set Hierarchies" button is pressed. This frees the user from having to remember the precise name and directory of hierarchy files. The subset of attributes that the user desires prop amounts for may also be selected when a "Set Prop" button is pressed. A list of all numeric attributes is presented to the user[8] and as many as are desired may be selected. These are marked with a 'p' in the attributes list box so the user has an easy way of telling how many props are specified. A special check box[9] is also available for the prop(votes) function.

The attribute and table thresholds may also be set by clicking in one of two simple text entry boxes.[10] The text in these boxes is preset to the threshold defaults of 10 and 5 for table and attribute thresholds respectively, but any valid integer may be entered. The system verifies that user's entry is a valid number and pops up an information box to signal erroneous input.

The titles of the target and contrast relations may also be set in simple text entry boxes. The contrast relation title box will not be activated if a characteristic task type is selected as the current task type.

Finally, the target predicate and contrast predicate if needed can be entered in larger editable text entry boxes. The predicates may be entered in normal DBLEARN syntax in these areas. This is basically a minimal patch until a more powerful graphical query definer can be devised. The current syntax of DBLEARN predicates is restricted enough that the user could be guided through the definition of queries by being presented with dynamically determined choices for each step.

One additional option is available to control the amount of information output by the system. A "Debug" button may be pressed to present a pop-up menu with two choices - intermediate output or final output only. The intermediate option causes such things as the high and low level queries, and the prime relation as well as the final generalized relation to be printed on the output stream. The final output only selection causes only the final generalized relation to be output.

When the user has defined all information for a learning task, the generalization process may be initiated by clicking an "OK" button. This sets in action the retrieval of database information, the building of concept hierarchies and the generalization process.[11] The task definition dialog disappears and a simple editor window appears into which the results of the learning task are entered.[12] The results may be saved to file from this window or printed as the user desires.[13]

Limitations of the Current Prototype As was already mentioned, the predicate definition is still accomplished by simply typing the predicate in. A more powerful guided query builder should be written for this.

There is also no features currently available for displaying information about concept hierarchies or relation schemas. There should be some way of displaying arbitrarily specified concept hierarchies from a specified file.

Figure 3: Interaction with the Database Access Module

In spite of these limitations, the graphical user interface shows great promise. Defining a learning task is much easier in this environment than the yacc command line interface, and the potential for user assistance is query building is much greater.

2.2 The Command Module

The command module is the primary controller of communication between various modules of the DBLEARN system. As we have seen, it receives requests from the user interface to set various parameters of the learning task or gain information about available structures. The command module also interfaces to the other three modules, the database module, the concept hierarchy module and the learning module to control most other aspects of program operation. It interfaces to the database system to get information about available databases, tables and attributes. It controls the loading of concept hierarchies. It initiates learning tasks and receives and relays interaction requests from the learning module as generalization progresses. It then returns the results of all of these to the user interface for display. The command module will be described in terms of its interaction with the database module, the concept hierarchy module and the learning module, and other miscellaneous functions that it performs.

2.2.1 Command Module's Interaction with the Database Module

As shown in Figure 3 The command module performs various functions with regard to the database module. The primary responsibility of the database module is to provide one or two relations to be generalized for characteristic or discriminate tasks. The command module provides functions necessary to build these. This includes starting the DBMS if necessary, retrieving information about the current database objects, connecting to specific databases, retrieving more detailed information about the current connected database, and building and initiating the actual queries for target and contrast relation retrieval.

Starts DBMS if Necessary On some systems, the DBMS may not be running when DBLEARN is started. In this case the command module provides a function to start the target DBMS as a background process. In some installations, this will be unnecessary in that the DBMS is running constantly as an essential part of the organization.

When the DBMS is running, the current user of DBLEARN needs to be logged onto the system. DBLEARN cannot be used to circumvent database security and so every user must access the DBMS according to his or her own security level. Success or error codes signal the results of these operations.

Retrieves General Database Information The user may not remember the specific names of database objects that he or she wishes to access, but the command module provides functionality to access this information. It allows for the querying of the database system tables to find out what databases are available, what tables or views are available in a specific database and what attributes are available in a specific table or view. The names are returned as lists of strings. This allows the user interface to provide this information to the user to guide the task definition process.

Connects to a Database After the user interface has been provided with what databases are available, the command module may be requested to connect with a specific database. Only one database may be current at a given time. The connect command establishes that database as the current database until a subsequent connection is established or the connection is broken through an explicit disconnect or through some exceptional error condition.

Retrieves Relation Schemas When a connection has been established, the user may want to know more specific information about the tables or view that are available. The command unit is able to query the database for schema information about any available table or view. The result is returned in a structure which can be passed to the user interface for display.[14] This will enable the user to review what options are available for task definition.

Initializes the Retrieval of Target and Contrast Relations One of the major inputs to the learning task is a relation or set of two relations to generalize. It is the command module's responsibility to guide the building of the necessary query to extract these relations and to interface to the database module to initialize the query to retrieve tuples for the learning module.

