A Heuristic for Evaluating Databases for Knowledge Discovery with DBLEARN

David Fudger*
Information Systems Department, City of Regina
Regina, Sask., Canada S4P 3C8
fudger@cs.uregina.ca

Howard J. Hamilton†
Department of Computer Science, University of Regina
Regina, Sask., Canada S4S 0A2
hamilton@cs.uregina.ca

Abstract

We propose a heuristic method for choosing databases for attempting knowledge discovery. The DBLEARN knowledge-discovery program uses an attribute-oriented inductive-inference method to discover potentially significant relations in a database. A concept forest defines the possible generalizations that DBLEARN can make for a database. The concept forest consists of trees, each of which represents a concept hierarchy for one attribute. We propose that the potential for discovery in a database be estimated by examining the complexity of its concept forest. One measure which has proven useful is the based on the depths and heights of all the interior nodes in their trees. Higher values for this measure indicate more complex concept forests, and thus, we believe, more potential for discovery. Given several databases and their concept forests, we rank them according to a heuristic measure, and recommend that DBLEARN be applied to those with the highest values.

1 Introduction

We propose a heuristic method for evaluating a relational database with regard to DBLEARN's potential for knowledge discovery in that database. *Knowledge discovery* is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [2]. DBLEARN is a knowledge-discovery system that summarizes information in a database based on a user-defined concept hierarchy for each attribute [1, 5, 6]. Our heuristic method is based on estimating the complexity of the concept hierarchies using a heuristic measure. A preliminary version of this paper appeared in [4].

*Current address: Information System Department, Brick Warehouse Corp., 11411 - 170 Street, Edmonton, Alberta, Canada.

†Supported by Natural Sciences Engineering Research Council of Canada (NSERC) Research Grant OPG0121504.

DBLEARN and other knowledge-discovery systems are being developed to address the rapid accumulation of computer-readable data [2]. DBLEARN combines the learning-from-examples paradigm from Machine Learning with database operations to extract knowledge from existing databases. It uses an attribute-oriented inductive-inference method [1] to summarize data in a database according to concept hierarchies for the attributes. The concept hierarchy for an attribute provides a tree which can be ascended until an appropriate level of generality is found. DBLEARN has been applied in controlled settings on several databases, including the NSERC Grant-Information Database [5] and the City of Regina water-meter database [3].

The three inputs needed for DBLEARN are:

1. *Base Data*: the unconditioned data present in a database;
2. *Learning Task*: specification of a subset of the base data and parameters on learning; and
3. *Concept Hierarchies*: expert-supplied domain knowledge about potential generalizations.

Typically the strategy for automatically analyzing a database with DBLEARN consists of the following steps: identify a target database, devise concept hierarchies for this database, and then generalize the data (based on the concept hierarchies) in the hopes of identifying new and useful information. The generalization method used is *attribute-oriented inductive inference*, whereby specific data values in tuples are repeatedly replaced by more general values, giving a reduced set of distinct tuples, until the threshold number of tuples is reached. The *tuple threshold* (or *table threshold*) is specified as part of the learning task. As implemented in the DBLEARN program, this generalization method requires large amounts of memory and processor time for a large database. This strategy for knowledge discovery from databases works well if:

1. a specific database has been identified or the database is to be chosen from a small set of databases, and
2. the DBLEARN program can interface with the database package (e.g., Sybase) of the database.

Often organizations have many data stores varying in structure, size and proprietary environments. Some organizations do not have the resources to analyze their databases in a random order. This paper proposes a heuristic method for measuring the complexity of the concept hierarchies for a set of databases and ranking the corresponding databases according to their potential for discovery. This heuristic has been implemented as a computer program that, given a list of the names of files (each containing the concept hierarchies for one database), produces a ranked list of the databases and their scores according to a selected measure. The ranked list indicates the order in which the databases should be analyzed to maximize the chance of making interesting discoveries soon.

The remainder of this paper is organized as follows. In Section 2 we describe concept hierarchies, and in Section 3 we describe our intuitions about evaluating the complexity of concept hierarchies. In Section 4 we present the evaluation procedure and analyze a heuristic measure for evaluating concept hierarchies. Finally, we draw conclusions in Section 5.

