Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases *

Jiawei Han and Yongjian Fu
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
{han, yongjian}@cs.sfu.ca

Abstract

Concept hierarchies organize data and concepts in hierarchical forms or in certain partial order, which helps expressing knowledge and data relationships in databases in concise, high level terms, and thus, plays an important role in knowledge discovery processes. Concept hierarchies could be provided by knowledge engineers, domain experts or users, or embedded in some data relations. However, it is sometimes desirable to automatically generate some concept hierarchies or adjust some given hierarchies for particular learning tasks. In this paper, the issues of dynamic generation and refinement of concept hierarchies are studied. The study leads to some algorithms for automatic generation of concept hierarchies for numerical attributes based on data distributions and for dynamic refinement of a given or generated concept hierarchy based on a learning request, the relevant set of data and database statistics. These algorithms have been implemented in the DBLearn *knowledge discovery system and tested against large relational databases. The experimental results show that the algorithms are efficient and effective for knowledge discovery in large databases.*

Keywords: Knowledge discovery in large databases, discovery methods, KDD system implementation, algorithms, dynamic generation and refinement of concept hierarchies.

1 Introduction

With the rapid growth in size and number of available databases in commercial, industrial, administrative and other applications [5], it is necessary and interesting to examine how to extract knowledge automatically from huge amounts of data. By extraction of knowledge in databases, large databases will serve as a rich, reliable source for knowledge generation and verification, and the discovered knowledge can be applied to information management, query processing, decision making, process control and many other applications. Therefore, knowledge discovery in databases (or data mining) has been considered as one of the most important research topics in 1990s by both machine learning and database researchers [17, 20].

There are different philosophical considerations on knowledge discovery in databases (KDD) [5, 23], which may lead to different methodologies in the development of KDD techniques [5, 11, 6, 18, 1, 21, 22, 23].

In our previous studies [2, 7, 8], an attribute-oriented induction method has been developed for knowledge discovery in relational databases. The method integrates a machine learning paradigm, especially *learning-from-examples* techniques, with database operations and efficiently extracts generalized data from actual data in databases. The method is based on the following assumptions in data mining.

First, it assumes that a database stores *a large amount* of *information-rich, relatively reliable* and *stable* data. The assumption on data reliability and stability motives the development of knowledge discovery mechanisms firstly in relatively simple situations and then evolution of the techniques step-by-step towards more complicated ones. Secondly, it assumes that a knowledge discovery process is initiated by a user's

*The research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and the Center for Systems Science of Simon Fraser University.

learning request. User-directed learning may lead to guided discovery with a focus on the interested set of data and therefore represents relatively constrained search for the desired knowledge. Thirdly, it assumes that generalized rules are expressed in terms of high level concepts for simplicity, conciseness and clarity. Fourthly, it assumes that background knowledge, such as conceptual hierarchies, etc., is generally available for knowledge discovery process. The availability of relatively strong background knowledge not only improves the efficiency of a discovery process but also expresses user's preference for guided generalization, which may lead to an efficient and desirable generalization process.

Following these assumptions, the attribute-oriented induction mechanism can be outlined as follows. First, a knowledge discovery process is initiated by a learning request, which is usually in relevance to only a subset of data in a database. A data retrieval process is initiated to collect the set of relevant data. Second, generalization is performed on the set of retrieved data using the background knowledge and a set of generalization operators. Third, the generalized data is simplified and transformed into a set of generalized rules for different applications. Clearly, background knowledge, especially concept hierarchies, plays an important role in efficient and successful induction in databases.

Concept hierarchies could be provided by knowledge engineers, domain experts or users, or embedded in some data relations. However, it is sometimes desirable to automatically generate some concept hierarchies or adjust some given hierarchies for particular learning tasks. In this paper, some algorithms are developed for automatic generation of concept hierarchies for numerical attributes based on data distributions and for dynamic refinement of a given or generated concept hierarchy based on a learning request, the relevant set of data and database statistics. Moreover, the methods for automatic generation of concept hierarchies for discrete-valued attributes are also considered in the discussion. The algorithms presented in this paper have been implemented in the DBLearn knowledge discovery system and tested against large relational databases. The experimental results show that the algorithms are efficient and effective for knowledge discovery in large databases.

The paper is organized as follows. In Section 2, concept hierarchy and its role in knowledge discovery in databases are introduced. In Section 3, algorithms for dynamic refinement of concept hierarchies are presented. In Section 4, algorithms for automatic generation of concept hierarchies for numerical attributes are presented. In Section 5, automatic generation of concept hierarchies for nominal data is discussed and the experiments on relational databases are explained. Our study is summarized in Section 6.

2 Concept hierarchy and its role in knowledge discovery in databases

2.1 Concept hierarchies

A concept hierarchy defines a sequence of mappings from a set of lower-level concepts to their higher-level correspondences. Such mappings may organize the set of concepts in *partial order*, such as in the shape of a tree (a hierarchy, a taxonomy), a lattice, a directed acyclic graph, etc., although they are still called "*hierarchies*" for convenience.