The command module provides functions to accomplish the various steps of this procedure. These begin with specifying the database tables or views from which the target and contrast relations will be extracted. Then the attributes of these tables which will be in the relations to be generalized can be specified. The target and contrast predicates must also be constructed.

These predicates delineate more precisely what subset of data is desired from the tables to be queried. This subset of database data may be described in terms of high level concepts rather than low level, specific attribute values. The high level concepts must match concepts that are defined in the concept hierarchy tree for the attribute to which they pertain. The concepts will be then translated by a call to the concept hierarchy module into more specific values relevant to the actual data.

From these various parts, the tables, attributes and predicates, one or two SQL queries are constructed and passed to the database module for initialization. The initialization process checks whether each query is valid and sets up the database to retrieve tuples.[15] If any error occurs, this will be signalled to the user through the user interface and corrective measures can be taken.[16]

2.2.2 Command Module's Interaction with the Concept Hierarchy Module

The primary functions of the command module's interactions with the concept hierarchy are to direct the loading of individual concept hierarchy trees, provide access to these trees so the user interface can display them, perform the translation from high level concepts to low level database attribute values, and to adjust concept hierarchies based on statistical information in database relations.[17]

The user may specify a file or set of files from which concept hierarchies will be read. It is the command module's responsibility to call the appropriate concept module function to load the proper hierarchy trees from these files and to return to the interface any errors or conditions that need response. Only trees that refer to attributes in the join of the relations of the current learning task need to be read, since only these attributes will be generalized. For this reason, the target relations need to be specified before the concept hierarchies are read in. Error conditions will include ill-formed concept hierarchy definitions or missing trees for a specified attribute. There is also the possibility of two or more trees defined for the same attribute. This could be handled by simply reading the first or last to be encountered, or the user could be asked which one to load. The return to the user would be code signalling success or error.

The concept hierarchy trees need to be available to the user interface for display. A simple function which accepts an attribute name as an argument and returns a pointer to the hierarchy tree for that attribute is all that is necessary here. The user interface could then display the concept in an appropriate manner.

Eventually the user will specify a query that may include a predicate to extract information from the database for the target or contrast relation. This predicate may include high level concepts as part of the specification for tuple extraction. These high level concepts need to be translated into values that are actually contained in the database, and the query needs to be adjusted accordingly. It is the command module's responsibility to call the appropriate concept functions to accomplish this when the user interface sets the target or contrast predicate. The return could be a code signalling success or error. A separate function would provide access to the translated predicate which would return the SQL string that will be used to query the database.

Finally the command module will provide the user interface with the ability to adjust a concept hierarchy based on the statistical distribution of attribute values in a relation.[18]

2.2.3 Command Module's Interaction with the Learning Module

The command module is responsible for providing all inputs to the learning procedures. These inputs include the relation(s) to be generalized and the relevant concept hierarchies needed to generalize the attributes of these relations. There are also certain parameters that specifically guide various aspects of the generalization process.[19] As shown in Figure 4, the command module provides functions for setting these parameters and for starting the generalization process. The command module stores these parameters in a single task

Figure 4: Interaction with the Learning Module

information structure, the fields of which are available through access functions.

Learning Task Types A function is provided for setting the type of learning task. The two major task types are CHARACTERISTIC and DISCRIMINATE. The first will cause the extraction of one set of data for generalization, and the second will extract two relations to be generalized and contrasted with each other. An additional task type needed is a FURTHER_GENERALIZATION task which will guide the process of taking a prime relation to a final generalized relation.[20]

Threshold Values Various threshold values guide the extent to which generalization is carried out. Functions are provided to set each of these. The attribute threshold specifies the maximum number of distinct values of any attribute that may exist in the generalized relation. When generalization reaches this stage, the prime relation is the result.

The table threshold specifies the maximum number of tuples that may be in the final generalized relation. After the prime relation is achieved, if there are more tuples than the table threshold, further generalization must be accomplished.

Two noise thresholds are also necessary for eliminating exceptional cases from the generalized relations.[21] In characteristic learning tasks, some of the tuples in the final relation may only represent a very small percentage of the original tuples in the ungeneralized relation. A t-threshold may be specified to prune those tuples that represent, for example, less than five percent of original tuples, since this percentage of examples may be considered more as exceptional than the rule.

In discriminate learning tasks, the target and contrast relations are first reduced to prime relations. An intersection of these is then performed and *overlapping tuples* (tuples that have a duplicate in the other relation) are marked. Unmarked tuples in the target relation represent classes of tuples that are derived solely from the ungeneralized target relation. Marked tuples represent classes that are represented to some degree in both target and contrast relations. The ratio of tuples represented by a class in the target relation compared to the total tuples in that class in both the target and contrast relations can be quantified as a *d-weight* for that class ([Han et al., 1993], p.36). The d-weight of unmarked, non-overlapping tuples is therefore 100%, since all examples of that class existed in the original target relation.

