Learning Semantically Robust Rules from Data

Yiheng Li   Latanya Sweeney

February 2004
CMU-CALD-04-100
CMU-ISRI-04-107

Center for Automated Learning and Discovery and
Institute for Software Research International
Data Privacy Laboratory

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

We introduce the problem of mining robust rules, which are expressive multi-dimensional generalized association rules. Consider a large relational table, where associated with each attribute is a hierarchy whose base values are those originally represented in the data, and values appearing at higher levels in the hierarchy represent increasingly more general concepts of base values. Attribute hierarchies provide meaningful levels of concept aggregation, such as the encoding of postal codes (ZIP) or dates, or the taxonomy of products. We find the least general rules formed by combining mixed levels of generalizations across attributes to convey the maximum expression of information supported by attribute hierarchies, parameter settings and data tuples. We term these "robust rules" and introduce a GenTree algorithm as a means to learn robust rules from a table. An example of a robust rule from a table having base values {*5-digit ZIP*, *gender*, *registration date* (*year*/*month*/*day*), *party*} might be "women living in Cambridge (021**) and registered in the 1970's (197*/xx/xx) tend to be Democrats." Previous studies on mining generalized association rules have been limited dimensionally (e.g., transactional data), by data type (e.g., quantitative data), and/or to rules expressed from either fixed-level or non-semantic abstractions. Such approaches limit the kinds of rules that can be learned. Experiments using GenTree with two real-world datasets, containing 10,000 six-attributed tuples and over 4,000 eight-attributed tuples each, show that learned rules convey more comprehensive information than possible with traditional association rule mining algorithms, because traditional approaches limit the expressivity of the rules they generate.

1. Introduction

The problem of discovering meaningful associations between data elements in a large sample of examples is important to marketing, management, and policy decision-making. Learned associations, expressed as rules, are the basis of association rule learning, first introduced in 1993 [1]. Given sets of items in an information repository, an association rule is an expression of the form Body $\Rightarrow$ Head, where Body and Head are sets of items. An association rule can be stated as "Body tends to Head." An example of an association rule is: "People living in ZIP 02139, tend to be Democrats." An association rule is corroborated by the data from which it is observed by two measures – support and confidence. *Support* is the probability that both Body and Head are satisfied: P(Body & Head). *Confidence* is the conditional probability that given Body is satisfied, Head is also satisfied: P(Head | Body). In a list of registered voters, the association rule "People living in ZIP 02139, tend to be Democrats" has 19% support and 57.9% confidence.

Finding the most "meaningful" rules for decision makers requires not only quantifiable corroboration of the rule using support and confidence, but also the ability to express the rule in semantic terms that are most useful to the decision maker. So, association rule learning can be formulated as a problem of searching through a predefined space of potential rules for the rules that best fits the examples quantifiably and semantically.

By selecting a rule representation, the human designer implicitly defines the space of all rules that an association rule learner (or "learner") can ever represent and therefore can ever learn. We use the symbol R to denote the set of all possible rules that a learner may consider regarding a relational table $\mathcal{D}$. R is determined by the human designer's choice of representations to express the semantics associated with data values in $\mathcal{D}$. The simplest choice is to rely on values in $\mathcal{D}$ alone. In this case, learned rules can only be expressed in the base values of $\mathcal{D}$. Even though some values in $\mathcal{D}$ may be semantically related to other values (e.g., both milk and juice are beverages) there is no representation provided by which the learner can consider joining related values (milk and juice) to attain more generalized rules (about beverages) that may be more corroborated or more generally expressed.

In 1995, the notion of associating a concept hierarchy with base values in transactional data was introduced [2, 8]. Rules could then be learned not only over the base values but also over generalized concepts that semantically related base values. For example, given transactions of grocery purchases and an "is-a" hierarchy that related milk and juice as beverages, rules involving milk, juice and beverages could then be learned. Such rules are called *generalized association rules* (or "cross-level" rules).
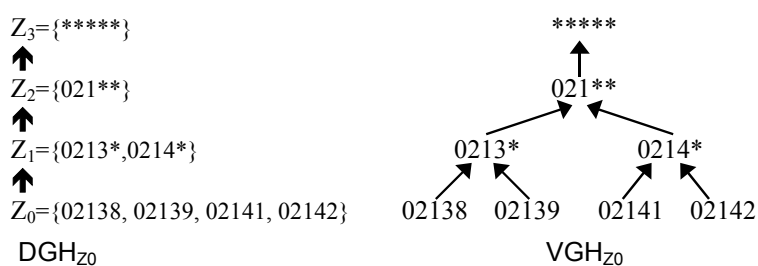

Figure 1. Attribute hierarchy for ZIP codes in Cambridge, Massachusetts, shown as domain generalization hierarchy on left and as value generalization hierarchy on right.

In the work presented herein, we allow the human designer to express a much larger and semantically extended rule space by associating membership hierarchies (e.g., "is-a") with each attribute in a large relational table $\mathcal{D}$. The base values appearing in an attribute of $\mathcal{D}$ appear as base values in the attribute's hierarchy. Values appearing at higher levels in the hierarchy represent more general concepts that remain semantically consistent with the base values. Therefore, attribute hierarchies impose a "more-general-than" relation on their values. Figure 1 has an example of an attribute hierarchy for postal codes (ZIP) in Cambridge, Massachusetts. Attribute hierarchies provide meaningful levels of concept aggregation,

which could be the encoding of ZIP codes or dates, or the taxonomy ("is-a") of products. Even quantitative data such as age or income, can have meaningful attribute hierarchies achieved by aggregating base values into increasing, non-overlapping ranges. For example, dates of birth can be generalized to month and year of birth, year of birth, 5-year age ranges, and so forth.

Attribute hierarchies greatly expand the size of the space of potential rules (R) a learner must consider. Rules can now express mixed levels of concept aggregation realized by combinations of values across different hierarchy tiers. We term these *multi-dimensional generalized association rules*. Figure 2 provides examples.

A. "People living in ZIP 02139, tend to be Democrats."
   [Support: 19.0%, Confidence: 57.9%]

B. "Women living in Cambridge (021**) and registered in the 1970's (197*/**/**), tend to be Democrats."
   [Support: 2.6%, Confidence: 83.0%]

C. "Republican Whites living in ZIP 15213 and owning home,
    tend to be females with no children."
   [Support: 2.1%, Confidence: 55.4%]

Figure 2. Multi-dimensional generalized association rules. Rule (A) is traditionally expressed in original base values. Rules (B) and (C) have terms from higher levels in their hierarchies. The source table for (A) and (B) has attributes {*5-digit ZIP*, *gender*, *registration date* (*year*/*month*/*day*), *party*}. The table for (C) additionally includes {*race/ethnicity*, *own home*, *number of children*}.

Increasing the size of R yields additional rules to consider, but the learner's task is to find "meaningful" rules, which are based on quantifiable corroboration and semantic expressiveness. Support and confidence quantify corroboration, and attribute hierarchies structure the semantics of rule expressions.

Predefined attribute hierarchies embed a priori groupings of concepts considered useful to human decision-makers, who construct them and who must interpret learned rules based on them. A learner can exploit the semantics of the "more-general-than" relation imposed by attribute hierarchies to assess the expressive power of potential rules to humans.

Each of the multi-dimensional generalized association rules in Figure 3 has the same quantified corroboration in data (e.g., the same support and confidence for the same tuples). Yet, to human decision-makers these statements do not equally convey the same expressive power.

A. "People living in ZIP 1521*,
    tend to be registered in 1965 (1965/**/**)."

B. "People living in ZIP 152**,
    tend to be registered in August 1965 (1965/08/**)."

C. "People living in ZIP 152**,
    tend to be living in ZIP 1521* and registered in August 1965 (1965/08/**)."

Figure 3. Multi-dimensional generalized association rules learned from a table having attributes {*ZIP*, *registration date*} and having equal support and confidence for the same corroborating tuples. Of these, rule (C) is a robust rule, having the most general expression for the Body and the most specific expression for the Head.

The interpretation of association rules by humans is like an *if-then* implication. In the mathematical implication $p \Rightarrow q$, p is the hypothesis, q is the conclusion, and $p \Rightarrow q$ is translated in English as "if p then q" [5]. An association rule (written Body $\Rightarrow$ Head), having a hypothesis (Body) with a large number of potential elements and a conclusion (Head) narrowly specific, is considered more useful, because the broadly stated hypothesis expands the scope of items subject to the hypothesis, and the specific conclusion provides exacting information about the subjects. So, rules having the most generalized

expressions in the Body and the most specific expressions in the Head are desired. We term these *robust rules*. Of the rules in Figure 3, only rule C is a robust rule.