A concept hierarchy can be defined on one or a set of attribute domains. Suppose a hierarchy H is defined on a set of domains $D_i, \ldots, D_k$, in which different levels of concepts are organized into a hierarchy. The concept hierarchy is usually partially ordered according to a general-to-specific ordering. The most general concept is the null description (described by a reserved word "*ANY*"); whereas the most specific concepts correspond to the specific values of attributes in the database.

Formally, we have

$$H_l : D_i \times \cdots \times D_k \Rightarrow H_{l-1} \Rightarrow \cdots \Rightarrow H_0, \tag{2.1}$$

where H_l represents the set of concepts at the primitive level, H_{l-1} represents the concepts at one level higher than those at H_l, etc., and H_0, the highest level hierarchy, may contain solely the most general concept, "ANY".

Since concept hierarchies define mapping rules between different levels of concepts, they are in general data- or application-specific. Many concept hierarchies, such as *birthplace* (*city, province, country*), are actually stored implicitly in the database, such as in different attributes or different relations, which can be made explicit by specifying certain attribute mapping rules. Moreover, some concept mapping rules can be specified by deduction rules or methods (such as in object-oriented databases) and be derived by deduction or computation. For example, the floor area of a house can be computed from the dimensions of each segment in the house by a spatial computation algorithm, and then mapped to a high level concept, such as *small*, *large*, etc. defined by deduction rules.

The mappings of a concept hierarchy or a portion of it may also be provided explicitly by a knowledge engineer or a domain expert. For example, "*status: {freshman, sophomore, junior, senior}* $\rightarrow$ *undergraduate*", "*annual income: "{1,000, ..., 25,000}* $\rightarrow$ *low_income*", etc., can be specified by domain experts. This is often reasonable even for large databases since a concept hierarchy registers only the *distinct* discrete attribute values or *ranges* of numerical values for an attribute which are, in general, not very large and can be specified by domain experts. Furthermore, some concept hierarchies can be discovered automatically [4, 7].

Different concept hierarchies can be constructed on the same attribute(s) based on different viewpoints or preferences. For example, the birthplace could be organized according to administrative regions, geographic locations, size of cities, etc. Usually, a commonly referenced concept hierarchy is associated with an attribute as the default one. Other hierarchies can be chosen explicitly by preferred users in a learning process. Also, it is sometimes preferable to perform induction in parallel along more than one concept hierarchy and determine an appropriate representation based on later generalization results.

2.2 Attribute-oriented induction in relational databases

Since this study is on automatic generation and dynamic refinement of concept hierarchies in a knowledge discovery system, we first briefly introduce the attribute-oriented (A-O) induction method [2, 7] implemented in the DBLearn system [9]. The induction method was firstly presented in [2] and has been substantially enhanced in the later system development [9].

Different kinds of rules, including *characteristic rules*, *discriminant rules*, *data evolution regularities*, etc., can be discovered by the DBLearn system. A characteristic rule is an assertion which characterizes a concept satisfied by all or most of the examples in the class undergoing learning (called the target class). For example, the symptoms of a specific disease can be summarized by a characteristic rule. A discriminant rule is an assertion which discriminates a concept of the class being learned (the target class) from other classes (called contrasting classes). For example, to distinguish one disease from others, a discriminant rule should summarize the symptoms that discriminate this disease from others. A data evolution regularity rule is an assertion which describes the general properties of a set of data in the database which change with time.

A relation which represents *intermediate* (or *final*) learning results is called an *intermediate* (or a *final*) generalized relation. In a generalized relation, some or all of its attribute values are generalized data, that is, nonleaf nodes in the concept hierarchies. An attribute in a (generalized) relation is at a *desirable* level if it contains only a small number of distinct values in the relation. A user or an expert may like to specify a small integer as a *desirable* attribute threshold for an attribute. Such a threshold can also be set as a default by the system. In this case, an attribute is at the desirable level if it contains no more distinct values than its *attribute threshold*. Moreover, the attribute is at the minimum desirable level if it would contain more distinct values than the threshold if it were specialized to a level lower than the current one. A special generalized relation R' of an initial relation R is the prime relation of R if every attribute in R' is at the minimum desirable level.

To deal with exceptions and noises, each tuple in *generalized relations* is attached with a count which is the number of tuples in the original relation which are generalized to the current generalized tuple.

A set of basic techniques for the derivation of a prime relation in learning a characteristic rule are presented as follows [2, 7].

1. Attribute removal: If there are a large set of distinct values in an attribute of the working relation, but (1) there is no generalization operator on the attribute, or (2) its higher-level concepts are expressed in another attribute, the removal of the attribute generalizes the working relation.

2. Attribute generalization: If there are a large set of distinct values in an attribute in the working relation,

but there exist a set of generalization operators on the attribute, a generalization operator should be selected and be applied to the attribute at every step of generalization.

3. Count propagation: The value of the count of a tuple should be carried to its generalized tuple, and the count should be accumulated when merging equivalent tuples in generalization.