A high d-weight signals that most of the examples in the class were derived from the target class with only a few exceptional cases from the contrast relation. A low d-weight signals that most of the examples of the class are in the contrast relation, and those in the target relation are exceptional. A medium d-weight carries very little information since the class is represented somewhat evenly in both relations. A *d-threshold* can be specified to include in the final generalized relation of a discriminate task those tuples that represent a high percentage of original tuples in both relations. A d-threshold set to 90%, therefore, would allow tuples with up to 10% exceptional cases in the contrast relation to still be included in the final result.[22]

Prop Values As generalization progresses, duplicate tuples are removed by the deletion of all but one of the duplicates. In this combining of tuples, there is a mechanism whereby the values of selected numeric attributes can be accumulated. The value of this total for a selected attribute of a single remaining tuple, therefore, represents the sum of all original attribute values that contributed to that tuple. When the final generalized relation has been achieved, a total for each specified numeric attribute is available for each tuple. From these values, a grand total for each selected attribute in the relation can be calculated. For each tuple, a ratio can now be calculated which represents how much an individual attribute's amount represents in proportion to the grand total for all tuples. This ratio is currently known as a *prop value*, which is perhaps derived from "proportion."

DBLEARN will calculate prop values for only those numeric attributes that the user selects. The command modules provides a function which takes as an argument a list of attribute names and declares these as prop type attributes. When generalization is initiated, the appropriate totalling mechanism will be activated for these.

There is a special prop value which can be requested by the user which performs a slightly different task. When duplicate tuple removal is in progress, a total of the number of tuples which contribute to a more general tuple is tracked as a vote amount. A final tuple's vote count represents the total number of original tuples that are represented by that generalized tuple. A proportion of a tuple's votes in relation to the total number of votes in the whole relation may be calculated and displayed with final generalized relation. This prop value represents the percentage of original tuples which are covered by the generalized tuple. A function of the command module is provided to signal the inclusion of this votes prop in the generalized relation.[23]

Task Initiation and Feedback When all information needed by a learning task has either been defined or is covered by some default value, the learning task itself may be initiated. The first stage of this is the generalization of the target relation to the prime relation. If the current task type is discriminate, the contrast relation will also be generalized. The command module provides a simple start command which initiates this process and generalizes the input relation to the attribute threshold. If errors occur, they are signalled by the command module.[24]

After initial generalization, an iterative process of further reduction must occur until the final generalized relation is attained.[25] The user would need to be queried as to the desired reduction method and the process continued until the table threshold is reached. As an option, the user could specify one reduction method for the rest of the generalization process and allow it to continue to completion. A copy of the prime relation should be saved so that different reduction strategies could be tried without having to repeat the first expensive stage of generalization.

2.2.4 Command Module's Other Functions

The only other function that does not fall in the above categories is the quit command.[26] Both interfaces have a quit command, which disconnects the current database, the logs off the user, and (for those systems where the DBMS was started by DBLEARN) removes the DBMS background process.

2.3 Database Module

The database module provides all the functions for starting and accessing the DBMS, gaining information about database objects and querying to retrieve relations. These are written with mixed C code and inline SQL statements. The inline statements are preprocessed by a database preprocessor which translates them into C database function calls.

2.3.1 Database Connection

As mentioned above, some DBLEARN implementations may run on machines where the DBMS is not running constantly. Where this is the case, the DBMS must be started as a background process. A part of this start-up procedure is the log-on procedure which establishes the current user's security level and database access privileges.[27] In addition a specific database must be connected. Functions are provided by the database module to accomplish all these tasks.

2.3.2 General Information Feedback

The database module also provides functions to gain information about available database objects. These include what databases are available and what tables and views are defined for these databases. Functions are also provided for gaining access to relevant schema information about the currently available tables, especially attribute names and types.

2.3.3 Relation Retrieval

The database module also contains functions for establishing queries and retrieving tuples from the resultant relation. Only one query may be current at a time, and this is established by passing a SQL string to a function which initializes the database to retrieve tuples from the resulting relation. After that, tuples may be retrieved one at a time until the database signals that no more are available.

2.4 Concept Hierarchy Module

The concept hierarchy module is primarily responsible for building the concept hierarchy trees, translating queries involving high level concepts into low level SQL queries, and providing the matching of attribute values with concepts in the hierarchy trees.

2.4.1 Concept Hierarchy Tree Construction

The internal concept hierarchy tree data structures are constructed from information specified in concept definition files. Before the user initiates a learning task, one or more files may be passed to the concept module for loading. If none are specified, a default file, called `concept` is read from the current directory. In addition, a list of attributes is available which specifies what attributes will be in the full join of the relations to be generalized. Only those trees corresponding to these attributes need be read in from the concept files.