A robust rule has quantifiable measures of support and confidence as well as semantic constraints. A Body is more generally expressive in one rule than another if its terms appear at the same or higher levels in attribute hierarchies and/or if it has fewer terms. Similarly, a Head is more specifically expressive in one rule than another if its terms appear at the same or lower levels in attribute hierarchies and/or it has more terms. Rules having the most generally expressive Body and the most specifically expressive Head for the same corroborating tuples are the robust rules for those tuples. A set of tuples will have at least one robust rule, which is a multi-dimensional generalized association rule. It may have more than one robust rule.

The work described herein uses the structure imposed by attribute hierarchies and observed examples in $\mathcal{D}$ to organize the search of R for robust rules. Before we describe how this search is efficiently achieved in technical terms (in Section 3), we discuss prior work (in Section 2). After the GenTree structure and algorithm are presented (in Section 3), experimental results on real-world datasets are provided (in Section 4) that show the nature and number of robust rules learned. This paper ends with a comparison of GenTree to foreseeable extensions to prior work that could attempt to learn robust rules.

2. Background

2.1 Partial Ordering and Learning

The idea of using general-to-specific ordering in concept learning has been studied. Given training examples, a concept learner recognizes similar instances not appearing in the training examples. Plotkin (1971) provided an early formalization of the more-general-than relation [4]. Simon and Lea (1973) gave an early account of learning as search through a hypothesis space [7]. Version spaces and the Candidate-Elimination algorithm were introduced by Mitchell (1982) in which a more-general-than partial ordering is used to maintain a set of consistent hypotheses and to incrementally refine this representation as each new positive training example is encountered [3]. Sebag (1996) presents a disjunctive version space approach to learn from noisy data [5]. A separate version space is learned for each positive training example, then new instances are classified by combining the votes of different version spaces. In the work presented herein, we extend these notions to rule learning to define and structure a larger and richer rule space for learning robust rules from a relational table.

2.2 Attribute Hierarchies and Semantics

Recently, Sweeney (2002) introduced attribute hierarchies as a basis for generalizing data to achieve semantically useful, yet provably anonymous data [10]. In the work presented herein, we similarly exploit the semantics of predefined attribute hierarchies not only to improve human interpretation of results, but also to enrich rule representations and to determine the semantic utility of learned rules.

2.3 Association Rule Learning

Agrawal et al. (1993) introduced the notion of learning single concept associations, expressed as rules, from base values in transactional data [1]. Srikant and Agrawal (1995) added a pre-defined "is-a" concept hierarchy over all the items in a transactional database to learn generalized association rules [8]. Learned rules involved terms at cross-levels of the hierarchy. Somewhat simultaneously, Han and Fu (1995) also introduced the use of a pre-defined "is-a" hierarchy to learn cross-level rules from transactional data, but the characterization was not one of learning more general rules by involving terms appearing increasingly higher in the hierarchy as was done in [8], but of "drilling down" to learn more specific rules by involving terms appearing increasingly lower in the hierarchy [2]. Srikant and Agrawal (1996) mined association rules from quantitative data in relational tables, thereby providing an early account of multi-dimensional

rules [9]. Quantitative values appearing in the table were partitioned into intervals whose ranges were determined from the distribution of values appearing in the table. Learned rules were expressed in terms of base values and intervals. However, different tables having the same attributes may have rules expressed in different intervals because intervals are not pre-defined. This can confound rule comparison across tables.

In the work presented herein, we associate pre-defined "more-greater-than" hierarchies with each attribute, categorical or quantitative, in a large relational table. We then provide an efficient means to learn multi-dimensional rules from the table using the hierarchies. Recognizing that many more rules can now be expressed, we provide a quantitative and semantic basis for determining the most useful (or "robust") rules.

3. Methods

Prior to presenting methods to learn robust rules efficiently, we precisely define our terms and introduce the notion of a generalization tree.

3.1 Technical Details of Attribute Hierarchies

Given an attribute A of a table $\mathcal{D}$, we define a **domain generalization hierarchy** $\mathsf{DGH_A}$ for A as a set of functions f_h : $h=0,\ldots,k\text{-}1$ such that:

$$A_o \xrightarrow{f_0} A_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{k-1}} A_k$$

$A=A_0$ and $|A_k| = 1$. $\mathsf{DGH_A}$ is over:
$$\bigcup_{h=0}^{k} A_h$$

Clearly, the f_h's impose a linear ordering on the A_h's where the minimal element is the base domain A_0 and the maximal element is A_k. The singleton requirement on A_k ensures that all values associated with an attribute can eventually be generalized to a single value. Because generalized values are used in place of more specific ones, it is important that all domains in the hierarchy be semantically compatible.

Given a domain generalization hierarchy $\mathsf{DGH_A}$ for an attribute A, if $v_i \in A_i$ and $v_j \in A_j$ then we say $v_i \leq v_j$ if and only if $i \leq j$ and:

$$f_{j-1}(\ldots f_i(v_i)\ldots) = v_j$$

This defines a *partial ordering* $\leq$ on:
$$\bigcup_{h=0}^{k} A_h$$

Such a relationship implies the existence of a **value generalization hierarchy** $\mathsf{VGH_A}$ for attribute A. Figure 1 illustrates a domain and value generalization hierarchy for domain $\mathsf{Z_0}$, representing ZIP codes for Cambridge, Massachusetts.

We recognize situations in which it may be advantageous to consider several different DGH's for the same attribute. However, as long as such DGH's have the same singleton as their maximal elements, we may represent all of them in one corresponding VGH. In this case, the VGH is a general Directed Acyclic Graph (DAG), instead of a simple tree structure.

The search for meaningful rules realized from a table can be efficiently organized by taking advantage of a naturally occurring structure over the rule space – a general-to-specific ordering of rules imposed by the domain generalization hierarchies.

For simplicity, let us assume only one DGH is associated with each attribute. Given table $\mathcal{D}(A_1, \ldots, A_m)$ and DGH_{A_i}'s, $1 \leq i \leq m$, the set of all possible generalizations of values in $\mathcal{D}$ comprise a **generalization hierarchy**, $\mathsf{GH}_\mathcal{D} = \mathsf{DGH}_{A_1} \times \ldots \times \mathsf{DGH}_{A_m}$, assuming the Cartesian product is ordered by imposing coordinate-wise order. $\mathsf{GH}_\mathcal{D}$ defines a lattice whose minimal element has values in the same domains as $\mathcal{D}$. Each node in the lattice represents a unique combination of generalization concepts for

expressing rules in $\mathcal{D}$. We know that the number of nodes in the lattice is the product of sizes of the attribute hierarchies: $\prod_{i=1}^{m}|DGH_i|$.

We define a **pair** to be any two such nodes that can be connected through a set of function links (l_1, ..., l_m), where

$$l_i = \prod_{h=h1}^{h2} f_{Aih} \quad \text{or } \equiv \text{(equal to self)}.$$

The total number of pairs in the lattice is the number of possible multi-dimensional cross-level rules that can be represented. The number of possible rules is greater, being instantiated by values appearing in the table.

Because we view learning as a search, it is natural to consider exhaustive search through the entire rule space. But even for a table having a few short attributes, these equations show exhaustive search prohibitive. So, we are interested in efficient algorithms that search very larges rule spaces for robust rules.

3.2 Notations in DAG

Because the concept of Directed Acyclic Graph (DAG) is heavily involved in our methodology, we clarify the basic notations we use. We use lower case letters, such as x, to denote *nodes* (or vertices) in a DAG. We say "x is connected to y" or "there is a connection between x and y", if there is an edge between x and y, regardless of direction. We call p a *parent* of c and c a *child* of p, if there is an edge starting at p and ending at c. We call "p' an ancestor of c" and "c a descendant of p'" if there is an edge from p' to c in the transitive-closure of the corresponding DAG.

3.3 Rule Definition

A *generalized association rule* in transaction databases is defined in [2]. We basically follow that definition and define the *multi-dimensional generalized association rule* in relational databases.