4. Attribute generalization control: Generalization on an attribute a_i is performed until the concepts in a_i has been generalized to a desired level, or the number of distinct values in a_i in the resulting relation is no greater than a prespecified or default attribute threshold.

Since the above induction process enforces only attribute generalization control, the prime generalized relation so extracted may still contain a relatively large number of generalized tuples. Two alternatives can be developed for the extraction of generalized rules from a prime generalized relation: (1) further generalize the prime relation to derive a **final generalized relation** which contains no more tuples than a prespecified *relation threshold*, and then extract the *final generalized rule*; and (2) directly extract **generalized feature table** and present feature-based multiple rules [7].

For Alternative 1, generalization on a prime generalized relation is performed until the number of distinct generalized tuples in the resulting relation is no greater than a prespecified relation threshold. At this stage, there are usually alternative choices for selecting a candidate attribute for further generalization. The interestingness of the final generalized rule relies on the selection of the attributes to be generalized and the selection of generalization operators. Such selections can be based on data semantics, user preference, generalization efficiency, etc. Many techniques developed in previous studies on machine learning [14], statistics [13], fuzzy set and rough set theories [22], etc. can be applied to the selection of attributes and operators. Interesting rules can often be discovered by following different paths leading to several generalized relations for examination, comparison and selection, which can be performed *interactively* by users or experts [23]. After this generalization, final generalized rule(s) can be extracted from a final generalized relation, where a tuple in the generalized relation is transformed to conjunctive normal form, and multiple tuples are transformed to disjunctive normal form [8].

2.3 The role of concept hierarchies in attribute-oriented induction

Concept hierarchy is essential in the attribute-oriented generalization. The following three important roles are played by concept hierarchies in attribute-oriented induction.

1. Retrieval of the relevant set of data: In the data retrieval process, a query constant could be specified at a general concept level, for example, the required GPA for an *undergraduate* (instead of "*freshman, ..., senior*") student could be *good* (instead of a concrete range). Concept hierarchies should be used to map the high level concepts into the constants at the concept level(s) matching those stored in the database.

2. Determination of generalization pairs in the derivation of the prime relation. Concept hierarchies should be used for concept tree climbing in the generalization process.

3. Further generalization of prime relations. Concept hierarchies will be used for further ascension of the concepts in the prime relation in order to achieve desired generalization results.

3 Dynamic Refinement of Concept Hierarchies

Although concept hierarchies can be provided by users or experts, a provided concept hierarchy may not be the best fit to a particular learning task. It is often necessary to dynamically refine or adjust an existing concept hierarchy based on the learning task, the set of relevant data and data distribution statistics.

Example 3.1 Suppose the database stores a concept hierarchy on geographic regions in the world. To find the regularities of the birth places of the undergraduate students in Simon Fraser University, it may be desirable to express the top level concepts as {*B.C., Other_Provinces_in_Canada, Foreign*}. On the other hand, to find the regularities of the birth places of the professors in the same university, the top level concepts may better be: {*North_America, Europe, Asia, Other_Regions*}. Such adaptation of different data distributions can be achieved by dynamic adjustment or refinement of concept hierarchies based on the set of relevant data. □

Example 3.1 indicates that dynamic refinement of concept hierarchies according to the distributions of the relevant set of data should be a regular practice in many generalization processes. At the first glance, dynamic adjustment/refinement of an existing concept hierarchy seems to be an overly complex process since it corresponds to dynamically regrouping the data, and its complexity grows exponentially to the size of the data. However, since the given concept hierarchy provides important semantic information about concept clustering, it is important to preserve the existing data partition as much as possible and perform minor refinements on the existing clustering, which will substantially reduce the total number of combinations to be considered.

The following observations may lead to the design of an efficient and effective algorithm for dynamic concept hierarchy refinement.

First, dynamic adjustment of concept hierarchies should not be performed during the collection of the set of relevant data (i.e., Step 1). This is because the data retrieval process involves only the mapping of higher level concepts in the query (or learning task) to their corresponding lower level data, which should be determined by the semantics specified in the existing concept hierarchy.

Secondly, concept hierarchy adjustment is a highly dynamic process. The next learning task may have different relevant set of data with different data distribution which may require the hierarchies to be adjusted differently from the current task. Therefore, an adjusted hierarchy is usually not saved for future usage.

Thirdly, it is often desirable to present the regularities by a set of nodes (i.e., which are usually "generalized" attribute values) with relatively even data distribution, i.e., not a blend of very big nodes and very small ones at the same level of abstraction. Thus, it is desirable to promote the "big" (carrying substantial weight or count) low-level nodes and merge the tiny nodes when presenting generalization results.

Finally, although a concept hierarchy could be quite deep, only the concepts at the levels close to or above that at the prime-relation are interested to users. Therefore, the adjustment of concept hierarchies can be focused at the level close to the prime relation without considering a complete adjustment of the entire hierarchy.

Based on the above observations, we introduce some new terminology and present the algorithm.