The definition of concept hierarchies requires that the leaf nodes of the trees represent all possible values that may be encountered in the relation to be generalized. These attribute values may be of two types, range or non-range. Range values are values that can inherently be divided into ranges bounded by lower and upper bounds. Range values include all numbers, dates and times. The notation $x\tilde{}y$ denotes a range value with a lower bound of x and an upper bound of y, such that for any $z \in x\tilde{}y, x \leq z < y$. In this way, attribute values with potentially limitless distinct values can be compactly represented in the concept hierarchy tree.

On the other hand, non-range values are all strings and alpha-numeric codes or range types which contain only a finite, limited set of distinct values that have no inherent ordering and therefore need to be individually specified. Since non-range values represent limited sets of distinct values, the concept hierarchy trees can have the actual attribute values as leaves.

The leaf nodes are arranged into related groups and these groups are represented by a single more general symbol. These higher level symbols are in turn grouped into yet more general concepts which are in turn represented by a higher level symbol. This generalization of concepts continues until the highest, most general concept "`ANY`" is reached at the root of the tree.

The concept hierarchy is not required to be a balanced tree. The user is free to define a tree with whatever structure is desired, placing some leaf values on the same logical level as other non-leaf, generalized concepts. The learning module has been adjusted to properly handle such tree configurations.

The function that builds this concept hierarchy tree, then, attaches it to a hierarchy pointer in the list of attributes built by the schema submodule of the concept hierarchy module and it becomes available to other modules.

2.4.2 High Level Concept Translation

DBLEARN is designed so that extraction of data from the database may be accomplished by expressing queries in terms of high level concepts in the concept hierarchy tree. Since the tree's leaf nodes represent all the possible values that may be encountered in the data, high level concepts can be translated into low level specifics by descending the tree from a higher node and retrieving all the leaf values for comparison against the database.

For range values, a predicate such as:

```
where discipline_code = "Computer"
```

might be translated into:[28]

```
where (discipline_code >= 23000 AND   discipline_code < 23500) OR
      (discipline_code >= 23500 AND   discipline_code < 24000) OR
      (discipline_code >= 24000 AND   discipline_code < 24500) OR
      (discipline_code >= 24500 AND   discipline_code < 25000)
```

which is simply a concatenation of leaf values under the concept `Computer`. For non range values, a predicate such as:

```
where province = "Prairies"
```

might be translated into:

```
where province = "Alberta"  OR
      province = "Saskatchewan"  OR
      province = "Manitoba"
```

When the user has entered the predicates for the target and contrast relations, these are passed by the command module to the concept module and this translation is accomplished. Errors may be signalled for syntactically incorrect predicates or concepts which are not found in the concept hierarchy tree.[29]

2.4.3 Attribute Value Matching

When tuples are read in from the database module, they are initially stored in a DBLEARN relation structure. The current version reads in the whole relation to be generalized and then begins generalization.[30] This makes it necessary to store the attribute values in a space efficient way. When generalization begins, the attribute values must be matched against the concepts in the hierarchy tree so that more general concepts may be retrieved. Since the tree will need to be searched many times, it is necessary to perform this searching quickly. Both these requirements are fulfilled indirectly through the way in which the concept hierarchy information is stored and built.

Since the leaf nodes of the concept hierarchy tree represent all values that may be found in the relation to be generalized, we know before any data is read from the database all possible values that might be encountered. For non-range values, therefore, a hash table can be constructed in which one copy of each hierarchy symbol is stored with its string representation. A pointer back to the concept hierarchy represented by this symbol can also be stored in this table. All attribute values read from the database, therefore, can also be hashed and stored as references to this table. In actuality, attribute values are hashed into the table, the matching concept is retrieved, and the attribute values are stored as pointers to the actual concepts they match. This gives instant access to generalization information when needed.

For range type values, the conceptual storage of the actual value can be used, for example, storing integers as integers. The concept hierarchy tree can have extra fields defined to store the inherent types for the upper and lower bounds of the range values. When the first round of generalization is accomplished, the range values can be matched against these boundary values. On the first level of generalization, all values, range or non-range are replaced by concept symbols. Range values after generalization, therefore, can simply be replaced by a pointer to a concept, just as other values.

Since attribute values are stored as pointers to concepts, when it comes time to generalize an attribute, all that is needed is to access the attribute's concept and reset the attribute's value to the concept's parent in the hierarchy tree. No further searching of the tree is necessary after the first initial setting of the attribute's value. This provides extremely fast generalization.

In the duplicate removal stage, attribute values need to be compared against each other. Since they are stored as pointers into the concept hierarchy tree, all that is needed is a direct comparison of these pointers to ascertain whether two values are equal. Again this is a fast and efficient method of comparison allowing the duplicate removal process to proceed quickly.

When the user interface is called upon to print any relation, the concept structure contains a reference into the hash table where its string representation is stored. This link can be easily followed to retrieve the needed string.

