Computational Intelligence, Volume 24, Number 1, 2008

LEARNING STRUCTURED BAYESIAN NETWORKS: COMBINING ABSTRACTION HIERARCHIES AND TREE-STRUCTURED CONDITIONAL PROBABILITY TABLES

MARIE DESJARDINS AND PRIYANG RATHOD

Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD, USA

LISE GETOOR

Department of Computer Science, University of Maryland, College Park, MD, USA

Context-specific independence representations, such as tree-structured conditional probability distributions, capture local independence relationships among the random variables in a Bayesian network (BN). Local independence relationships among the random variables can also be captured by using attribute-value hierarchies to find an appropriate abstraction level for the values used to describe the conditional probability distributions. Capturing this local structure is important because it reduces the number of parameters required to represent the distribution. This can lead to more robust parameter estimation and structure selection, more efficient inference algorithms, and more interpretable models. In this paper, we introduce Tree-Abstraction-Based Search (TABS), an approach for learning a data distribution by inducing the graph structure and parameters of a BN from training data. TABS combines tree structure and attribute-value hierarchies to compactly represent conditional probability tables. To construct the attribute-value hierarchies, we investigate two data-driven techniques: a *global clustering* method, which uses all of the training data to build the attribute-value hierarchies, and can be performed as a preprocessing step; and a *local clustering* method, which uses only the local network structure to learn attribute-value hierarchies. We present empirical results for three real-world domains, finding that (1) combining tree structure and attribute-value hierarchies improves the accuracy of generalization, while providing a significant reduction in the number of parameters in the learned networks, and (2) data-derived hierarchies perform as well or better than expert-provided hierarchies.

Key words: Machine learning, Bayesian networks, abstraction hierarchies, background knowledge, clustering, MDL.

1. INTRODUCTION

Bayesian networks (BNs) are a widely used representation for capturing probabilistic relationships among variables in a domain of interest (Pearl 1988). They can be used to provide a compact representation of a joint probability distribution by capturing the dependency structure among the variables, and can be learned from data (Cooper and Herskovits 1992; Heckerman 1995). Two forms of learning are possible: supervised learning (also called inductive learning or classification, where one attribute is distinguished as the *class variable* to be predicted) and unsupervised learning (where the goal is to discover patterns in the data without a class label). Here, we focus on the unsupervised learning problem of *density estimation*—that is, representing a joint probability distribution by inducing the graph structure and parameters of a BN from training data. Note that the resulting BN is not necessarily causal; the goal is simply to construct an efficient, compact representation of an observed data distribution.

The conditional probability distributions associated with the variables in a BN are most commonly represented as explicit conditional probability tables (CPTs), which specify multinomial distributions over the values of a variable for each combination of values of its parents. However, researchers have found that explicitly representing *context-specific independence* relationships can reduce the number of parameters required to describe a BN (Boutilier et al. 1996). This reduction in parameters can result in more efficient inference (Poole and Zhang

Address correspondence to Marie desJardins, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA; e-mail: mariedj@cs.umbc.edu

2003) and more robust learning. In particular, learning methods have been developed that use tree-structured CPTs (Friedman and Goldszmidt 1996) (TCPTs) and graph-structured CPTs (Chickering, Heckerman, and Meek 1997) to represent the context-specific independence relationships among the variables.

Previous methods for learning BNs assume that categorical features are already represented at an appropriate level of abstraction. This is typically accomplished by a domain expert who manually identifies a "minimal" set of value groups that capture important distinctions in the domain. This process is essential for effective learning, because many-valued categorical variables result in very large CPTs, and therefore are impractical for standard BN learning methods.

As an example of an application that would benefit from automated value grouping, consider an automotive domain, in which the goal is to capture the relationships among various properties of different car models, such as the manufacturer, car type, model year, gas mileage, performance, reliability, and cost. If this information is extracted from an existing database or online source, many of the variables are likely to be either many-valued categorical variables (such as manufacturer and car type), or continuous numeric variables (such as gas mileage and cost). Discretization methods exist for numeric variables; however, little work has been done on finding appropriate groupings for categorical variable values.