Let $\mathcal{A} = \{A_1, ..., A_m\}$ be a set of attributes in a relational dataset $\mathcal{D}$, where each A_i has an associated hierarchy, the VGH of which is a DAG. A ***multi-dimensional generalized association rule*** is an implication of the form $X \Rightarrow Y$, where $X = \{v_1^x, v_2^x, ..., v_m^x\}$, $Y = \{v_1^y, v_2^y, ..., v_m^y\}$, and v_i^x, v_i^y are a values in A_i's hierarchy. It is required that no v_i^y be an ancestor of v_i^x, i.e. $v_i^y \le v_i^x$, for all $1 \le i \le m$. We call X the Body of a rule and Y the Head of a rule. We follow the traditional definition of confidence and support for association rules. *Confidence* equals c if $c\%$ of the data tuples in $\mathcal{D}$ that support Body also support Head; *Support* equals s if $s\%$ of data tuples in $\mathcal{D}$ support both Body and Head. Given our definition, clearly, any data tuple that supports Head must also support Body, because each attribute value in Body is more-general-than (or equal to) the corresponding value in Head. Hence, support for a multi-dimensional generalized association rule is simply the percentage of data tuples that support Head.

Traditional association rules are special cases of our definition. For example, a traditional rule of the form $\{v_1^x\} \Rightarrow \{v_2^y\}$, is equivalent to our form of $\{v_1^x, *_2, *_3, ..., *_m\} \Rightarrow \{v_1^y, v_2^y, *_3, ..., *_m\}$, where $v_1^x = v_1^y$ and each $*_i$ represents the highest level of generalization in A_i's hierarchy (root in corresponding VGH_{Ai}).

The benefit of our definition is its capability of expressing more interesting rules. Consider the following example. Let $\mathcal{A} = \{5\text{-}digit\ ZIP,\ registration\ date\ (year/month/day),\ status\}$, we may have $\{021**, 1996/**/**, *\} \Rightarrow \{021**, 1996/08/**, active\}$, with 89.8% confidence and 4.5% support. This rule learns both in "width", i.e., from knowing nothing to learning something about an attribute (e.g., "$* \Rightarrow active$" for attribute *status*); and in "depth", i.e., from knowing something to learning something more specific about an attribute ("$1996/**/** \Rightarrow 1996/08/**$" for attribute *registration date*). The latter type of learning ("in depth") cannot be achieved traditionally.

3.4 GenTree Definition

A generalization tree (**GenTree**) is a DAG, which represents the multi-dimensional generalization relations among all data tuples in a relational dataset over a set of hierarchical attributes, and satisfies the properties of *completeness* and *conciseness*.

There are two types of nodes in GenTree: leaves and non-leaves. Each **leaf** node represents a corresponding data tuple. Each **non-leaf** node represents a multi-dimensional generalization form and the set of data tuples that can be generalized to that form. **Root** is a special non-leaf node, which represents the overall generalization of all attributes (total suppression of all attributes) and the set of all data tuples.

3.4.1 Notations in GenTree

We use:

Form(x) to denote the corresponding multi-dimensional generalization form (or expression form if x is a leaf) which x represents. An example is *Form*(x) = (ab*, 1*) in Figure 4.
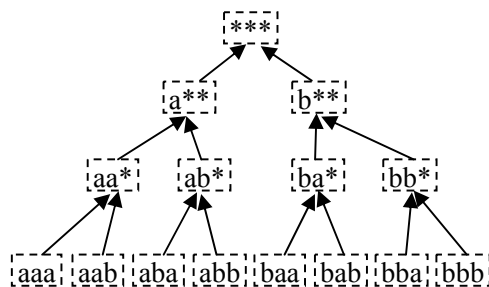
Form(x)$_i$ to denote the value of the i'th attribute in *Form*(x); and,

Tuples(x) to denote the set of tuples that can be expressed by, or can be generalized to *Form*(x).
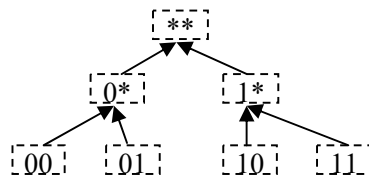
Table 𝒟

attribute 1	attribute 2
aba	11
aab	11
abb	10
abb	11
aaa	11

Hierarchy (VGH) of an attribute 1

Hierarchy (VGH) of an attribute 2
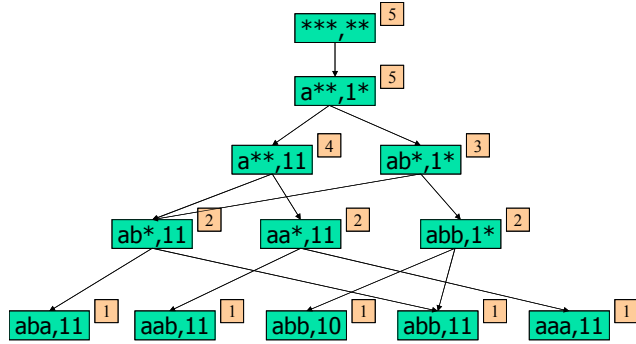

GenTree based on 𝒟

Figure 4. A GenTree using the table and VGH's shown. Tuples in the table appear as leaves. Parents are generalizations. The number associated with node x shows how many tuples are represented by it, i.e., $|Tuples(x)|$.

3.4.2 Definitions in a Generalization Tree

We define $Form(x)_i < Form(y)_i$ and $Form(y)_i > Form(x)_i$ iff. $Form(y)_i$ is "more-general-than" $Form(x)_i$ in VGH_i. That is, the path from $Form(x)_i$ to $Form(y)_i$ in VGH_i exists;

We define $Form(x) = Form(y)$ iff. $Form(x)_i = Form(y)_i$ for all $1 \leq i \leq m$;

We define $Form(x) < Form(y)$ (and $Form(y) > Form(x)$) iff. $Form(x)_i \leq Form(y)_i$ for all $1 \leq i \leq m$, and $Form(x)_j < Form(y)_j$ for at least one j, $1 \leq j \leq m$;

We define x an **ancestor** of y (and y a **descendant** of x) if $Form(x) > Form(y)$, and also define x a **parent** of y (y a **child** of x) if x and y are directly connected. Clearly, *root* is an ancestor of all other nodes and a leaf is never an ancestor.

Given node b, an ancestor a, and c, a parent of a, the rule $Form(a') \Rightarrow Form(b)$ is a **robust rule** iff: $Form(a) \leq Form(a') < Form(c)$ and $\nexists Form(x)$, s.t. $Form(a') < Form(x) < Form(c)$ over the same tuples.

3.4.3 Properties of a Generalization Tree

Completeness: Let Q be the set of all data tuple sets $Tuples(z)$'s, where $z \in$ {all nodes in a GenTree}, $\nexists Form(z')$, s.t. $Tuples(z') \notin Q$.

That is, for any data tuple set that is not found in Q, you will not be able to use a generalization form based on attributes' hierarchies to represent it accurately.

Conciseness: For $\forall$ node $z \in$ {all nodes in a GenTree} (except *root*), $\nexists$ another form $Form(z')$, s.t. $Tuples(z') = Tuples(z)$ and $Form(z') < Form(z)$.

That is, each node's form is the most concise one with regard to the tuples represented by it. *Root* is excluded because it has the most general generalization form, and serves as a foundation for tree construction.

3.4.4 Multiple Parents and Children

With regard to parent and child, since GenTree is a DAG, each node may have multiple parents as well as multiple children. However, it is easy to prove that a node x's possible maximal number of parents is the number of unsuppressed attributes in $Form(x)$; and the possible maximal number of children is

$$\sum_{v \in non\_base(Form(x))} Fanout(v)$$

where $non\_base(Form(x))$ is the set of $Form(x)$'s non-base attribute values (in corresponding hierarchies), $Fanout(v)$ is the number values that can be generalized to v directly (in corresponding hierarchy). For example, let's look at the node of the form (aa*, 11) in Figure 4. The value "aa*" of attribute 1, is an unsuppressed and non-base value with $Fanout$("aa*") = 2 in the attribute 1's hierarchy; the value "11" of attribute 2, is an unsuppressed but base value in attribute 2's hierarchy. So the node could have at most 2 parents and 2 children, while in the actual tree it has 1 parent and 2 children, which is determined by the property of conciseness.

3.5 GenTree Construction

The basic steps of GenTree construction algorithm are shown in Figure 5. The main steps of this algorithm are step 2.6 and 2.7, which are shown in more detail in Figure 6 and 7, respectively.