Definitions 3.1 A *concept hierarchy* consists of a set of *nodes* organized in a partial order. A node is a *leaf node* if it has no child(ren), or a *nonleaf node* otherwise. An *occurrence count* of a node is a number associated with the node, representing, if a leaf node, the number of occurrences of the value in the task-relevant data set, or if a nonleaf node, the sum of the occurrence count of its children nodes. A *total occurrence* of an attribute is the sum of the occurrence counts of all the leaf nodes in the initial data relation.

The dynamic hierarchy adjustment algorithm is presented as follows.

Algorithm 3.1 (Dynamic concept hierarchy adjustment) *Dynamic adjustment of concept hierarchies for attribute-oriented induction based on data distribution of an attribute in the initial data relation.*

Input. (i) A learning task-relevant initial relation $\mathcal{W}_0$, (ii) an attribute A, (iii) the *attribute threshold* T for attribute A, and (iv) a prespecified concept hierarchy H.

Output. An adjusted concept hierarchy H' of attribute A for the derivation of the prime relation and for further generalization.

Method. The adjustment essentially consists of two processes: top-down "big" nodes promotion and bottom-up "small" nodes merging.

1. Initialization:
   (a) Assign the level number to each node in the hierarchy H according to the given partial order;

(b) Scan once the corresponding attribute of each tuple in the initial relation $\mathcal{W}_0$, calculate the occurrence count $c_i.count$ for each leaf node c_i, and propagate them to the corresponding parents in the hierarchy H. The *total_occurrence* is the sum of the counts of all the leaf nodes in the hierarchy. Notice only the nonzero count nodes are considered in the following computation.

2. Top-down adjustment of concept hierarchy H.

   (a) Set a buffer set, *Prime*, initially empty, and another buffer set, *Buff*, to hold the set of nodes at the top-level of H.

      i. Calculate the weight of each node c_i, $c_i.weight := c_i.count/total\_occurrence$.

      ii. Set weight threshold τ, $\tau := 1/T$.

      iii. Perform node marking: A node, if weighted no less than τ, is a big node; otherwise, a small one. A big leaf node is marked B, a big nonleaf node is marked B', a small leaf node is marked S, and a small nonleaf node is marked S'.

   (b) Call *expand_buffer*, which is implemented as follows.

      i. Move every B-marked node from *Buff* to *Prime*;

      ii. Replace every B'-marked node by its children;

      iii. Repeat this process until no change (i.e., only the nodes marked S or S' are left in *Buff*).

   (c) Perform weight re-calculation and node re-marking again as following.
   If $|Prime| + |Buff| \leq T$, move all the nodes from *Buff* to *Prime*, and the process terminates. Otherwise, set T' to $T - |Prime|$, $total'$ to the sum of the counts in *Buff*, $weight'$ of each node in *Buff* to $count/total'$, and $\tau' := 1/T'$. Mark the node based on the $weight'$ and τ' and repeat the *expand_buffer* and weight re-calculation processes until no change.

3. If there are still nodes left in *Buff*, perform bottom-up merging of the remaining nodes in *Buff* as follows.

   Starting at the bottom level, step up one level (suppose, to level i) and merge the nodes in *Buff* which share a common ancestor at level i. If the $weight'$ of the merged node is no less than τ', move it to *Prime* (and decrement T'). If the total number of nodes in *Buff* is no more than T', then move all the nodes in *Buff* to *Prime*, else perform weight re-calculation, step up a level, and repeat the process. If there is no more level to climb (the hierarchy is in the shape of a forest), group the nodes into T' groups and move them to *Prime*.

   We have the following convention for naming a merged node. Name a node $A+B$ if it is the result of merging two nodes A and B. Otherwise, name it $E - A$ if it is equivalent to an existing node E with one child node A removed. Otherwise, name it *Other_E* if it is equivalent to an existing node E with more than one child node removed.

4. Build up the generalization linkage between the nodes in *Prime* and the attribute data in the initial relation. □

Theorem 3.1 *There are no more than T (attribute-threshold) nodes in Prime, and there exists a generalization linkage between every node in the initial relation and a node in the prime relation after the execution of Algorithm 3.1.*

Rationale. According to the algorithm, every node moved into *Prime* must satisfy one of the following three conditions: (1) a node with a weight greater than τ or τ', (2) when $|Prime| + |Buff|$ is no more than T or T', or (3) the reamining nodes are grouped into T' groups (i.e., T' new nodes) when there is no more level to climb. Moreover, the computations of T', τ and τ' ensure that the number of the accumulated nodes is no more than T. Thus the algorithm cannot generate more than T nodes in *Prime*. Also, every non-zero count node is either a leaf node moved into *Prime*, or is associated with a nonleaf (ancestor) node that is finally moved into *Prime* according to the algorithm, there should exist a generalization linkage from the node to a node in the prime relation after the execution of the algorithm. □

Furthermore, Algorithm 3.1 is designed based on the consideration that the nodes in the *Prime* relation should carry relatively even data distribution, and the shape of the hierarchy should be maximally preserved. Therefore, hierarchy adjustment following the algorithm should produce desirable results.