In our original investigation (desJardins, Getoor, and Koller 2000) we applied our methods to an epidemiological data set of tuberculosis patients. In that application, the categorical variables were place of birth and ethnicity. Because the data was from an existing database, these variables had a large number of values. Our method was able to automatically identify an appropriate abstraction level for each of these variables.[1]

We propose the use of *attribute-value hierarchies* (AVHs), combined with automated search, to solve the value grouping process for categorical variables in BN learning, replacing the manual feature engineering process described above. An AVH relates base-level values of a categorical variable to abstractions (groupings) of these values.

In the automotive domain, an example of a many-valued categorical variable would be the country manufacturer. The base-level values are specific manufacturers such as *Honda*, *Toyota*, *Ford*, and *BMW*. Abstract values might include *Japanese-Manufacturer*, *American-Manufacturer*, and *European-Manufacturer*. Attribute-value hierarchies are taxonomies describing the relationships between base-level values and abstract values.

In earlier work (desJardins et al. 2000) we introduced Abstraction-Based Search (ABS), an algorithm that uses AVHs during BN learning. ABS searches the space of possible abstractions at each variable in the BN. The abstraction process effectively collapses the corresponding rows of the CPT, thus reducing the number of parameters needed to represent the BN.

The abstractions provided by AVHs in ABS are complementary to those provided by TCPTs. Therefore, we have also developed TCPT ABS (Tree-Abstraction-Based Search, TABS) (desJardins, Getoor, and Rathod 2005), which integrates ABS with TCPTs. Our earlier empirical results showed that TABS significantly reduces the number of parameters required to represent a learned BN, compared to standard BN learning, ABS, or TCPT learning alone.

Here, we present a more detailed comparison of learning algorithms that use tree-structured conditional probability tables (TCPT), learning algorithms that make use of AVHs (ABS), and learning algorithms that combine both (TABS). In addition, we introduce and contrast two data-driven methods for constructing AVHs. The first method is a global technique

[1] Unfortunately, this data is proprietary, and we no longer have access to it, so we were not able to test our newer methods on it.

that uses all of the training data to cluster each variable. The second method is a local method that uses only the local structure to construct an AVH for each parent link in the BN. Our original motivation for developing these techniques was to enable the use of AVHs in domains where expert-provided AVHs are not available. However, in our experiments, we found that in general, the learned AVHs yielded equally or more accurate BNs (using a log likelihood measure) as expert AVHs—and the learned AVHs resulted in substantially fewer parameters in the BNs than expert AVHs. Interestingly, the global AVHs sometimes outperformed local AVHs for standard ABS, but with TCPTs, local AVHs always yielded better performance than either global or expert hierarchies. We speculate this this is because the TCPTs are specifically intended to take advantage of local (context-specific) independence, so they benefit more from the local hierarchies. Across all three dimensions, TCPTs with abstraction and local hierarchies consistently yielded the best log likelihood and the fewest (or close to fewest) parameters.

The remainder of the paper is organized as follows. We first give background on BNs, TCPTs, AVHs, and learning methods for BNs with and without TCPTs (Sections 2–4). Next, we introduce the ABS and TABS methods for incorporating AVHs into the learning process (Sections 5 and 6), and the global and local clustering algorithms for deriving AVHs from training data (Section 7). We then provide experimental results in three real-world domains (Section 8), summarize related work (Section 9), and present conclusions and future work (Section 10).

2. BAYESIAN NETWORKS

We assume that the reader has some familiarity with basic concepts of BNs (Pearl 1988) and local search-based methods for learning BNs for density estimation (Heckerman 1995). In this section, we briefly introduce the notation, learning methods, and scoring functions that are used in the remainder of the paper.