2.5 Learning Module

The learning module contains the heart of the machine discovery process. Its primary inputs are the relation to be generalized, a set of concept hierarchy trees to guide the generalization and a structure detailing parameters that affect the generalization procedures. All of these elements have been described above. The primary steps in the learning process are initial generalization, duplicate removal, and further generalization. For characteristic learning tasks, the resultant generalized relation is the final product. For discriminate learning tasks, there will be two generalized relations which need to be compared to extract interesting differences.

2.5.1 Initial Generalization

The initial generalization process simply involves iterating through each attribute of the input relation, replacing each attribute value with a more general concept and tracking the number of distinct values of attributes that remain after each iteration. When the number of distinct values of every attributes falls within the range of the attribute threshold, this stage of generalization ceases.

As has been mentioned, the attributes are stored as pointers to concepts and so the actual replacement of a value with a more general concept is fast and easy. One issue that needs addressing, however, is how to handle unbalanced trees.

A leaf node may not exist at the maximum depth of the tree when the tree is unbalanced. A measure of its distance from the maximum, distance_to_max, may be calculated by subtracting the node's depth from the maximum depth of the tree. For example, in

Figure 5: Example Concept Hierarchy Tree

the tree shown in Figure 5, the distance_to_max of Alberta, Saskatchewan and Manitoba is 0, whereas B.C.'s is 1 and Ontario's is 2. As generalization progresses, the level of generalization is tracked, with the first round being level 0. As attributes are accessed, if the generalization level is greater than or equal to the corresponding node's distance_to_max, the node is generalized. Otherwise it is not. In the above tree, the three prairie provinces would be generalized in the first round, B.C. and the Prairies in the second round, and Western and Ontario on the third round.

2.5.2 Duplicate Removal

When the attribute threshold has been reached for each attribute in the relation to be generalized, there will likely be many duplicate rows in the resulting relation. These are removed through a simple procedure. A new relation is created into which the first tuple of the generalized relation is copied. Then the rest of the tuples in the relation are compared to the elements of this new relation. As each of the original tuples is read, if it does not match any tuple in the new relation, it is added to it. If a tuple match is found, the vote count of the tuple in the new relation is increased by the vote amount of the duplicate. If any prop values have been defined for any attribute in the relation, these values are also accumulated in the new relation. The old tuple is then discarded.

The result is the prime relation, a relation in which no attribute has more distinct values than are allowed by the attribute threshold.

2.5.3 Further Generalization

If the number of tuples in the prime relation is less than the table threshold, the generalization process is complete. If not, further generalization must be accomplished. This generalization process proceeds by choosing for generalization the attribute with the most distinct values. Alternate selection criteria are discussed in Section 3. After each round of further generalization, duplicates are removed and another attribute is selected for generalization. When the total number of remaining tuples is less than or equal to the table threshold, automatic generalization ceases. If requested by the user, generalization is continued beyond the table threshold.

2.5.4 T-threshold Noise Reduction

During this generalization of the prime relation to the final relation, noisy, exceptional cases may be removed from the generalized relation if a t-threshold has been specified by the user (see Section 2.2.3).[31] In the current version, when the total number of tuples falls below twice the table threshold, all tuples with a proportion of votes less than the t-threshold are removed.[32]

2.5.5 Discriminate Task Reduction

For characteristic learning tasks, the process is complete at this point, but not for discriminate tasks. Two relations exist for discriminate tasks, the target and contrast relations. The purpose of the discriminate learning task is to find interesting distinctions between two sets of data. Any tuples, therefore, in the target and contrast relations that contain the same information will not contribute to this goal and must be noted. The two relations are therefore compared row for row and those which are the same are marked as non-distinct. These tuples will not be displayed in the final output.

The only exception to this is if a d-threshold has been specified (see Section 2.2.3). If a d-threshold has been specified by the user, those marked tuples in the target relation that have a vote percentage higher than the threshold value are displayed with the unmarked tuples.

The results of the learning procedure are stored as a pointer to the final generalized relation. This pointer may be accessed by the user through the command module for display.

This concludes the overview of the architecture of DBLEARN and its methods of implementing the characteristic and discriminate learning algorithms. The following section will describe some possible changes to the system that might improve its design or add to its capabilities.

3 General Improvements to DBLEARN

In this section, we present a set of planned improvements to DBLEARN. The improvements relate to four of the five modules.

3.1 Interface Development

In general, the responsibility for display should be completely relegated to the interface modules. All output should be extracted from the other modules, and the structures built by these modules should be made available to the interfaces for specialized display.

3.1.1 Command Line Expansion

As long as the command line interface remains a part of the program, some changes to the method of defining tasks should be made and some additional features should be added. The discrepancies between the discussion in the previous section and the actual implementation, which should all be updated, are not included below.

Reuse of Task Results If a copy of the prime relation were kept from the completed learning task, the second stage of generalization could be repeated with different reduction strategies.