2 Concept Hierarchies

A *concept hierarchy* is a collection of generalization relations for an attribute in a database. Following [1], we define a *generalization relation* to be a relation between an exhaustive set of attribute values and a single higher-level, more-general value. A generalization relation can be represented by $\{A_1, ..., A_k\} \subset B$, which indicates that B is a generalization of each A_i, for $1 \leq i \leq k$. All concept hierarchies for a database are obtained from domain experts and stored together in a *concept hierarchy table* (also called a *concept hierarchy file*). For example, a concept hierarchy table for a university's enrolled student data might include the following generalization relations:

$$
\begin{array}{rcl}
\{\texttt{Chem220}, \texttt{Chem240}, \texttt{Chem300}, \texttt{Chem400}\} & \subset & \texttt{Chemistry} \\
\{\texttt{Mathematics}, \texttt{Chemistry}, \texttt{Biology}, \texttt{Physics}\} & \subset & \texttt{Science} \\
\{\texttt{English}, \texttt{French}, \texttt{Philosophy}, \texttt{Sociology}\} & \subset & \texttt{Arts} \\
\{\texttt{Arts}, \texttt{Science}\} & \subset & \texttt{Course} \\
\{\texttt{1st year}, \texttt{2nd year}, \texttt{3rd year}, \texttt{4th year}\} & \subset & \texttt{Undergraduate} \\
\{\texttt{Masters}, \texttt{PhD}\} & \subset & \texttt{Graduate} \\
\{\texttt{Undergraduate}, \texttt{Graduate}\} & \subset & \texttt{Student}
\end{array}
$$

Each line describes one generalization relation. The first four lines describe the `Course` concept hierarchy and the last three lines describe the `Student` concept hierarchy.

A convenient way to think of a concept hierarchy for an attribute is as a tree whose arcs (links) have all been reversed. In a tree, all arcs go from the root node towards the leaf nodes, but in a concept hierarchy, all arcs go from the "leaf nodes" toward the "root node" (the single sink of the directed acyclic graph). To simplify presentation, we refer to a concept hierarchy as a type of tree and we ignore the reversal of the links. Also, we refer to the set of concept hierarchies for a database as a *concept forest*. The attribute values in the database correspond to the leaf nodes of the tree. Concept hierarchies for one or more attributes in the database are required by DBLEARN if any generalization is to be performed.

Concept hierarchies are essential to DBLEARN's generalization process. DBLEARN first builds a table of relevant data from a database. Each attribute is generalized by ascending a specific tree until the distinct number of values for that attribute is less than the user-specified *attribute threshold*. After all attributes have been generalized, duplicate tuples are removed and the attributes with the most distinct values are generalized further until the number of tuples is less than or equal to the user-specified tuple threshold. The remaining tuples are called the *final generalized relation*. When generalization stops, DBLEARN can be said to have a set of active nodes in each of the concept hierarchies, where an *active node* is a node corresponding to a generalized attribute value that appears in at least one of the tuples in the final generalized relation. For example, the final generalized relation might contain the following (generalized) tuples

```
Chemistry       Undergraduate      14%
Chemistry       Graduate            2%
```

(where the final column identifies the percentage of the original tuples summarized in this generalized tuple). Here, `Chemistry` is an active node in the `Course` concept hierarchy and `Undergraduate` and `Graduate` are both active nodes in the `Student` concept hierarchy. `Chem220`, `Chem240`, `Chem300` and `Chem400` are **not** active nodes in the `Course` concept hierarchy because generalization has proceeded to the ancestor of these nodes. For more details on the method used in DBLEARN, see [5, 7].

3 Complexity of Concept Hierarchies

Let us consider the relationship between the concept hierarchies and the potential for DBLEARN to find interesting results. If more concept hierarchies are present, DBLEARN has more opportunities to perform generalizations. Also, a more complex tree affords more nodes that may be active and thus may be mentioned in the conclusions. As well, an active node that is farther from the leaf nodes is conceptually farther from the base values for the attribute. Finally, an active node that is farther from the root node allows more specific results to be stated; more specific results are less likely to be "platitudes," i.e., general statements about general classes. For all these reasons, if the concept hierarchies are more complex, more interesting results are likely to be found.