A BN consists of an annotated directed acyclic graph that represents the joint probability distribution of a set of random variables, $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. We assume that the X_i are all discrete finite-valued variables. The domain of variable X_i is denoted by

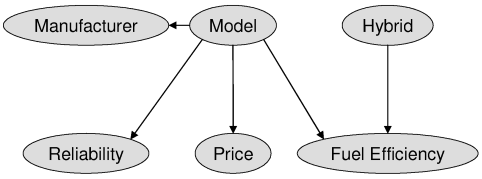
$$Dom(X_i) = \{v_{i1}, v_{i2}, \ldots, v_{ir_i}\},$$

where r_i is the number of values that X_i may take on ($r_i = |Dom(X_i)|$). A BN over the set of variables $\mathbf{X}$ is represented as a pair, $B = (G, \Theta)$. The BN structure, G, is a directed acyclic graph over $\mathbf{X}$, where the edges represent dependencies between variables. The BN parameters, Θ, specify the set of conditional probabilities associated with B. A variable X_i is conditionally independent of the other variables in the network, given its parents Π_i. Using this independence assumption, the joint probability distribution can be factored as:

$$P(\mathbf{X}) = \prod_i P(X_i \mid \Pi_i). \tag{1}$$

There are a number of ways to repesent the parameters (conditional probability distributions) in the network. When the random variables are discrete, the most common is using a table of multinomials, which are referred to as CPTs. (We will later use the term *flat CPT* to distinguish these simple tables from abstract and tree-structured CPTs.)

The CPT for variable X_i contains a row for each possible instantiation π_i of X_I's parents Π_i; the row gives the conditional distribution of X_i given that particular instantiation of Π_i. We use $TB(X_i)$ to denote the CPT associated with X_i, and $\Theta_{TB(X_i)}$ to denote the associated



(a)

M	H	P(FE|M,H)
miata	yes	0.8
911	yes	0.8
accord	yes	0.8
maxima	yes	0.8
explorer	yes	0.8
suburban	yes	0.8
miata	no	0.6
911	no	0.3
accord	no	0.6
maxima	no	0.5
explorer	no	0.2
suburban	no	0.1

(b)

FIGURE 1. (a) A simple Bayesian network. (b) The CPT for *P(Fuel Efficiency | Model, Hybrid)*.

collection of parameters. If r_i is the number of possible instantiations of X_i, and q_i is the number of possible instantiations of the parents Π_i, then $|\Theta_{TB(X_i)}| = r_i \cdot q_i$ is the total number of parameters in the table. (Because of the constraint that the probabilities in each row of the CPTs must sum to one, there are fewer then $r_i \cdot q_i$ free parameters. Therefore, one can encode a CPT compactly using $q_i \cdot (r_i - 1)$ parameters.)

Example 1. Figure1(a) shows a simple Bayesian network describing the *Reliability*, *Price*, and *Fuel Efficiency* of a car, based on its *Manufacture*, its *Model*, and whether or not the car is a *Hybrid*. The *Model* determines the *Manufacturer*, so there is a directed edge from *Model* to *Manufacturer*. Figure1(b) shows the CPT for *Fuel Efficiency*, which depends on *Model* and *Hybrid*.

2.1. Tree-Structured CPTs

While BNs with flat CPTs are easy to understand, and are useful for compactly representing a multivariate joint distribution, they can only capture certain kinds of conditional independence. Let $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ be sets of random variables. $\boldsymbol{X}$ is *conditionally independent* of $\boldsymbol{Y}$ given $\boldsymbol{Z}$ in a distribution P if:

$$P(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z}) = P(\boldsymbol{x} \mid \boldsymbol{z})$$

for all values $\boldsymbol{x} \in Val(\boldsymbol{X})$, $\boldsymbol{y} \in Val(\boldsymbol{Y})$, and $\boldsymbol{z} \in Val(\boldsymbol{Z})$. This relationship is denoted $I(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{Z})$. This kind of conditional independence is easily captured by a BN with flat CPTs. However, the requirement that this constraint holds *for all possible values* of $\boldsymbol{Z}$ is sufficient but not necessary to establish conditional independence. A more flexible definition is the notion of *context-specific* independence (Boutilier et al. 1996). Let $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$, and $\boldsymbol{C}$ be disjoint sets of random variables. $\boldsymbol{X}$ is *contextually independent* of $\boldsymbol{Y}$ given $\boldsymbol{Z}$ in context $\boldsymbol{c} \in Val(\boldsymbol{C})$ if:

$$P(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{c}) = P(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{c}) \tag{2}$$

for all values $\boldsymbol{x} \in Val(\boldsymbol{X})$, $\boldsymbol{y} \in Val(\boldsymbol{Y})$, and $\boldsymbol{z} \in Val(\boldsymbol{Z})$. This relationship is denoted $I(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{Z}, \boldsymbol{c})$.