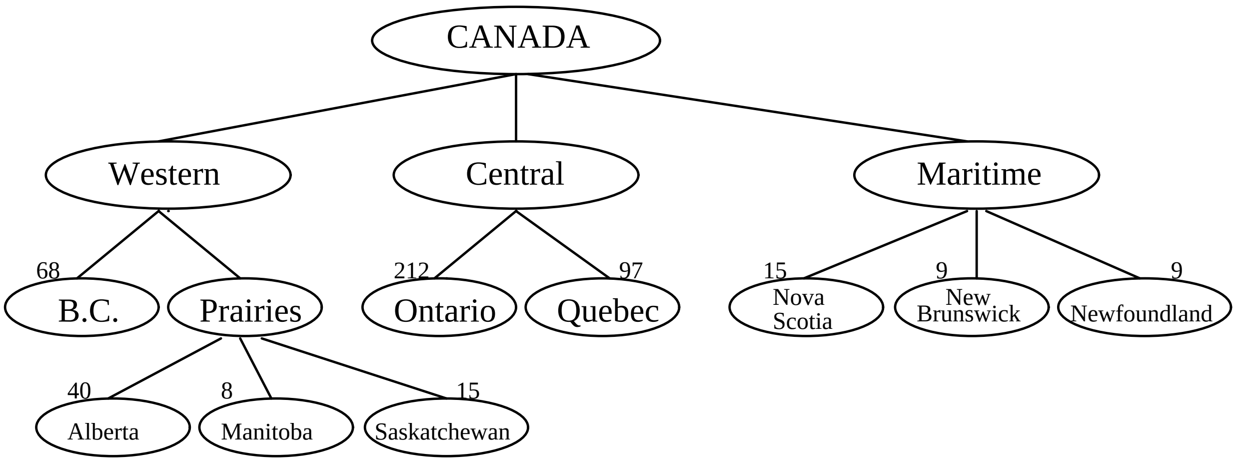

Figure 1: Original concept hierarchies for *provinces*.

Example 3.2 The original concept hierarchy for attribute "*Province*" (in Canada) is given, as shown in Fig. 1. The learning query is to find a characteristic rule of the NSERC Operating Grants in Computing Science in relevance to provinces, and the attribute threshold for "*Province*" is 7. Using the original hierarchy *without* dynamic adjustment, the derived prime relation consists of 7 values in the attribute: {*British Columbia*(68), *Prairies*(63), *Ontario*(212), *Quebec*(97), *New Brunswick*(15), *Nova Scotia*(9), *Newfoundland*(9)}, (where each number in parentheses indicates the number of CS grants in the corresponding province), which corresponds to 7 level-three nodes in Fig. 1. This is undesirable since the level-four node "*Alberta*" has count 40, whereas each Maritime province (at level-three) has much smaller counts. Notice that some nodes, such as *Ontario*(212), are leaf nodes which, though quite big, cannot be split further. Following Algorithm 3.1, the dynamic adjustment of hierarchy is performed based on the current learning task and node counts. This results in Fig. 2, in which "*Alberta*" is promoted, and the maritime provinces are "merged". The attribute in the prime relation consists of 6 nodes: {*British Columbia*(68), *Alberta*(40), *Sas+Man*(23), *Ontario*(212), *Quebec*(97), *Maritime*(33)}, with a relatively even distribution among all the nodes at the prime relation level. □

4 Automatic Generation of Concept Hierarchies for Numerical Attributes

Numerical attributes occur frequently in data relations. Concept hierarchies for numerical attributes can be generated automatically by the examination of data distribution characteristics. The following two standards are used for automatic generation of concept hierarchies for numerical attributes.

1. *Completeness*: The value ranges of the hierarchy of a numerical attribute should cover all of its values in the set of data relevant to the current learning task.

2. *Uniformity*: The set of ranges presented in the prime relation should have relatively even distribution based on the frequency of occurrences of the attribute values in the set of data relevant to the current learning task. This is based on that people would usually like to compare the concepts with relatively even distributions in the relevant data set.

Example 4.1 Suppose the learning task is to study the characteristics of Canadian scientific researchers in relevance to provinces, the amount of operating grants received, and their ages. The latter two attributes are numerical ones. For automatic construction of hierarchies for the attribute "*Grant_Amount*", the completeness requirement implies that the hierarchy constructed should cover all the amounts in the relevant data set, which could be in the range of {\$2,000 – \$97,000}, i.e., *2k – 97k*. The uniformity requirement implies