Reuse of Tasks The ability to save a task definition during a session and later read that definition would be handy for repetitive activities. For example, from the command-line, one might enter:

```
save myfile
```

which would save the most recent task definition, including any applicable defaults, in the specified file. Entering

```
read myfile
```

would cause the stored task to be read from the file and executed. This would save on task entry time and would reduce the chance of error. At present, the best one can do is store the complete command-line input in a file and run DBLEARN with input from that file.

3.1.2 Graphical User Interface Development

The initial needs of the graphical user interface are more tied to the structure of the command and learning modules. Currently, the learning module handles most of the individual tasks that should be handled by the command module, such as building the concept hierarchy trees and initializing the database and schemas. In addition, the learning module handles the transition from initial generalization to further generalization. These should be extracted to the command module level. This would solve the current problem of no intermediate communication between the learning module and the GUI. It would also make error correction easier for the command line interface.

3.2 Database Module

The database module suffers primarily from inefficiency. Where DBMS's automatically optimize queries, this may not be a major issue, but some DBMS's do not do this and the results are blatantly noticeable. The primary problem is in the building of the database query. This is actually handled by the concept hierarchy module at the current time, but is included under this section since it is so database specific.

3.2.1 Better Extraction of Generalized Concepts

When the user uses a high level concept to specify a subset of data to extract from the database, this is currently built into complex, inefficient queries. There should be better use of the IN function in a select statement. The high level predicate:

```
where province = "Prairies"
```

is currently translated into something like:

```
where province = 'Alberta'  OR
      province = 'Saskatchewan'  OR
      province = 'Manitoba'
```

It would be much more efficiently and easily translated into:

```
where province in ('Alberta', 'Saskatchewan', 'Manitoba')
```

Similarly, the range value high level query:

```
where discipline_code = "Computer"
```

is currently translated into something like:

```
where (discipline_code >= 23000 AND discipline_code < 23500) OR
      (discipline_code >= 23500 AND discipline_code < 24000) OR
      (discipline_code >= 24000 AND discipline_code < 24500) OR
      (discipline_code >= 24500 AND discipline_code < 25000)
```

It would be much more efficiently translated as:

```
where (discipline_code >= 23000 AND discipline_code < 25000)
```

Some additional information in the concept hierarchy trees would make this easier. If the lower and upper bound values of range values could be propagated up the tree as the tree is being built, then concept translation would be a simple matter.

3.2.2 Better Ordering of Joins

Much more important than this though is the ordering of queries. Where multiple tables are being joined to perform a query, the tables need to be reduced in size as much as possible before the join. In one test query, reordering the table reduction before the join reduced the time necessary for the query from over 30 hours for an incomplete result to just a few minutes. The number of tuples in the query that DBLEARN built would have been over 45 million. The number in the better ordered query was about 150,000. The table reductions that can be performed before a join can be detected by the type of argument to the right side of a comparison operator. In DBLEARN syntax (and possibly in SQL), the left value of a comparison expression (such as "`disc_code = Computer`") is an attribute name. Where the right side are also attribute names, these expressions must be applied after a join of some sort. All other comparisons can be applied to a single table to reduce the size of that table before the join.

3.2.3 Earlier Deletion of Ungeneralizable Attributes

In the current version, the initial relations are read in and then generalized. When attribute oriented generalization is being performed, if one attribute does not have a concept hierarchy defined for it, it is deleted from the tables. This involves a fair amount of computational time. This should be detected before the relations are even read in and the initial query adjusted even before the database is accessed to not even read the attribute in the first place.

3.3 Concept Hierarchy Module

3.3.1 More Conceptual Hierarchy Definition File Structure

The structure of a concept hierarchy file that is similar to how we display the concept hierarchy files in text format would be powerful and easier to build. Multiple hierarchies could still be defined in the same file. The different levels of the tree could be defined by tabbing in one tab space for each level of the tree. A symbol found on a line with no tabbing would signal an attribute name and the beginning of a concept definition. Our file for province definition could look like:

```
province
      Western
            British Columbia
            Prairies
                  Alberta
                  Saskatchewan
                  Manitoba
      Ontario
      Quebec
      Maritimes
            New Brunswick
            Nova Scotia
            Prince Edward Island
            Newfoundland
```

The ANY symbol is not required since every concept hierarchy has one. This would be much easier to define than the current style which is error prone and hard to understand.

3.3.2 Expansion of Range / Non-range Concepts

The current definition of range values only includes numeric based attributes, but this is not conceptually as powerful as it could be. Dictionary entries and Dewey Decimal codes are examples of strings or alpha-numeric codes that have range properties. Similarly, university course numbers are numeric values but do not necessarily have the range property of inherent ordering. If this understanding is adopted, the range type concept hierarchy trees would need to be explicitly declared as such, not detected on the basis of attribute type. This would require an addition to the concept file definition structure.

As mentioned above, where range values are defined for leaf nodes, it would be advantageous to propagate these values up the hierarchy tree so that every node has an upper and lower bound. This would make high level concept translation much easier.