The most common representation for context-specific independence is using *tree-structured CPTs (TCPTs)*. The leaves in a TCPT represent different conditional distributions over the values of X_i; the path from the root to a leaf defines the parent context $\boldsymbol{c}$ for that

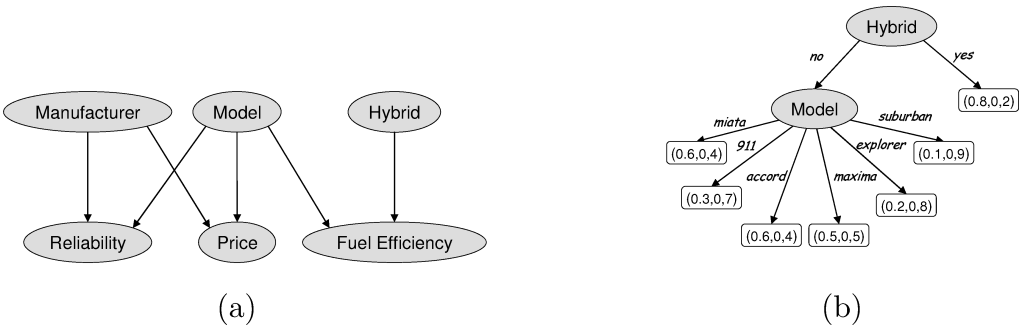


FIGURE 2. (a) A simple Bayesian network; note that because we are using TCPTs instead of flat CPTs, the structure of the BN may be different. (b) The TCPT for *P(Fuel Efficiency | Model, Hybrid)*.

distribution. That is, a particular context $\boldsymbol{c}$ specifies the values for a subset of X_i's parents, $\boldsymbol{C} \subseteq \Pi_i$. The other parents ($\Pi_i \backslash C$) correspond to the $\boldsymbol{Z}$ variables in equation (2). We use $TR(X_i)$ to denote the TCPT associated with X_i, and $\Theta_{TR(X_i)}$ to denote the associated collection of parameters. If l_i is the number of leaves of the tree $TR(X_i)$, then $|\Theta_{TB(X_i)}| = l_i \cdot r_i$ is the total number of parameters in the tree. For a particular leaf l, we use $\pi_i(l)$ to denote the *set* of parent instantiations π_i that correspond to that leaf, and θ_l to denote the conditional distribution described at that leaf.

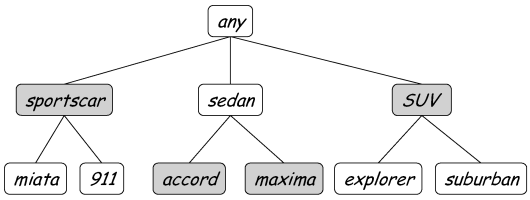
Example 2. Figure 2(a) shows a Bayesian network over the same set of random variables as Example 1. Figure 2(b) shows a TCPT for *Fuel Efficiency* based on *Model* and *Hybrid*. In this case, we see that if the car is a hybrid (*Hybrid* = *yes*), then *Fuel Efficiency* is independent of *Model*; however, when *Hybrid* = *no*, then *Fuel Efficiency* depends on *Model*. Note also that the BN structure is different; this will often be the case with TCPTs, because they are able to capture more intricate forms of contextual independence.

A TCPT can result in a more compact representation than a flat CPT. We define the *compression ratio* as the ratio of the number of parameters required by a flat CPT to a TCPT. In our example, the compression ratio for *Fuel Efficiency* is 1.7, because the flat CPT required 24 parameters, while the TCPT requires 14.