In step 2.6, the algorithm searches the current tree in a top-down direction for possible new non-leaf nodes that could be created due to t. It starts by setting a cursor x at *root* and comparing x with t: 1) If t will be a descendant of x, the algorithm moves x down to each of its children; 2) If $Form(x)_i \geq Form(t)_i$ for some attribute i, it determines whether each most concise common ancestor g of x and t is already created, if not, g is created and is inserted in step 2.7, then x is moved down to its children; 3) In other situations, same actions as in 2) except that x will not move further down. Step 2.6 guarantees the properties of completeness and conciseness.
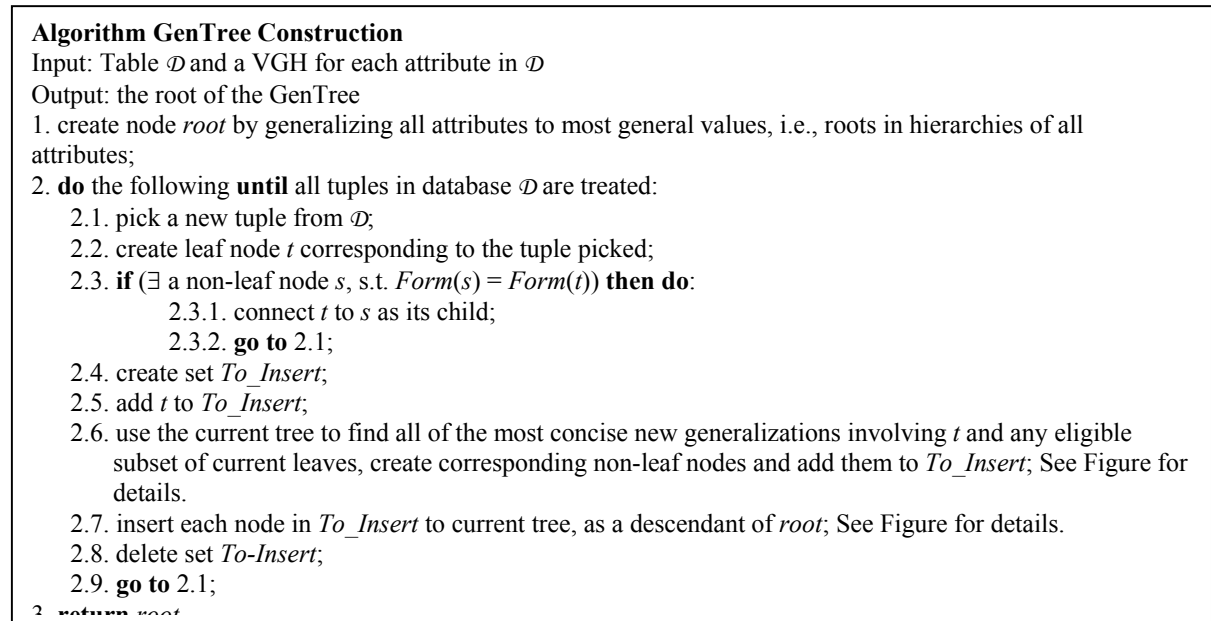

Algorithm GenTree Construction
Input: Table $\mathcal{D}$ and a VGH for each attribute in $\mathcal{D}$
Output: the root of the GenTree
1. create node *root* by generalizing all attributes to most general values, i.e., roots in hierarchies of all attributes;
2. **do** the following **until** all tuples in database $\mathcal{D}$ are treated:
    2.1. pick a new tuple from $\mathcal{D}$;
    2.2. create leaf node t corresponding to the tuple picked;
    2.3. **if** ($\exists$ a non-leaf node s, s.t. $Form(s) = Form(t)$) **then do**:
           2.3.1. connect t to s as its child;
           2.3.2. **go to** 2.1;
    2.4. create set *To_Insert*;
    2.5. add t to *To_Insert*;
    2.6. use the current tree to find all of the most concise new generalizations involving t and any eligible subset of current leaves, create corresponding non-leaf nodes and add them to *To_Insert*; See Figure for details.
    2.7. insert each node in *To_Insert* to current tree, as a descendant of *root*; See Figure for details.
    2.8. delete set *To-Insert*;
    2.9. **go to** 2.1;
3. **return** *root*

Figure 5. Basic Steps of GenTree Construction.

In Step 2.7, the algorithm's task is to find the proper parents and children for the new node x. The basic idea is starting from *root*, search downward for eligible parents of x. The tricky part is to find x's children, details are shown in Figure 7.

3.5.1 Size of GenTree

For simplicity, consider full tree-like hierarchies. The maximum number of more-general values of a base value equals "the height of the hierarchy – 1".

For a combination of hierarchical attributes, the maximum number of cross-level ancestors of a leaf representing a base-valued tuple, is related to the product of the heights of the hierarchies:

$$(\prod_i H_i) - 1$$

where H_i is the height of hierarchy associated with attribute i.

An extreme upper bound of GenTree size is:

$$n \cdot \prod_i H_i$$

where n is the number of data tuples.

This is huge! BUT leaves have ancestors in common, and not all ancestors are necessary for a practical dataset, so actual GenTree size is much, much smaller than this upper bound!

Algorithm findNewNonLeafNodes
Input: new leaf node *t*, node *root*.
Output: set NewNonLeaves, containing non-leaf nodes representing the most concise new generalizations involving *t* and any eligible subset of current leaves rooted at *root*.
1. **let** NewNonLeaves ← Φ
2. exploreNodePair(*t*, *root*, NewNonLeaves)
3. **return** NewNonLeaves

Algorithm exploreNodePair
Input: new leaf node *t*, current node *x*, node set NewNonLeaves.
Output: void.
1. **if** (*x* has been called by another exploreNodePair(*t*, *x*, NewNonLeaves) **then return**
2. compare *Form*(*t*) with *Form*(*x*)
   2.1. **case** (*Form*(*t*) < *Form*(*x*)):
      2.1.1. **for** each child *y* of *x* **do** exploreNodePair(*t*, *y*, NewNonLeaves)
   2.2. **case** (*Form*(*t*)$_i$ ≤ *Form*(*x*)$_i$ for some *i*):
      2.2.1. G ← generalize(*t*, *x*)
      2.2.2. **for** each node *g* in G, **if** (∄ *h* in current tree, s.t. *Form*(*h*) = *Form*(*g*)) **then** add *g* to NewNonLeaves
      2.2.3. **for** each child *y* of *x* **do** exploreNodePair(*t*, *y*, NewNonLeaves)
   2.3. **case** (else):
      2.3.1. G ← generalize(*t*, *x*)
      2.3.2. **for** each *g* in G, **if** (∄ *h* in current tree, s.t. *Form*(*h*) = *Form*(*g*)) **then** add *g* to NewNonLeaves

Algorithm generalize
Input: node *x*, node *y*.
Output: node Set G, containing all of such node *g*, that is one of the most concise generalizations of *x* and *y*, i.e., s.t. *Form*(*g*) ≥ *Form*(*x*), *Form*(*g*) ≥ *Form*(*y*), with a guarantee that ∄ *g'* s.t. *Form*(*g'*) ≥ *Form*(*x*), *Form*(*g'*) ≥ *Form*(*y*) and *Form*(*g'*) < *Form*(*g*).
1. **for** each *i*, 1 ≤ *i* ≤ *m*, **do**:
   1.1. in the hierarchy of the *i*'th attribute, find the closest "more-general" values shared by *Form*(*x*)$_i$ and *Form*(*y*)$_i$
   1.2. add all found values in set V$_i$
2. **let** G ← Φ
3. **for** each different generalization form, i.e., a combination of values chosen across V$_i$'s for all *i*, 1 ≤ *i* ≤ *m*, create a corresponding node *g* and add *g* to G
4. **return** G

Figure 6. Algorithms that perform Step 2.6 in GenTree construction.

3.6 Rule Mining Algorithm

Our goal is to efficiently mine robust rules from a table using a GenTree. We begin by precisely defining a robust rule. We then describe a conceptual algorithm for mining robust rules, and present an efficient one.