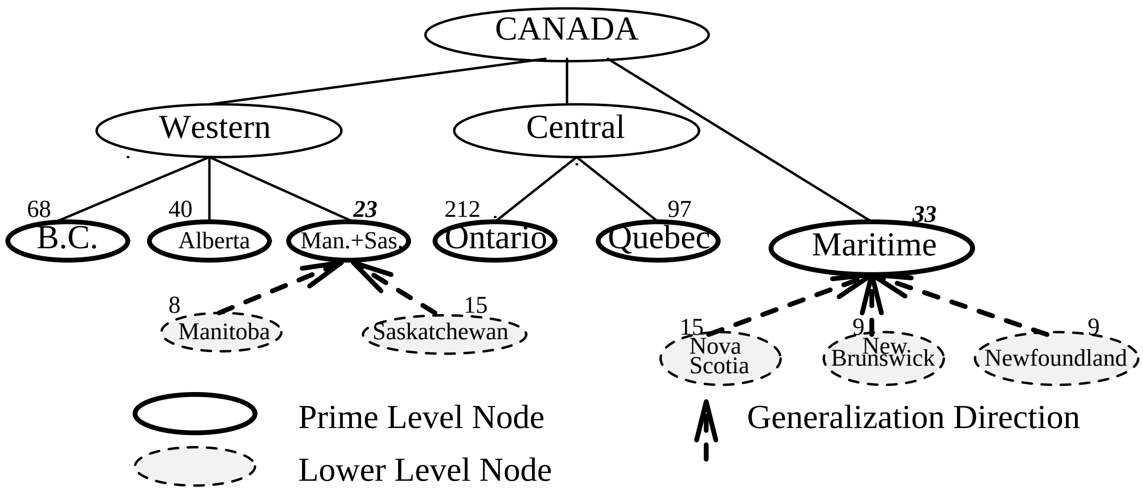

Figure 2: Dynamically adjusted concept hierarchies for *provinces*.

that the ranges of the amounts of the grants in the prime relation should be relatively evenly distributed across the whole range. If the threshold value is 4, and more people receive grants in the amount of low and medium ranges, the desired distribution could be $\{[2-12k), [12-16k), [16-23k), [23-90k)\}$. Such a set of ranges has been generated automatically. □

Based on the similar observations analyzed in the last section, the algorithm for automatic generation of concept hierarchies for numerical attributes of an initial data relation (i.e., the set of data relevant to a particular learning task) is presented as follows.

Algorithm 4.1 (Automatic generation of concept hierarchy for numerical attributes) *Automatic generation of concept hierarchy for a numerical attribute based on its data distribution in the initial data relation.*

Input: An initial data relation that contains a numerical attribute A with an attribute threshold T.

Output: A concept hierarchy H_A on A for the presentation of the prime relation.

Method. The hierarchy H_A is constructed as follows.

1. *Estimation of the total value range by data sampling.* Sampling a set of values of A in the initial data relation. Let low and $high$ be respectively the smallest and the largest value of the sampled data.
2. *Derivation of interval value.* Let $interval = (high - low)/(k \times T)$, where k is a constant reflecting the fineness of the segmentation. Usually, k is set between 5 to 10. Rounding or truncating is performed on $interval$ to make it customized to human. For example, an $interval$ of 474 is rounded up to 500. The range $low/high$ is truncated/rounded accordingly.
3. *Creation of segments.* A set of segments are created based on the range and interval. $[low, low + interval]$, $[low + interval, low + 2 \times interval]$, ..., $[low + (k \times T - 1) \times interval, high]$.
4. *Merge of segments based on data distribution.* Segments are merged into nodes based on their occurrence frequency distribution.

   First, a histogram (occurrence frequency) is computed based on the data set of the attribute in the initial relation. Each segment is attached a $count$ which is initialized to 0. The computation is performed as follows.

```
For each tuple t in the initial data relation
```

```
if there is a segment s = [l, h] such that l ≤ t[A] < h
then count[s] := count[s] + 1;
else { create a segment new: [low + k × interval, low + (k + 1) × interval]
             where k = (t[A] − low)/interval;
     count[new] := 1;}
```

Then, segments are merged into nodes so that these nodes will have relatively even distribution of occurrence frequencies. This is implemented as follows. Arrange segments in ascending order based on their range values. Merge the sequence (of segments) whose sum of the counts reaches the closest to $total\_count/T$ into one node, with its *low* range set to the *low* of the first segment, and *high* set to the *high* of the last segment. Repeat the process for the remaining segments until there is no segment left.

```
sum := 0;
first := 1;
node_count := 0;
for i := 1 to n do {
    sum_sav := sum;
    sum := sum + count[s[i]];
    if (sum ≥ total/T) or (i = n) then {
        if node_count = T − 1  % This is the last node.
        then i := n
        else if sum − total/T > total/T − sum_sav
        then i := i − 1;
        merge segments from first to i into a new node;
        sum := 0;
        node_count := node_count + 1;
        first := i + 1; } }
```

The above piece of code shows the segment merging process. □

Theorem 4.1 *The worst-case time complexity of Algorithm 4.1 is $O(n)$, where n is the number of tuples in the initial data relation.*

Rationale. Step 1 (data sampling) costs less than n since it only takes a proper subset of the initial relation and linear time to find *high* and *low*. Steps 2 & 3 work on the creation of intervals and segments using *low*, *high* and T, which is much smaller than n. In Step 4, the computation of the histogram takes $O(n)$ time since it scans the initial relation once in the computation; where the merge of segment takes the time proportional to the number of segments, which is smaller than n. Obviously, adding all the steps together, the worst-case time complexity of the algorithm is $O(n)$. □

Notice when the size of the relevant data set is huge, it could still be costly to calculate the histogram, and the histogram of the reasonably-sized *sampled* data may be used instead. Also, if the distribution is known beforehand, nodes can be built based on the known distribution.