2.2. BNs with Attribute-Value Hierarchies

Abstraction hierarchies are a commonly used form of knowledge that is intended to capture useful and meaningful groupings of feature values in a particular domain. An attribute-value hierarchy (AVH) is essentially an IS-A hierarchy for categorical feature values. The leaves of the AVH describe base-level values, and the interior nodes represent abstractions of base-level values. Each abstract value corresponds to a set of base-level values. The root node of the AVH is a wildcard value that is equivalent to the set of all base-level values. An AVH need not be balanced (i.e., the path length from leaf values to the root can vary), and the branching factor (number of children) can vary within the hierarchy. Given a value a in the AVH, $\mathsf{spec}(a)$ returns its specializations (i.e., its children in the AVH), and $\mathsf{gen}(a)$ returns its generalization (i.e., its parent). The specialization of a leaf is empty, and the generalization of the root is empty.

A cut through the AVH defines a set of feature values called an *abstraction*, which is formally equivalent to a partitioning of the base-level attribute values. An abstraction is *legal* if every base-level value is covered by exactly one value in the abstraction. We use $abs(X_i)$ to



(a)

M	H	P(FE|M,H)
sportscar	yes	0.8
sedan	yes	0.8
SUV	yes	0.8
sportscar	no	0.35
sedan	no	0.55
SUV	no	0.15

(b)

FIGURE 3. (a) A simple attribute-value hierarchy for the car model attribute. (b) The abstract CPT for *P(Fuel Efficiency | Model, Hybrid)*, where *Model* is represented at the abstraction level $A(Model: Fuel\ Efficiency) = \{sportscar, sedan, SUV\}$.

denote the set of legal abstractions of X_i. (Note that an abstraction can include both abstract (interior) and base-level (leaf) values from the AVH.)

An advantage of using AVHs is that we can define a CPT for a node over abstract values of the parent, rather than having a row for each parent value. When a parent is represented at an abstract level, the rows in the CPT are effectively collapsed together.[2] We refer to this representation for a CPT as an *abstract CPT*, and use $AB(X_i)$ to denote the abstract CPT associated with X_i. Abstract CPTs have the advantage of reducing the number of parameters required to describe the conditional probability distribution at a node. Abstract CPTs represent a different form of context-specific independence than TCPTs. Specifically, they indicate that certain distinctions among the parent values are unimportant.

For each $Y \in \Pi_i$, we define an associated abstraction level $A(Y : X_i)$. (A BN node can have a different abstraction level for each of its children, so the abstraction level depends on which child node's conditional probability distribution is being modeled.) An abstract instantiation of the parents of X_i, denoted π_{a_i}, is an assignment of a value from $A(Y : X_i)$ for each $Y \in \Pi_i$. In the abstract CPT, there is a row in the table for each possible abstract instantiation π_{a_i} of the parents. Similar to a context in a TCPT, π_{a_i} defines a set of of parent instantiations π_i for which X_i has the same conditional distribution, denoted $\theta_{\pi_{a_i}}$. For a particular abstract instantiation, π_{a_i}, we use $\pi_i(a_i)$ to denote the set of base-level instantiations which map to $\theta_{\pi_{a_i}}$.

Example 3. Figure 3(a) shows an AVH for the car model in our running example. The set of shaded nodes $\{sportscar, accord, maxima, SUV\}$ corresponds to a legal *abstraction*; it is a cut through the tree. Figure 3(b) shows the abstract CPT *AB(Fuel Efficiency | Model, Hybrid)* that represents the conditional probability distribution *P(Fuel Efficiency | Model, Hybrid)*. In this example, the abstraction used for *Model* is $\{sportscar, sedan, SUV\}$.

An abstract CPT can result in a more compact representation than a flat CPT. In our example, the compression ratio for *Fuel Efficiency* is 2.0, because the flat CPT required 24 parameters, while the abstract CPT requires 12.

2.2.1. Combining TCPTs and AVHs. Both TCPTs and abstraction hierarchies allow us to capture context-specific independence. A natural question is whether they capture the same type of independence. It turns out that they do not. The TCPTs allow us to completely

[2]In principle, the values of the child node can also be abstracted; in this case, columns of the CPT are collapsed. The abstract values are then partially specified in inference and learning tasks; this can be handled by a uniform distribution or by applying another method for partially specified or missing data. In this work, however, we consider only parent abstractions.