Let us consider measures of complexity for concept hierarchies. A single concept hierarchy can be regarded as a tree. No generally accepted measure of the complexity of a tree is used in the data structures literature. Two obvious candidates are:

- the number of nodes, and
- the depth of the tree, i.e., the length of the longest branch.

One example for each of these measures will be sufficient to illustrate why neither is suitable for our application.

First, consider a wide, shallow concept hierarchy, i.e., a tree with many nodes but with all nodes at depth 1, such as that shown in Figure 1(a). We define the *depth of a node* such that the depth of the root node is 0, the depth of a direct descendant of the root node is 1, and so on. Here all base values

```
                                                      A
                                                  ___|___
                                                  B     C
                                                  |     |
                                                  D     E
                  A                               |     |
_________________|_________________               F     G
|   |   |   |   |   |   |   |   |   |             |     |
B   C   D   E   F   G   H   I   J   K             H     I
                                                  |     |
                                                  J     K

                 (a)                                (b)
```

Figure 1: Concept Hierarchies: (a) CH1, (b) CH2

generalize to the same value (A, corresponding to ANY). Regardless of how many nodes are present at depth 1, no interesting conclusions can be formed by DBLEARN, since any generalization will simply refer to ANY value. We regard any generalization to the root node as uninteresting because it indicates that the values of the attribute are simply being ignored.

Next, consider a deep tree with few children for each node, such as that shown in Figure 1(b). In this case there are few base values (only J and K in fact) and many levels of the tree with only one child for each parent. A generalization up either branch of this tree (other than the final generalization to A) does not produce any true generalization, since each higher value actually corresponds to the same number of base values. From this example, we see that any linear chain in a tree is uninteresting and should be compressed.

Next we turn to some less obvious examples. Figures 2 and 3 show the concept forests for two databases. Concept forest CF1 has 8 attributes (i.e., 8 separate trees), a maximum depth of 4, a total of 27 nodes, and 15 leaf nodes. Concept forest CF2 has 2 attributes, a maximum depth of 3, a total of 19 nodes, and 12 leaf nodes. Although CF1 is larger in all these aspects, CF2 seems to provide more complicated domain knowledge than CF1.

Since the leaf nodes represent the base values of the attribute, they do not provide opportunities for drawing interesting conclusions. The base values are present in the database, and any conclusion mentioning them is not based on "nontrivial extraction" and does not represent "implicit knowledge," as are required for knowledge discovery. Conclusions regarding the base values can be drawn using standard database `count` queries.

Since neither the root node nor the leaf nodes allow interesting results to be found, attention should be restricted to the other nodes. An *interior node* is a node that is neither the root node nor a leaf node. Concept forest CF1 has 4 interior nodes (M, S, T, U), while concept forest CF2 has 6 interior nodes (C,

```
   A    B    C    D               E             F    G    H
   |    |    |    |         ______|______       |    |    |
   I    J    K    L         |     |     |       P    Q    R
                            M     N     O
                      ______|_______
                      |            |
                      S            T
                 _____|_____    ___|___
                 |    |    |    |     |
                 U    V    W    X     Y
             ____|____
             |       |
             Z       AA
```

Figure 2: Concept Forest CF1

```
               A                           B
         ______|_______             _______|_____
         |            |             |           |
         C            D             E           F
      ___|___      ___|___       ___|___    ____|____
      |     |      |     |       |     |    |   |   |
      G     H      I     J       K     L    M   N   O
______|     |_____
|     |     |    |
P     Q     R    S
```

Figure 3: Concept Forest CF2

D, E, F, G, H).

In the next section, we describe a measure for evaluating the relative complexities of concept forests based on the ideas introduced in this section.

4 Evaluation Procedure and Heuristics

The following procedure is used for measuring the complexity of a concept forest.

1. Augment each concept hierarchy for an attribute until it is complete. A *complete concept hierarchy* includes generalizations for all possible values for the attribute. The augmentation may be accomplished best by adding an OTHER node as a default for any unexpected values.

2. Arrange each concept hierarchy in tree form, with ANY as the root of each tree.

3. Delete any node (other than the root) that has only one child and then make the child into a new child of the grandparent. This removes linear chains.