    Definition: Given table $\mathcal{D}$, attribute set $\mathcal{A} = \{A_1, \ldots, A_m\}$, associated hierarchies and minimum support and confidence thresholds *minsup* and *minconf*, a multi-dimensional generalized association rule $X \Rightarrow Y$, is a **robust rule** with respect to the VGH_{Ai}'s if the following conditions are satisfied:
        1. |*Tuples*(*Y*)|/|$\mathcal{D}$| ≥ *minsup* and |*Tuples*(*Y*)|/|*Tuples*(*X*)| ≥ *minconf*;
        2. ∄ *Y'*, s.t. *Tuples*(*Y'*) = *Tuples*(*Y*) and *Y'* < *Y*;
        3. ∄ *X'*, s.t. *Tuples*(*X'*) = *Tuples*(*X*) and *X'* > *X*.
    In order to learn robust rules, we start by finding Heads. The property of conciseness of GenTree assures that the generalization form of each node (except *root*) is most explicit with regard to the tuple set

it represents. The property of completeness assures that the set of node forms (except that of *root*) in GenTree may serve as a complete candidate set of Heads. Given *minsup* as a user-specified minimum support, we may prune the candidate set by removing $\forall Form(p)$, s.t. $|Tuples(p)| < |\mathcal{D}| \cdot minsup$.

For each candidate Head Y ($Form(y) = Y$), how do we find a good corresponding Body X in order to complete a rule $X \Rightarrow Y$?

Algorithm insertToTree
Input: node x, node r.
Output: void.
Assume: $Form(x) < Form(r)$, or $Form(x) = Form(r)$ only if x is a leaf node and r is a non-leaf node
1. **if** (r has been called by another insertToTree(x, r)) **then return**
2. **let** ConnectAsChild = true //show whether x needs to be connected to r as a child
3. **for** each child s of r **do**:
   3.1. compare $Form(x)$ with $Form(s)$
      3.1.1. **case** (($Form(x) < Form(s)$) or ($Form(x) = Form(s)$, x is a leaf and s is a non-leaf)):
         3.1.1.1. insertToTree(x, s)
         3.1.1.2. **let** ConnectAsChild = false
      3.1.2. **case** (x is a non-leaf, $Form(x) > Form(s)$):
         3.1.2.1. insertBetween(s, x, r)
         3.1.2.2. **let** ConnectAsChild = false
      3.1.3. **case** (x and s are a non-leaves, $Form(x)_i \leq Form(s)_i$ or $Form(x)_i \geq Form(s)_i$ for each i, $1 \leq i \leq m$):
         3.1.3.1. addPossibleSubTree(s, x)
4. **if** (ConnectAsChild = true) **then** connect x to r as its child

Algorithm insertBetween
Input: node s, node x, node r.
Output: void.
1. connect x to r as its child
2. **if** (s is not a child of x yet) **then** connect s to x as its child
3. disconnect s, a former child, from r

Algorithm addPossibleSubTree
Input: node s, node x.
Output: void.
1. **if** (s has been called by another addPossibleSubTree(s, x)) **then return**
2. **for** each child u of s **do**:
   2.1. **if** ($Form(u) < Form(x)$ and u is not a child of x) **then** connect u to x as its child
   2.2. **else if** (u is a non-leaf node, $Form(x)_i \leq Form(u)_i$ or $Form(x)_i \geq Form(u)_i$ for each i, $1 \leq i \leq m$) **then**
      addPossibleSubTree(u, x)

Figure 7. Algorithms that perform step 2.7 in GenTree construction.

From the definition, we know that data tuples that support Body form a superset of those for Head. The structure of GenTree and the property of completeness assure that all possible generalization forms for the tuples are represented by the ancestors shared by the leaves (tuples). We might falsely assume then that the forms of all such ancestors are useful Bodies. The property of conciseness guarantees each of these ancestors has a most specific generalization form representing a corresponding superset , but on the contrary, a Body of a robust rule must be of a most generalized form. So, the set of y's ancestor nodes only provide an initial set of candidates for Bodies. The node y itself is also included if rules with confidence of 100% are considered. Given *minconf* a user-specified minimum confidence, we also prune candidates by removing all nodes p s.t. $|Tuples(p)| > |Tuples(y)|/minconf$.

Of the ancestors that are candidates for Bodies, how do we find most generalized form(s) for each of them? For each node c taken from the candidates, examine each of its parent c'. Usually, $|Tuples(c')| >$

$|Tuples(c)|$ because of the superset-set relation, and the pair(c, c') bound the expression of a Body X such that $Form(c) \leq X < Form(c')$. We generate the most generalized form(s) in terms of c and c', as described below.

We compose Body $X = Form(x)$ such that $Form(c) \leq Form(x) < Form(c')$ and $\nexists Form(r)$ where $Form(x) < Form(r) < Form(c')$. This is done as follows using each c_i, c_i' and VGH_i. Let S be the set of all possible generalization forms between c and c', inclusive. Remove all generalizations from S whose tuples would not be $Tuples(c)$. Remove each generalization G_1 from S if there exists a G_2 in S such that $G_2 > G_1$. Each of the resulting forms in S is an expression for X.

This is a conceptual non-efficient algorithm for finding robust rules in a GenTree. Figure 8 provides an efficient algorithm. Starting at each child of *root*, we traverse the GenTree in a top-down direction and treat the generalization form of each node a candidate of Head Y ($Form(y) = Y$). If the number of tuples which a node represents does not satisfy *minsup*, none of its descendants will be visited. For each candidate node y, all eligible ancestor nodes (with regard to *minconf*) will be visited as a candidate of X's foundation. We create actual X's by further generalizing them to most general forms that represent the same set of tuples.

Algorithm mineRobustRules
Input: node *root*, rule parameters *minsup* and *minconf*.
Output: association rule set R, which satisfies *minsup* and *minconf*.
1. **let** R $\leftarrow$ Φ
2. **for** each child s of *root* **do** findRulesFor(s, *minsup*, *minconf*, R)
3. **return** R

Algorithm findRulesFor
Input: node s, rule parameters *minsup* and *minconf*, rule set R.
Output: void.
1. **if** (s has been called by another findRulesFor(s, *minsup*, *minconf*, R)) **then return**
2. **if** ($|Tuples(s)|$ does not satisfy *minsup*) **then return**
3. findGoodPairs(s, s, *minconf*, R) //rules with 100% confidence also considered
4. **for** each child q of s **do** findRulesFor(q, *minsup*, *minconf*, R)

Algorithm findGoodPairs
Input: node s, node s', rule parameter *minconf*, rule set R.
Output: void.
Assume: $Form(s') \geq Form(s)$
1. **if** (s and s' has been called by another findGoodPairs(s, s', *minconf*, R)) **then return**
2. **if** ($|Tuples(s)|/|Tuples(s')|$ does not satisfy *minconf*) **then return**
3. **let** NewRules $\leftarrow$ Φ
4. **for** each parent p of s' **do**:
   3.1. **for** each pair of attribute values ($Form(p)_i$, $Form(s')_i$) **do**:
      3.1.1. **if** ($Form(p)_i > Form(s')_i$) **then for** each value $\underline{v}$ which can be generalized directly to $Form(p)_i$ and $\underline{v}$
          $\geq Form(s')_i$ in the hierarchy of the i'th attribute **do**:
         3.1.1.1. **let** FORM $= Form(p)$
         3.1.1.2. replace $FORM_i$ with $\underline{v}$
         3.1.1.3. **let** new_rule be "$FORM \Rightarrow Form(s)$" with support $= |Tuples(s)|/|\mathcal{D}|$ and confidence $=$
             $|Tuples(s)|/|Tuples(s')|$
         3.1.1.4. **if** (new_rule $\notin$ NewRules) **then** add new_rule to NewRules
4. add all rules contained in NewRules to R
5. **for** each parent p of s' **do** findGoodPairs(s, p, *minconf*, R)

Figure 8. Algorithms for mining robust rules from a GenTree.