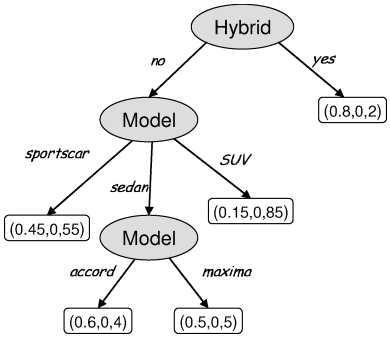


FIGURE 4. An abstract TCPT for *P(Fuel Efficiency | Model, Hybrid)*. In this example, the tree splits first on *Hybrid*; next, *Model* is refined to {*sportscar*, *sedan*, *SUV*}; finally, *sedan* is further refined to {*accord*, *maxima*}.

ignore a random variable in certain contexts, while abstract CPTs represent finer-grained constraints, which allow us to ignore the distinctions between certain *values* of a random variable.

We show in this paper how to combine the two approaches by building a TCPT that splits on abstract values, rather than the base-level values. We refer to this approach as *abstract tree-structured CPTs*, and use $AT(X_i)$ to denote the abstract TCPT associated with X_i.

Figure 4 shows an abstract TCPT, *AT (Fuel Efficiency | Model, Hybrid)*, for the conditional probability distribution *P (Fuel Efficiency | Model, Hybrid)*. An abstract TCPT can result in a more compact representation than a flat (or abstract) CPT. In our example, the compression ratio for *Fuel Efficiency* is 2.4, because the flat CPT required 24 parameters, while the abstract TCPT requires 10.

3. LEARNING BNs FROM DATA

Given a set of N instances, $D = \{d_1, d_2, \ldots, d_N\}$, where each d_j is a vector of values for each random variable:

$$d_j = \langle x_{1j}, x_{2j}, \ldots, x_{nj} \rangle \in Dom(X_1) \times \ldots \times Dom(X_n),$$

we would like to learn a BN that best matches this data. Because this problem is NP-hard (Chickering, Geiger, and Heckerman 1994), typically a simple greedy hill-climbing search is used. At each step, the new graph is evaluated using a scoring function; if the modification leads to a better network, then it is retained. When there are no missing values in the data, the commonly used scoring functions can be decomposed locally, so that when an edge is added or deleted, only the score of the variable X_i whose parent set Π_i changed needs to be re-scored.

3.1. Scoring Functions

A variety of scoring functions have been used in local search for BN learning; the most commonly used are the Bayesian score and Minimum Description Length (MDL) score (or its equivalent, the Bayesian information criterion). In our work, we use the MDL score, because it provides a natural framework for incorporating the notion of model complexity. The MDL score is also related to regularization, which is frequently used in machine learning to avoid overfitting and to bias learning toward simpler models.

The MDL score favors the hypothesis (i.e., the BN, B) that minimizes the description length of the data. The description length is the sum of the encoding length of the data using the BN (which essentially measures the training error) and the encoding length of the BN itself (which captures the idea of model complexity). The mathematical derivation of the MDL score was presented by Bouckaert (1994) and by Lam and Bacchus (1994); our presentation here is based on Friedman and Goldszmidt (1996). We first describe the MDL score for flat CPTs, and then describe the MDL score for abstract CPTs, TCPTs, and abstract TCPTs.

Note that our general approach, of searching through the abstraction space, could be applied to BN learning using any scoring function. This would require only extending the scoring function to incorporate a notion of the score of an abstraction level, as we have done here for the MDL score.

3.1.1. MDL for Flat CPTs. There are two components of the encoded model: the BN itself and the data encoded using the BN. The description length DL is thus given by:

$$DL(D) = DL(B) + DL(D \mid B). \tag{3}$$

The description length of the BN, $DL(B)$, can be further decomposed into the description length of the structure G, plus the description length of the parameters. To describe the structure, for each X_i, we need to record the parents Π_i. Because there are n random variables, $\log n$ bits are required to record each parent,[3] so the description length of the structure is simply:

$$DL(G) = \sum_{i=1}^{n} |\Pi_i| \cdot \log n, \tag{4}$$

where $|\Pi_i|$ is the number of parents of X_i.[4] To encode the parameters of the BN, Θ, we have

$$DL(\Theta) = \sum_{i=1}^{n} DL(TB(X_i)). \tag{5}$$

For flat CPTs,

$$DL(TB(X_i)) = DL(\theta_{TB(X_i)}) = \frac{1}{2}(r_i - 1) \cdot q_i \cdot \log N, \tag{6}$$

where r_i is the number of possible instantiations of X_i, q_i is the number of possible instantiations of the parents Π_i, and $\frac{1}{2} \log N$ is the number of bits required to represent a probability entry (see Friedman and Yakhini 1996, for a discussion of why the penalty term is logarithmic in the sample size N).