3.4 Learning Module

We have already mentioned the need to maintain a copy of the prime relation for reuse. This would allow for experimentation by the user of different further reduction strategies.

The ability for the user to control the method of further generalization is also needed in the learning module. Currently the only strategy used to choose the attribute to generalize is the one with the most distinct values. Two alternate selection criteria are the highest reduction ration and lowest reduction ratio. The attribute with the *highest reduction ratio* is that attribute which, when generalized one level, would cause the table to be maximally reduced in size. Conversely, the *lowest reduction ratio* corresponds to the the least reduction in size. As well, the user should be allowed to explicitly choose an attribute for further generalization. In this case, the choices should be relayed to the user through the command module and the user's choice returned.

4 Conclusion

This paper has described DBLEARN and its architecture as it is and as it should be. The current version of DBLEARN is faster, easier to maintain, and easier to extend than the orginal prototype. The further reorganization of DBLEARN to match more closely the design given here should continue to increase the maintainability and extendibility of the software. As well, the suggested improvements to the implementation of DBLEARN that are given in Section 3 may further increase DBLEARN's overall usefulness.

References

[Cai et al., 1991] Cai, Y., Cercone, N., and Han, J. (1991). Attribute-oriented induction in relational databases. In *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, Cambridge, MA.

[Carter and Hamilton, 1994] Carter, C. and Hamilton, H. (1994). Performance improvement in the implementation of DBLEARN. Technical Report 94-5, Department of Computer Science, University of Regina, Regina, Sask., Canada. 10 pages.

[Frawley et al., 1992] Frawley, W., Piatetsky-Shapiro, G., and Metheus, C. (1992). Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):57–70.

[Han et al., 1992] Han, J., Cai, Y., and Cercone, N. (1992). Knowledge discovery in databases: An attribute-oriented approach. In *Proc. of 18th Int'l Conf. on Very Large Data Bases*, Vancouver.

[Han et al., 1993] Han, J., Cai, Y., and Cercone, N. (1993). Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(1):29–40.

[Huang and Cai, 1993] Huang, Y. and Cai, Y. (1993). A tutorial on the DBLEARN system. School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada.

Endnotes

1. Currently, this modularity is not complete. Because DBLEARN was originally implemented for a command line interface with text screen output, much output to stdout and stderr remains scattered in the code. This needs to be extracted and handled by the user interface modules. All other modules should be rewritten to pass back data structures from their tasks. It will be the interface's responsibility to do all display in whatever form is available. This will make the task of updating the graphical user interface to handle dynamic exchange with the learning component much easier.

2. This is a documented feature of DBLEARN, but we have not verified whether or not it works in the current version.

3. A list box is a graphical box containing a list of entries, one of which may be selected by keyboard entry or a mouse click. The selected item is then entered into a special selection cell at the top of the box. A drop down list box appears at first to be just a single text cell with a down arrow to the right of it. When the arrow is clicked with a mouse, the box of entries drops down in a similar fashion to a drop down menu. When an item is selected, that item replaces the one in the selection cell and the drop down list disappears.

4. A standard list box may be single or multiple select. A single select list box allows only one entry to be selected at a time and is appropriate for a drop down style since the single selection may be displayed in the select cell that is left after the list box disappears. A multi-select list box allows as many entries to be selected as the user desires, displaying selected items in reverse video. It is more appropriate to have at least several lines of this type of list box displayed at all times, allowing the user to see which items are selected. If more items are anticipated than there is space for in the display structure, the list may be scrollable. Scrollable list boxers have a scroll bar to the right of the entries which allows the user to scroll up or down through the list in a similar fashion to scrolling through a word processing document. A pop-up version of this is a dynamically created list box which allows the selection of a number of items and then when an OK button is clicked, the list box disappears.

5. A correlation name is a short name or letter associated with a relation which replaces the name of the relation in a subsequent where clause. For example, a query might be constructed as follows:

```
select amount, province from award, organization
where award.org_code = organization.org_code
```

However, this involves more typing than the average user wants to do. Rather, correlation names 'a' and 'o' are added to the relation names award and organization and the query is rewritten:

```
select amount, province from award a, organization o
where a.org_code = o.org_code
```

which is much more efficient.

6. A push button is a simple button shaped object which initiates a single action when selected with a mouse.

7. A browsable file list is one where a lists of directories and files are separately displayed. When a directory is double clicked, its subdirectories are displayed in the directory list and the files in it are displayed in the file list. In this way the directory structure may be explored and files located easily. A multi-select version allows any number of files to be selected, though at the current time it is limited to files in the same directory. It would be useful for these files to be scanned for the individual concept hierarchies that they contain. If only one hierarchy for each target attribute that has been selected exists in the file list, then they can be automatically loaded. If there happens to be any duplicates, a choice should be given to the user of which is to be loaded.

8. Currently all attributes are presented, not just numeric ones.

9. A check box is a simple boolean selector presented as a small box in which a check appears when it is selected with a mouse. Subsequent selections toggle between checked and unchecked states.