For some GenTree, there is a special situation: *root* has only one child q. In this case, $Tuples(q) = Tuples(root)$, representing all data tuples. A rule "$Form(root) \Rightarrow Form(q)$ with 100% support and 100% confidence" must be true. This means the most specific generalization form for all tuples in the corresponding dataset is $Form(q)$, i.e., the value of the i'th attribute for each tuple can be generalized to $Form(q)_i$, for all i. So, if any rule's Body includes $Form(q)_i$, in order to make it most general, a special

operation is needed: replace $Form(q)_i$ with $Form(root)_i$, i.e., the i'th attribute should be suppressed in Body. See Figure 9 for a rule mining example.
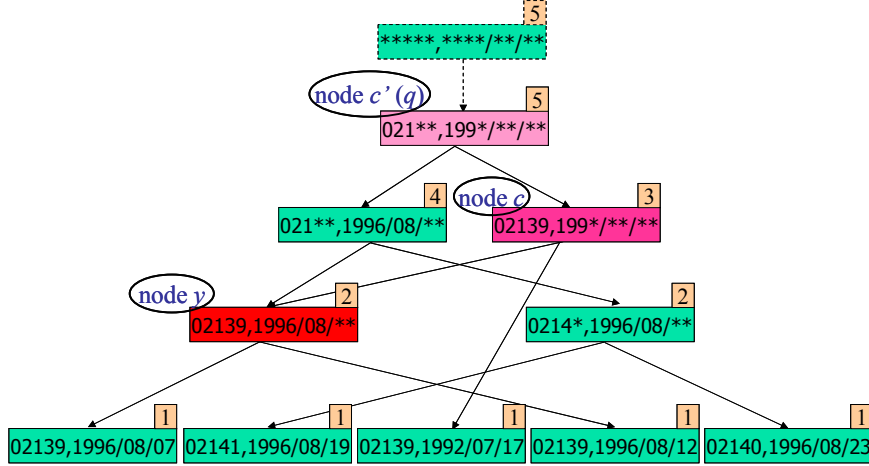

Figure 9. Rule mining example: Head $Y = Form(y)$, a possible Body X is bounded by c and c'. Under usual GenTree mining conditions, Body X = (0213*, 199*//**) based on c and c'. However, in the above GenTree, c' is the only child q of *root*, which is a special GenTree mining case in which we replace "199*/**/**" with "****/**/**" (supression). So finally, Body X = (0213*, ****/**/**), and the corresponding robust rule is (0213*, ****/**/**) $\Rightarrow$ (02139, 1996/08/**).**

Theorem1: Given table $\mathcal{D}$, attribute set $\mathcal{A}$ = {A_1, ..., A_m}, associated hierarchies and minimum support and confidence thresholds *minsup* and *minconf*, a rule generated by the GenTree algorithm is a robust rule.

Proof sketch: Condition 1 is satisfied as stated above; the property of conciseness guarantees condition 2 and the algorithm "generalize" in Figure 6 ensures that the GenTree constructed is consistent with the property of conciseness; condition 3 is satisfied by algorithm "findGoodPairs" in Figure 8.

Theorem2: The set of rules generated by the GenTree algorithm is complete with regard to *minsup* and *minconf*.

Proof sketch: The theorem is guaranteed by the property of completeness, the algorithm "exploreNodePair" in Figure 6 ensures that the GenTree constructed is consistent with the property of completeness and the rule mining algorithm ensures that all possible Body-Head combinations are considered.

4. Experimental Results

4.1 Voter List for Pittsburgh, Pennsylvania

We conducted the first experiment on dataset $\mathcal{D}_1$, The 2001 voter list for ZIP 15213 in Pittsburgh, Pennsylvania having a total of 4,316 records [12]. The values of 8 attributes for each record, i.e. {*sex*, *birthdate* (*year/month/day*), *regis_date* (*year/month*), *party_code*, *ethnic_code*, *income* (code), *home_owner*, *havechild*}, were selected as a tuple. With rule parameters *minsup* = 2% and *minconf* = 50%, 8160 robust rules were learned after applying the GenTree algorithm. The following are rule examples:

{*, 196*/**/**, 198*/**, D, *, *, *, *} $\Rightarrow$ {F, 196*/**/**, 198*/**, D, *, *, *, *} "Democrats born in 1960's and registered in 1980's tend to be Female." [Support: 2.2%, Confidence: 52.2%]

{*, ****/**/**, 19**/**, R, W, *, D, *} $\Rightarrow$ {F, 19**/**/**, 19**/**, R, W, *, D, F} "Republican Whites owning home tend to be females with no children." [Support: 2.1%, Confidence: 55.4%]

Of the 8160 robust rules, 167 of them learned information in "depth". The ratio is not high because only 2 attributes, i.e. *birthdate* and *regis_date,* have multi-level hierarchies; the remaining 6 attributes are treated as categorical, i.e. 1-level hierarchies, in original dataset.

Figure 10 shows the growth of GenTree Size for $\mathcal{D}_1$. It is clear that the actual GenTree size is much, much smaller than the "extreme upper bound" and grows linearly with the number of tuples.

4.2 Voter List for Cambridge, Massachusetts

We conducted the second experiment on the 1997 voter list for Cambridge, Massachusetts [11]. We randomly sampled 10,000 records as dataset $\mathcal{D}_2$, from the original 54,805 records. The values of 6 attributes for each record, i.e. {*5-digit ZIP*, *sex*, *birthdate* (*year*/*month*/*day*), *regdate* (*year*/*month*/*day*), *party*, *status*}, were selected as a tuple. With rule parameters *minsup* = 2% and *minconf* = 50%, 4140 robust rules were learned after applying the GenTree algorithm. Figure 11 shows a part of rules that were learned.

Of the 4140 robust rules learned, 1117 of them learned information in "depth". Only 3 attributes in $\mathcal{D}_2$, i.e. *5-digit ZIP*, *birthdate* and *regdate*, that are associated with multi-level hierarchies. We may infer that even larger portion of rules would learn "in depth" if more attributes have multi-level hierarchies.

Figure 12 shows the growth of GenTree Size for $\mathcal{D}_2$. It also indicates that the actual GenTree size is much, much smaller than the "extreme upper bound" and grows linearly with the number of tuples.
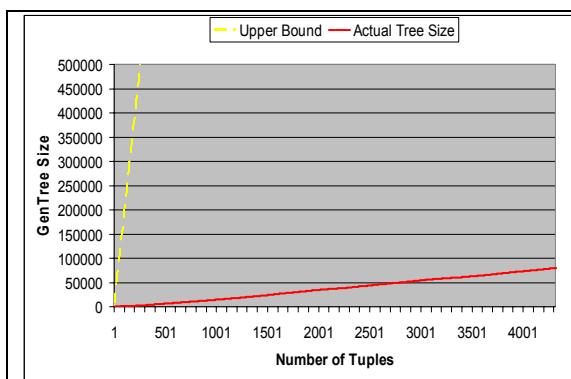

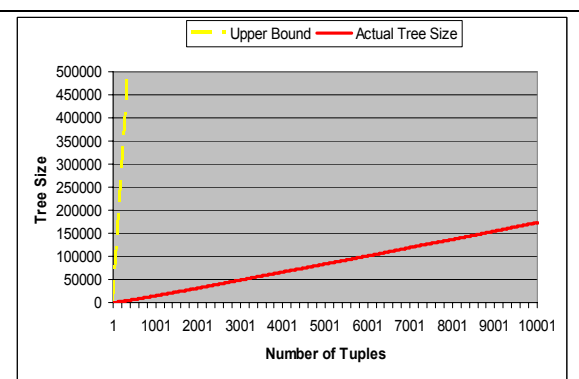| Figure 10. GenTree Size versus Number of Tuples for dataset "Voter List for ZIP 15213". | Figure 12. GenTree Size versus Number of Tuples for dataset "Cambridge Voter List, 1997". |

ZIP	Party	Sex	Birth	Reg	Status	Infers	ZIP	Party	Sex	Birth	Reg	Status	Support	Confidence
*****	*	F	197*/**/**	1996/**/**	*	==>	021**	*	F	197*/**/**	1996/**/**	A	3.27%	99.70%
0213*	*	F	197*/**/**	1996/**/**	*	==>	0213*	*	F	197*/**/**	1996/**/**	A	2.53%	99.61%
*****	R	*	19**/**/**	1996/**/**	*	==>	021**	R	*	19**/**/**	1996/**/**	A	2.01%	99.50%
02138	D	*	19**/**/**	1996/**/**	*	==>	02138	D	*	19**/**/**	1996/**/**	A	3.90%	99.49%
*****	*	F	19**/**/**	197*/**/**	A	==>	021**	D	F	19**/**/**	197*/**/**	A	2.64%	83.02%
02139	*	*	****/**/**	19**/**/**	*	==>	02139	D	*	****/**/**	19**/**/**	*	18.98%	57.90%

Figure 11. Sample from 4140 robust rules learned from Cambridge, Massachusetts voter data. The topmost rule reads "Women voters in this list born in the 1970's and registered in 1996, live in Cambridge (021) and are active." The bottom rule reads "Voters living in ZIP 02139 and registered in last century (19**/**/**) tend to be Democrats."**

4.3 Discussion of Experimental Results

The number of rules learned through $\mathcal{D}_1$ is almost twice that of $\mathcal{D}_2$, although $|\mathcal{D}_1|$ is less than half of $|\mathcal{D}_2|$. This is not surprising because the former has 8 attributes while the latter has 6. The number of rules is should grow exponentially with the number of attributes.