Example 4.2 Suppose the learning task is to find the characteristic rules for the research grants in computing science from the NSERC database without a provided concept hierarchy for attribute "*Grant_Amount*". First, data sampling results in "*high = 62,350*", "*low = 5,468*" and "*interval = 1,000*". Segments are then created, and a histogram is calculated for the current task following the Algorithm 4.1. Then, the hierarchy is built using the histogram, following the segment merge method presented in Algorithm 4.1. The result is shown in Fig. 3. □

There are some other techniques on automatic generation of concept hierarchies for numerical attributes. For example, Chiu et. al. proposed an algorithm for discretization of data using hierarchical maximum entropy [3]. By this method, the initial node is the whole data set. Based on the statistical assumptions, the expected frequency of the node is computed and compared with the real frequency. If the difference is

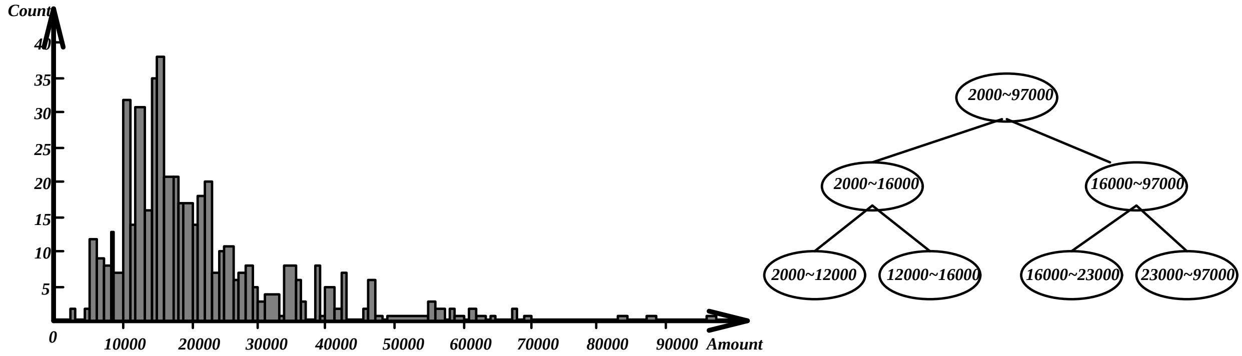

Figure 3: Histogram and Concept hierarchy generated for the *Amount*

bigger than a threshold, the node is split into several subsets based on hierarchy maximum entropy, and the process is called recursively. Our method provides a simpler and more efficient way of computation in large data sets and still achieves elegant results.

5 Discussion

5.1 Automatic generation of concept hierarchies for nominal values

Algorithms have been developed and implemented in the DBLearn system for dynamic adjustment of concept hierarchies for different kinds of attributes and automatic generation of concept hierarchies for numerical values. Moreover, a convenient graphics user interface is being developed for users to input concept hierarchies at both the schema level (e.g., $address(City \subset Province \subset Country)$) and the individual concept level (e.g., $freshman \subset undergraduate$). Nevertheless, automatic generation of concept hierarchies for nominal attributes still remains to be an attractive goal because of the substantial efforts for construction and maintenance of concept hierarchies in large databases.

There have been many interesting studies on automatic generation of concept hierarchies for nominal data, which can be categorized into different approaches: machine learning approaches [15, 4, 19], statistical approaches [1], visual feedback approaches [12], algebraic (lattice) approaches [16], etc.

Machine learning approach for concept hierarchy generation is a problem closely related to concept formation. Many influential studies have been performed on it, including Cluster/2 by Michalski and Stepp [15], COBWEB by Fisher [4], ID3 by Quinlan [19], hierarchical and parallel clustering by Hong and Mao [10], and many others.

These approaches are under our careful examination and experimentation and our goal is to develop an *efficient* algorithm to *maximize the automatic data clustering capability* for *large databases*. Our progress will be reported in detail when the algorithm development and experiments reach a mature stage.

5.2 Algorithm testing and experiments in large databases

A prototyped knowledge discovery system, DBLearn, has been constructed based upon the attribute-oriented induction technique [9]. The system takes learning requests as inputs, applies the knowledge discovery algorithm(s) on the data stored in a database, with the assistance of the concept hierarchy information stored in a concept hierarchy base. The learning requests are specified in the syntax similar to SQL, a standard relational database language. The outputs of the system are generalized relations or knowledge rules extracted from the database. The system is implemented in C with the assistance of UNIX software packages LEX and YACC (for compiling the DBLearn language interface) and operates in conjunction with the SyBase DBMS software. A database learning language for DBLearn is specified in an extended BNF grammar.

Experimentation using DBLearn has been conducted on several large real databases, including *the NSERC Grants Information system*, which contains the information about the research grants awarded by NSERC (*the Natural Sciences and Engineering Research Council of Canada*) in the year of 1990-1991. The database consists of 6 large data relations. The central relation table, *award*, contains 10,087 tuples with 11 attributes.