10. A text entry box is a simple boxed area of the screen which when clicked allows editable text entry.

11. The concept hierarchies should be built and database query initialized as soon as the information for them is defined. This would allow for better error checking and correction.

12. The entry of the result into the text window is actually accomplished by a dirty patch. The window structures are not configurable to be able to receive input from stdout, but the DBLEARN currently sends all of its output to stdout. To get around this, stdout is redirected to a file and then when the learning task was finished, that file is read into the text window.

13. These features are not implemented yet, though the menu items for them are available. The methods available for the edit window object include saving and printing the contents of the text area, so the functions would be easily added.

14. Structure returns are not currently implemented. In the present version, the functions themselves are geared to the yacc interface and will simply print the information directly to the output stream. However, if the general concept is followed of requesting

information and receiving back a structure that is then passed to whatever user interface is available, this will make dynamic exchanges between various interfaces much more distinct, and the user interfaces more modular. The individual user interfaces themselves would be fully responsible for all display and output issues.

15. The relations themselves are not retrieved at this point even though the current version retrieves the complete relation before generalization. Rather, the learning module is given the direct ability to retrieve the data one tuple at a time. This will facilitate the implementation of "on the fly" generalization at a later time.

16. The query building phase is currently handled by the learning module after the complete task has been defined. This needs to be extracted since the input to a learning task is not the information to build a relation for generalization, but the relation itself. This extraction would enable better error handling and less need for re-entry of information from the command line interface especially. When the query of the current yacc interface fails, the whole task must be retyped.

17. These functions are currently not handled by the command module, but by the learning module which directs all the loading of concept hierarchies and translation of high level concept to low level attribute values. These should be extracted from the learning module as they are not inherently part of the learning task, but one of the input elements to them.

18. We have not yet examined this function in `adjust.c` to determine how it works or what it really does. It is mentioned in [Huang and Cai, 1993].

19. The current version also contains a function for controlling output level. The "set demo 1" command will cause the output of intermediate information in various stages of the generalization process. This is needed since different tasks handle their own output of information. With this more modular design, however, the user interface is responsible for all display of information and has access to all the intermediate information that is normally printed out as part of the expanded output function. The output of more information is therefore a unitary function of the interface and not DBLEARN as a whole. As such, the output level can be set and handled by the interface alone without the command module's intervention. Similarly, two titles are assigned to the target and contrast relations. These are simply descriptive names to be used in the display of the results of generalization, and fall under the same type of category as the "demo" flag. These also can be handled by the interface module.

20. Currently the learning procedure takes the learning process from start to finish. We propose to divide this into two parts, the initial generalization to the prime relation where every attribute has been reduced to below the attribute threshold, and an iterative further generalization where the prime relation is reduced to more and more generalized states until the final generalized relation is obtained. This will allow the necessary communication between the learning module and the user interface, especially the graphical user interface which cannot (and should not) be accessed from any

module except the command module. Since the present version assumes the availability of stdout, the learning module handles its own interaction with the user, but this violates the modularity that we are trying to build into the system.

21. Two variables in the current version, `minor` and `major`, correspond to the t-threshold and d-threshold described by Han et al. ([Han et al., 1993], p. 37). These, however, are not settable except by hard coding a value into them. The variable `minor` is used for pruning low vote tuples, but `major` is not used at all.

22. Currently no provision is included for d-thresholds. Only those tuples which represent 100% of examples are included in the final generalized relation for discriminate tasks.

23. In the current version, the prop votes amount is actually calculated dynamically when the relation is printed out by simply dividing the vote amount by the total number of original tuples in the relation. This is a function beyond the scope of just printing a relation and might conceptually perhaps be better left to the generalization process which could set a field in a tuple structure to a percentage value.

24. Errors might be defined as fatal occurrences that will not allow the generalization process to proceed. This could be like a loss of database connection. A warning would signal a non-fatal but unusual occurrence that the user should be aware of. This might be like a value being encountered in the initial relation for which there is no concept hierarchy defined. A strategy for responding to this condition must be devised. One possibility to signal these types of errors would be to build a linked list of return structures that note each exceptional non-fatal occurrence. The tuple that contained the condition could then just be ignored for the time being.

25. A variable could be set that specifies the further reduction to proceed immediately using a particular predefined reduction method. This would allow the whole task to be unitary instead of iteratively interactive.

26. A help command is documented in the yacc version, but it doesn't do anything.

27. There are other more complex issues involved here concerning client / server connections and multiple DBMS access. The current version, however, has not addressed these issues.

28. This of course could be more efficiently be represented as:

    ```
    where discipline_code >= 23000 and discipline_code < 25000
    ```

    but this is not done in the current version. Some database optimization ideas will be presented in Section 3.

29. Currently no error checking is done.

30. It would be much more efficient to perform generalization on the fly as tuples are read in.

31. D-thresholds are not implemented in the current version.

32. This code is actually commented out right now.