The ratio of rules that learned information in depth is much higher for $\mathcal{D}_2$ than for $\mathcal{D}_1$. The reason is, $\mathcal{D}_2$ has 3 attributes with multi-level hierarchies (*ZIP*, *birthdate*, *regdate*), but $\mathcal{D}_1$ has only 2 (*birthdate*, *regis_date*).

For experiments with both datasets, it is shown that the actual tree sizes are much, much smaller than the "extreme upper bound!"

4.4 Comparison to Fixed-Level Mining

Because mining for robust rules is novel, we are unable to find other currently existing approaches against which to compare. However, theoretical comparisons between our approach and an adapted traditional a priori algorithm can be done.

One way to adapt the traditional approaches that do not incorporate hierarchies to learn generalized rules is to replace base attribute values with generalized equivalents.

We experimented on $\mathcal{D}_2$ based on the following settings for attributes with multi-level hierarchies, i.e. {*ZIP*, *birthdate*, *regdate*}: Free cross-level and 8 Fixed-level settings, e.g., (5-digit, year, year), (3-digit, year, decade) and (5-digit, decade, year). Figure 13 provides the comparison of the number of rules that can be learned based on the above settings. We are not surprised to see that the number of rules coming from any Fixed-level settings is much smaller than that of free cross-level. In order to analyze the distribution of these rules, we projected them on the support-confidence plane for each setting (Figure 14). After removing free cross-level rules which are overlapped by Fixed-level ones in terms of data corroboration, i.e. having the same set of supporting data tuples for Body and Head respectively, it turned out that there are as many as 2931 free cross-level left (Figure 15). Notice the large numbers of high confidence and of low support rules learned only by our method.
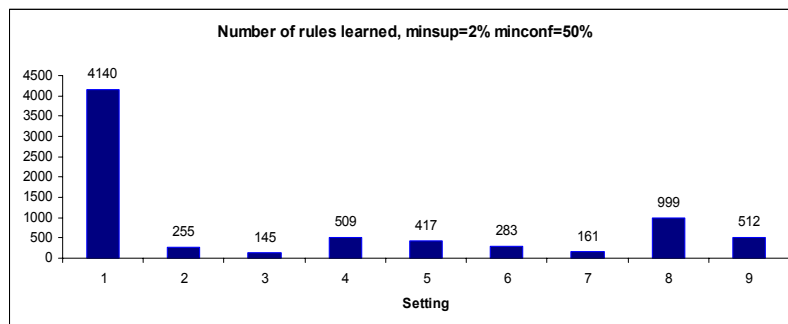

Figure 13. Number of rules from cross-level versus fixed-level mining, having minimum support of 2% and minimum confidence of 50% with attributes {*ZIP*, *birthdate*, *regdate*}. Legend: (1) free cross-level; (2) 5-digit, year, year; (3) 3-digit, year, year; (4) 5-digit, decade, year; (5) 5-digit, year, decade; (6) 3-digit, decade, year; (7) 3-digit, year, decade; (8) 5-digit, decade, decade; (9) 3-digit, decade, decade.

The total number of rules generated by those 8 Fixed-level settings is $(255 + 145 + \dots + 512) = 3281$, but in terms of data corroboration they only account for $(4140 - 2931) = 1209$ rules. This redundant overlapping is due to the limited expressivity of traditional rule forms, which we will discuss further in the next section.
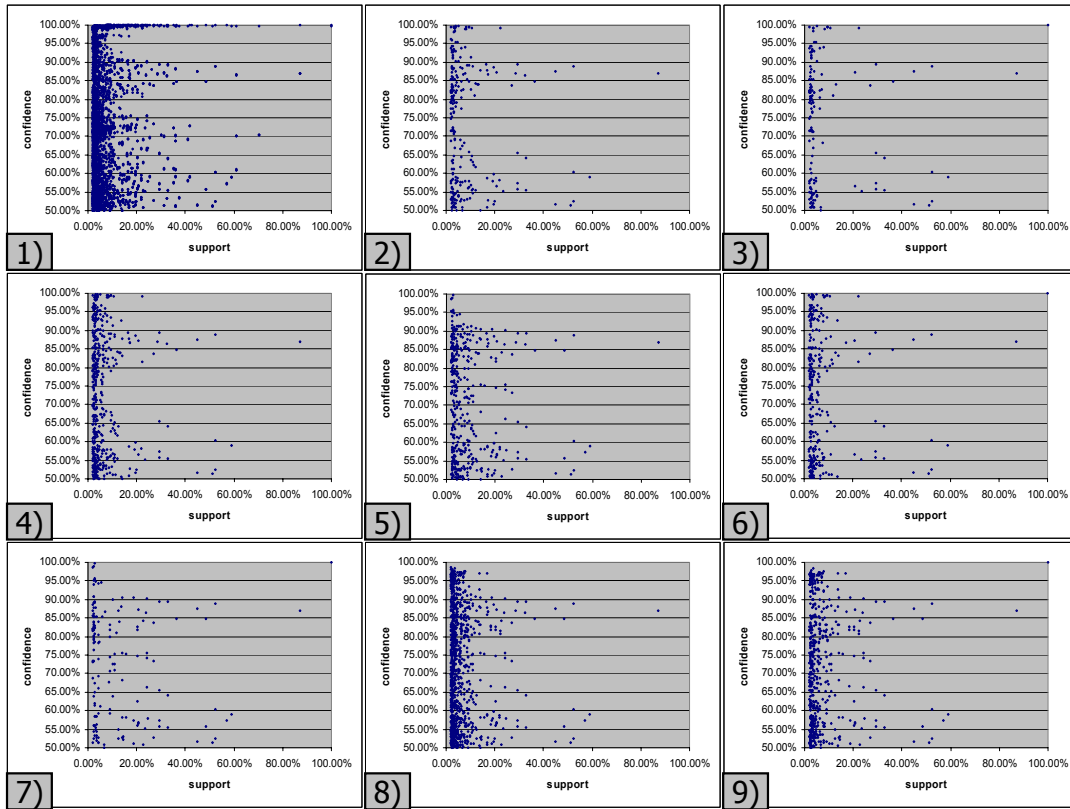
Figure 14. Rules projected on support-confidence planes for different settings as described in Figure 11.
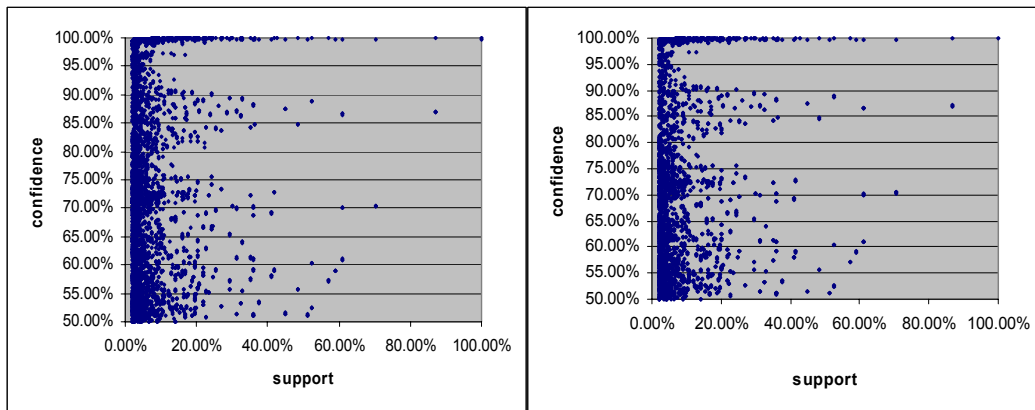

Figure 15. Before (left) and after (right) removing overlaps: 4140 robust rules (left), 2931 remaining (right).