The description length of the data, $DL(D \mid B)$, is the number of bits required to encode the data D given B. This can be approximated by:

$$DL(D \mid B) = \sum_{i=1}^{N} \log P(d_i), \tag{7}$$

[3] All of the logarithms are assumed to be base 2.

[4] Friedman and Goldszmidt (1996) actually use a more compact encoding, which stores the number of parents and an index into all $\binom{n}{k}$ subsets of k variables.

where P is the distribution defined by the BN B. Factoring this according to the BN structure, equation (7) can be rewritten in terms of the conditional entropy of each random variable:

$$DL(D \mid B) = N \cdot \sum_{i=1}^{n} H(X_i \mid \Pi_i), \tag{8}$$

where $H(X \mid Y) = -\sum_{x,y} p(x, y) \log p(x \mid y)$. If we choose the parameters that minimize the encoding (which is the obvious choice, because these are the parameters that maximize the likelihood of the data), equation (8) becomes:

$$DL(D \mid B) = N \cdot \sum_{i=1}^{n} \sum_{x_i, \pi_i} -\frac{N_{x_i,\pi_i}}{N} \cdot \log \frac{N_{x_i,\pi_i}}{N_{\pi_i}}, \tag{9}$$

where the $N_{\cdot}$ terms are the empirical counts of the number of times the specified assignment to a subset of the random variables is observed in the data D, and x_i and π_i are particular instantiations of X_i and Π_i. (Note that following common practice, we use Laplace smoothing (adding one to every count) to avoid zero counts.)

3.1.2. MDL Score for TCPT Learning. When the CPTs at each node in the BN are represented as trees (T_{X_i}), the encoding of the local probabilities becomes:

$$DL(TB(X_i)) = DL_{struct}(T_{X_i}) + DL(\theta_{TB(X_i)}), \tag{10}$$

where $DL_{struct}(T_{X_i})$ is the description length of the structure of the TCPT T_{X_i}.

We use the MDL scoring function for TCPTs described by Friedman and Goldszmidt (1996). For each node, one bit is needed to record whether it is a leaf node or an internal node. For each internal node in the tree, we need to describe which parent is tested, and then describe its subtrees. Along a path from the root to a leaf, each parent can only be tested once. We use n_π to denote the number of untested parent nodes at an internal node. The structure component is calculated by the following recursive formula:

$$DL_{struct}(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf} \\ 1 + \log(n_\pi) + \sum_{t_k \in children(t)} DL_{struct}(t_k) & \text{otherwise.} \end{cases} \tag{11}$$

The second component in equation (10), $DL(\theta_{TBX_i})$, is the description length of the parameters for the TCPT, which is given by:

$$DL(\theta_{TBi}) = \frac{1}{2}(r_i - 1) \cdot l_i \cdot \log N \tag{12}$$

(Recall that the term l_i refers to the number of leaves in the TCPT.) This is essentially the same as equation (6), the description length for the parameters of the flat TCPT.

Finally, the $DL(D \mid B)$ component of the score is the same as that for a flat CPT (equation (8)).

3.1.3. MDL for Abstract CPTs. For abstract CPTs, the MDL computation is the same as for flat CPTs, except that we need to record the abstraction level for each parent. $DL(\theta)$ is given by:

$$DL(AB(X_i)) = DL_{abs-levels}(AB(X_i)) + DL(\theta_{AB(X_i)}), \tag{13}$$

where

$$DL_{abs\text{-}levels}(AB(X_i)) = \sum_{Y \in \Pi_i} \log |abs(Y : X_i)| \tag{14}$$

and

$$DL(\theta_{AB(X_i)}) = \frac{1}{2}(r_i - 1) \cdot a_i \cdot \log N, \tag{15}$$

where r_i is the number of possible instantiations of X_i, a_i is the number of possible abstract instantiations of the parents Π_i, and $\frac{1}{2}\log N$ is the number of bits required to represent a probability entry.