The background knowledge in DBLearn is represented by a set of concept hierarchies. The provided concept hierarchies can be adjusted dynamically for a particular learning task, and concept hierarchies for numerical attributes can be generated automatically based on data distribution statistics. The concept hierarchy generation and refinement algorithms described in this paper have been successfully implemented and tested against large databases.

To examine the effects of these algorithms, our experiments compare different test runs, including those applying the algorithms vs. those without. In most test runs, automatic generation of concept hierarchies for *numerical attributes* produces more desirable value ranges for different learning tasks than the user-provided ranges since the algorithm adapts data distribution statistics *dynamically* and in a better calculated way than human experts. Also, in most cases, concept hierarchies with dynamic refinement generate more desirable results than those without since the former promotes "important" (i.e., *heavily weighted*) nodes and suppresses "trivial" (i.e., *lightly weighted*) nodes in an organized way. These have been shown in the examples presented in the previous sections, which are taken from the experiments on the *NSERC Grants Information system*. Many tests on other databases were also performed using the same algorithms.

6 Conclusions

Progress has been made in this paper on the further development of the attribute-oriented induction method for efficient knowledge discovery in large relational databases, with an emphasis on the development of new algorithms for dynamic generation and refinement of concept hierarchies.

Three algorithms are presented in this paper: (1) the refinement of the basic attribute-oriented induction algorithm for learning characteristic rules; (2) dynamic refinement of concept hierarchy based on a learning task and the data statistics; and (3) automatic generation of concept hierarchies for numerical attributes based on the relevant set of data and the data distribution. These algorithms have been implemented in the DBLearn data mining system and tested against several large relational databases. The experimental results show that the algorithms are efficient and effective for knowledge discovery in large databases.

A challenging task is the automatic generation of concept hierarchies for nominal (discrete) data in large databases. We are currently examining several algorithms which work well for a relatively small amount of data and some proposals for similar tasks on large databases. The progress of our study on the development of efficient algorithms in this class for large relational databases will be reported in the future.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.

[2] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, 1991.

[3] D. K. Y. Chiu, A. K. C. Wong, and B. Cheung. Information discovery through hierarchical maximum entropy discretization and synthesis. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 125–141. AAAI/MIT Press, 1991.

[4] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[5] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.

[6] B. R. Gaines and J. H. Boose. *Knowledge Acquisition for Knowledge-Based Systems*. London: Academic, 1988.

[7] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 547–559, Vancouver, Canada, August 1992.

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

[9] J. Han, Y. Fu, Y. Huang, Y. Cai, and N. Cercone. DBLearn: A system prototype for knowledge discovery in relational databases (system demonstration). In *Proc. 1994 ACM-SIGMOD Conf. Management of Data*, Minneapolis, MN, May 1994.

[10] J. Hong and C. Mao. Incremental discovery of rules and structure by hierarchical and parallel clustering. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 177–193. AAAI/MIT Press, 1991.

[11] K. A. Kaufman, R. S. Michalski, and L. Kerschberg. Mining for knowledge in databases: Goals and general description of the INLEN system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 449–462. AAAI/MIT Press, 1991.

[12] D. Keim, H. Kriegel, and T. Seidl. Supporting data mining of large databases by visual feedback queries. In *Proc. 10th of Int. Conf. on Data Engineering*, Houston, TX, Feb. 1994.

[13] D. J. Lubinsky. Discovery from database: A review of AI and statistical techniques. In *Proc. IJCAI-89 Workshop on Knowledge Discovery in Databases*, pages 204–218, Detroit, MI, August 1989.

[14] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2.* Morgan Kaufmann, 1986.

[15] R. S. Michalski and R. Stepp. Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 5:396–410, 1983.

[16] R. Missaoui and R. Godin. An incremental concept formation approach for learning from databases. In V.S. Alagar, L.V.S. Lakshmanan, and F. Sadri, editors, *Formal Methods in Databases and Software Engineering*, pages 39–53. Springer-Verlag, 1993.

[17] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

[18] G. Piatetsky-Shapiro and C.J. Matheus. Knowledge discovery workbench for exploring business databases. *Int. J. Intell. Syst.*, 7:675–686, 1992.

[19] J. R. Quinlan. Learning efficient classification procedures and their application to chess end-games. In Michalski et. al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 463–482. Morgan Kaufmann, 1983.

[20] M. Stonebraker, R. Agrawal, U. Dayal, E. Neuhold, and A. Reuter. DBMS research at a crossroads: The vienna update. In *Proc. 19th Int. Conf. Very Large Data Bases*, pages 688–692, Dublin, Ireland, Aug. 1993.

[21] R. Uthurusamy, U. M. Fayyad, and S. Spnggler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 141–158. AAAI/MIT Press, 1991.

[22] W. Ziarko. The discovery, analysis, and representation of data dependancies in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 239–258. AAAI/MIT Press, 1991.

[23] J. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 31–54. AAAI/MIT Press, 1991.