4. If the root node has only one child, remove the child, and make the grandchildren (if any) into children of the root.

5. Determine the complexity of the concept forest using the measure discussed below.

Let f be a concept forest consisting of k concept hierarchies (trees) $t_1, \ldots t_k$, with depths $d_1, \ldots, d_k$ respectively. Measure m_1 is calculated based on the number of interior nodes weighted by their depth and height in their tree. The *height* of a node in a tree is the **minimum** distance of that node from the nearest leaf node. The height of a leaf is 0, and the height of any nonleaf node is 1 more than the minimum height of any of its children. (This definition of *height* differs from the standard definition used in the data structures where the height of a non-leaf node is 1 more than the *maximum* height of any of its children.) We define:

$$m_1(f) = \sum_{t \in f} \sum_{i \in t} (d_i + h_i),$$

where d_i is the depth of node i in tree t and h_i is the height of node i. Higher values for $m_1(f)$ indicate more complex concept hierarchies.

For example, as shown in Table 1(a), measure m_1 evaluates to 13 on concept forest CF1 (see Figure 2), where the only interior nodes are `M`, `S`, `T`, and `U`. Also, as shown in Table 1(b), measure m_1 evaluates to 15 on concept forest CF2 (see Figure 3). Here, m_1 ranks CF2 as more complex than CF1 and suggests that DBLEARN would be more likely to make interesting discoveries in the database corresponding to CF2 than in the database corresponding to CF1.

node i	d_i	h_i	total
M	1	2	3
S	2	1	3
T	2	1	3
U	3	1	4
Total			13

(a)

node i	d_i	h_i	total
C	1	2	3
D	1	1	2
E	1	1	2
F	1	1	2
G	2	1	3
H	2	1	3
Total			15

(b)

Table 1: Evaluation of Measure m_1 on (a) CF1 and (b) CF2

5 Conclusions

DBLEARN relies heavily on the concept hierarchies to provide new knowledge based on the existing data. If the tree structure is short and simple, the data will generalize very quickly from many tuples to fewer than the tuple threshold, but few new and interesting discoveries will be made. This was demonstrated using small examples. If there are more distinct values to generalize to, the attribute-oriented inductive-inference method used by DBLEARN produces results with more specific values. By paying attention to the number of concept hierarchies, the number of interior nodes in these hierarchies, and the depth and height of each of these interior nodes, a measure of complexity was designed for concept hierarchies.

There are several reasons to use the measure described in this paper. First, the size of a concept hierarchy is very small compared to the size of a database. This allows for a very fast cursory analysis of the potential for knowledge discovery before DBLEARN is applied to the databases.

Secondly, as it exists today, DBLEARN must be ported to any proprietary platform where it is to be run, which may involve changes to DBLEARN to deal with a different operating system or a different database environment. The heuristic procedure proposed in this paper can be performed easily using a concept hierarchy file before any porting of DBLEARN is done. As well, m_1 could conceivably be applied to a paper description of the concept hierarchies for a database. The results of the heuristic procedure could be used to determine if porting DBLEARN to a new environment would be worth the effort.

Thirdly, the heuristic procedure could be used during the design of the concept hierarchies to guide the designers to more useful concept hierarchies. For example, if the heuristic procedure gives a low score to a concept forest, the domain experts should be consulted to try and refine the concept hierarchy further. However, aiming for a high score according to our procedure cannot replace the need for the concept hierarchy to describe useful relationships in the domain.

Finally, and perhaps most importantly, the measure described here can be used to direct the knowledge discovery process among many databases.

DBLEARN requires a large amount of memory space and can take several hours to analyze a relatively small database. Using our heuristic procedure, we can rank many databases by the level of complexity in their concept hierarchies, and then direct DBLEARN to the most likely candidates for discovery. This is especially useful when complete analysis of only a few databases is possible.

In ongoing work we are applying the heuristic described in this paper to a wide variety of academic and commercial databases.

References

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

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

[3] D. Fudger. Knowledge discovery in large databases: DBLEARN applied to a large commerical database. Department of Computer Science, University of Regina, Regina, Sask., April 1993.

[4] D. Fudger and H.J. Hamilton. Heuristic evaluation of databases for knowledge discovery with DBLEARN. In *RSKD'93: International Workshop on Rough Sets and Knowledge Discovery*, pages 29–39, Banff, Canada, October 1993.

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

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

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