5. Discussion

5.1 Comparison to Fixed-Level Mining

Traditional rule mining methods, if adapted, are similar to mining fixed-level rules at all possible combination of levels. In transaction database domain, there is a concept of "frequent itemset" (Figure 16). Adapting this idea to relational database domain, we have a concept of "frequent attribute-value set". We use frequent attribute-value set with fixed levels of abstractions to demonstrate how traditional rule

mining approaches might be adapted to mine for multi-dimensional generalized association rules, though such rules are not necessarily robust.

Previous algorithms are mostly based on "Apriori", if adapted:

1. Scan database, for each attribute, find all "frequent 1-attribute-value sets" which satisfy *minsup* at different levels of corresponding hierarchies

2. Generate "frequent 2-attribute-value set" candidates by combining "frequent 1-attribute-value sets"

3. Scan database again for validation: find all "frequent 2-attribute-value sets" which satisfy *minsup*

4. Continue to find eligible "frequent k-attribute-value sets" based on "frequent (k-1)-attribute-value sets"

Items Bought
A,B,C
A,C
A,D
B,E,F

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

Figure 16. Example of "frequent itemset".

This adapted method is computationally expensive if *minsup* is small, due to costly candidate generation for "frequent attribute-value sets" involving many attributes. Of course, interesting rules are usually of small support. The adapted method also requires multiple scans over the table, which is a burden.

Rules mined by the adapted method are based on "frequent attribute-value sets." For a rule Body $\Rightarrow$ Head, Body and Head are selected from the same "frequent attribute-value set". We treat a "frequent attribute-value set" as a form of (Body & Head), with subsets as Bodies and corresponding complementary subsets as Heads. An eligible pair must satisfy the *minconf* requirement. This requires extra computation to calculate these subsets and determine confidence.

Rules mined from this adapted method can only learn additional attributes in Head, i.e., learning in "width," and is unable to provide more specific attribute values in Body, i.e., learning in "depth."

In our GenTree algorithm only the most concise multi-dimensional generalizations are found, so there is no redundancy. Relations are expressed clearly by connections between nodes. We implemented an efficient rule mining algorithm based on GenTree, so there is no extra computational cost and no redundant rules. GenTree construction is computationally cheap for small *minsup* requirement, because there are no costly candidate generating and validating procedures to perform.

Multi-dimensional generalized association rules are richer in expressivity, capable of expressing robust rules through learning in "depth".

ZIP	RegDate
02139	1996/08/07
02141	1996/08/19
02139	1992/07/17
02139	1996/08/12
02140	1996/08/23

Figure 17. Dataset Example.

In the dataset shown in Figure 17, consider tuple set {(02139, 1996/08/07), (02139, 1996/08/12)} for example. Using the adapted method: "frequent 2-attribute-value sets" were found and rules learned with *minsup*=40% and *minconf*=60%. The results are as follows:

{0213*, 1996/**/**}, {0213*} $\Rightarrow$ {1996/**/**} [40%, 66.7%]

{0213*, 1996/08/**}, {0213*} $\Rightarrow$ {1996/08/**} [40%, 66.7%]

{02139, 1996/**/**}, {02139} $\Rightarrow$ {1996/**/**} [40%, 66.7%]

{02139, 1996/08/**}, {02139} $\Rightarrow$ {1996/08/**} [40%, 66.7%]

The above rules have redundancy and need extra pruning. The robust rule is:

{0213*, ****/**/**} $\Rightarrow$ {02139, 1996/08/**} [40%, 66.7%]

Below is the rule in a compatible traditional form. Our GenTree algorithm learns this rule automatically.

{0213*} $\Rightarrow$ {02139, 1996/08/**}

5.2 Future Work

Multiple Hierarchies for an Attribute. Rather than using a single VGH for an attribute, there are situations in which multiple VGH's seem natural because they capture different criteria for describing a value. Even though these descriptions can be encoded into a single hierarchy and GenTree used to mine robust rules, different rules result depending on the level at which the encoding takes place. Here is a simple example. Consider a relational table of customer demographics and purchased beverages. An "is-a" hierarchy associates both milk and juice as beverages in the top levels, and brands of juices and milks in the lower levels. But some juices and milks have the same brands. We could alternatively have the hierarchy organized first by brand and then by type of beverage. Notice how the same rules learned from these two different hierarchies may have different values for support and confidence! This happens because the ordering of characteristics within a hierarchy matters not only semantically, but also quantifiably. Replacing this imposed ordering of characteristics with multiple hierarchies (e.g., separate beverage and brand hierarchies) for the same attribute poses another new exploration in rule mining. An extension to GenTree construction seems particularly well suited to accommodate attributes having multiple hierarchies. Additional nodes representing the various forms of cross-level generalizations using the hierarchies would be added. Mining robust rules from the revised GenTree would then use the same algorithm as presented herein.

Size of a GenTree. The extreme upper bound of a GenTree is much, much larger than the size of a GenTree in practice because of the ways in which tuples combine. In the real-world, tuples tend not to represent every possible combination of values. Also, in real-world data the number of tuples tends to be much larger than the number of attributes, thereby increasing the number of tuples likely to share common values. The actual size of a GenTree is related to the fractal dimension of the data. Exploring this relationship further allows us to determine a tighter upper bound based on the fractal dimension. There will be two key components here: 1) how to estimate the fractal dimension of a dataset with attribute hierarchies in a fast yet accurate way, and 2) how to incorporate the fractal dimension in the upper bound formula.

Privacy and k-anonymized data. The notion of *k*-anonymized data is simple. Given a table in which each tuple contains information about a single person and no two tuples relate to the same person, a table is *k*-anonymized if for every tuple there are at least *k*-1 other tuples in the table having the same values over the attributes considered sensitive for re-identification [10]. A *k*-anonymized table provably provides privacy protection by guaranteeing that for each tuple there are at least *k* individuals to whom the tuple could refer. The use of *k*-anonymity fits nicely into stated laws and regulations because these expressions need only identify a population, attributes, and a value for *k*, all of which are easy to formulate in human language. Despite its popularity, however, the use of *k*-anonymized data has been limited due to a lack of methods to learn from the data. Because GenTree stores the number of tuples at each node that account for the generalized form of the node, it is easy to see that GenTree could be used to produce *k*-anonymized data. More importantly, GenTree would be the first to mine association rules from *k*-anonymized data. These are natural by-products of this work.

In this paper, we have introduced the problem of mining robust rules from a large relational table. We used the partial ordering imposed by "more-general-than" attribute hierarchies to extend the rule space. A GenTree, introduced herein, organizes search through the extended rule space for robust rules, which are cross-level generalizations based on quantifiable corroboration (support and confidence) and on semantic expressiveness. In real-world data, many more and different kinds of useful rules are learned using GenTree than is possible with prior methods.

6. Acknowledgements

References

[1] Agrawal, R. Imielinski, T. and Swami, A. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Washington, DC, May 1993.

[2] Han, J. and Fu, Y. Discovery of multiple-level association rules from large databases. *In Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[3] Mitchell, T. Generalization as search. *Artificial Intelligence, 18,* 2, 1982, 203-226.

[4] Plotkin, G. A note on inductive generalization. In Meltzer & Michie (Eds.), *Machine Intelligence,* 5, 1970 (pp.153-163). Edinburgh University Press.

[5] Rosen, K. *Discrete Mathematics and Its Applications,* (pp.1-9). New York: McGraw-Hill 1991.

[6] Sebag, M. Delaying the choice of bias: A disjunctive version space approach. In *Proc. of the 13th Int'l Conference on Machine Learning*, San Francisco: Morgan Kaufman1996.

[7] Simon, H. and Lea, G. Problem solving and rule induction: a unified view. In Gregg (Ed.), *Knowledge and Cognition,* (pp.105-127). New Jersey: Lawrence Erlbaum Assoc 1973.

[8] Srikant, R. and Agrawal, R. Mining Generalized Association Rules. In *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[9] Srikant, R. and Agrawal, R. Mining quantitative association rules in large relational tables. In *Proc. of the ACM SIGMOD Conference on Mgt of Data*, Montreal,Canada, June 1996.

[10] Sweeney, L. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10 (5), 2002; 571-588.

[11] Voter List for Cambridge, Massachusetts, Registry of Voters, Massachusetts, 1997.

[12] Voter List for Pittsburgh, Pennsylvania ZIP15213, Registry of Voters, Pennsylvania, 2001.