3.1.4. MDL Score for Abstract TCPTs. The MDL computation for abstract TCPTs uses the same graph and data description lengths as the other types of CPT. The description length of an abstract TCPT is the same as equation (11), the description length for a TCPT. The only difference is that each parent can occur multiple times along a path from the root to the leaf, at successively lower levels of abstraction. Therefore, instead of n_π possible splits, where n_π is the number of untested parent nodes at an internal node, we have a_π possible splits, where a_π is the number of parent nodes whose value in the current context $\boldsymbol{c}$ is not a base-level value. (In other words, a_π is the number of parent nodes with remaining refinements.)

$$DL_{struct}(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf} \\ 1 + \log(a_\pi) + \sum_{t_k \in children(t)} DL_{struct}(t_k) & \text{otherwise.} \end{cases} \tag{16}$$

4. LEARNING BNs WITH TCPTs

As mentioned in the previous section, learning the BN structure is typically accomplishing using a greedy hill-climbing search. For learning BNs with flat CPTs, the local search operators are AddEdge(X, Y), which adds X as a parent of Y; DeleteEdge(X, Y), which removes X from the parent set of Y; and ReverseEdge(X, Y), which reverses the edge between X and Y. The constructed graph must be acyclic, so before adding or reversing an edge, we must check that the resulting graph will remain acyclic.

When learning tree-structured CPTs, the search becomes a bit more complex; in addition to deciding which node should be the parent of another, we must build a good decision tree representation for the conditional distribution of each node given its parents. There are two distinct approaches, which we refer to as *batch* CPT learning and *incremental* TCPT learning.

4.1. Batch TCPT Learning

In batch learning, each time the parents of a node are changed—by either adding or removing a parent—the decision tree is re-learned. This is the approach described by Friedman and Goldszmidt (1996). Because learning an optimal decision tree is in itself a hard problem, the batch tree learning algorithm is also a locally greedy search algorithm. The algorithm is an instance of the classical recursive decision tree learning algorithm: at each step, it chooses a random variable to split on, and then, after filtering the instances appropriately, recursively builds subtrees for the partitioned instances. The choice among the possible random variables to split on is made by choosing the one with the best MDL score. After building the full decision tree, there is a "post-pruning" step, which removes any splits that do not improve the overall MDL score. The batch tree learning approach is computationally expensive because the decision tree must be recomputed at each step.

Procedure AddSplit(Y, X, l)
1. Remove l from $RefCandidates(X)$
2. Create a new internal node t, which corresponds to splitting l on Y
3. Replace l with t, and create $|Val(Y)|$ new leaf nodes below t
4. For each child l' of t,
   a. Set $SplitCandidates(l') = SplitCandidates(l) - Y$
   b. Add l' to $RefCandidates(X)$

Procedure ExtendTree(T_X)
1. Select a leaf node l for refinement from $RefCandidates$ with probability $p(l) \propto SplitCandidates(l)$
2. $Y = \text{argmin}_{Y \in SplitCandidates(l)}\{Score(T_X \cup Y)\}$ where $T_X \cup Y$ is the TCPT that results after splitting l on Y
3. If $Score(T_X) \cup Y < Score(T_X)$, then AddSplit$(Y, X, l)$

FIGURE 5. Incremental TCPT learning algorithm: the AddSplit and ExtendTree procedures.

4.2. Incremental TCPT Learning

An alternate approach is to fully integrate the tree construction process into the BN structure search. We present an incremental approach that (1) does not require re-learning the entire TCPT each time a node is added; (2) does not require post-pruning; and (3) allows for abstraction to be incorporated seamlessly.

Instead of using the AddEdge and DeleteEdge operators to add and remove edges from the network structure, we use AddSplit and RemoveSplit to add and remove split nodes in the individual TCPTs, as the basic operations in the hill-climbing search.[5] When the learning algorithm adds a split on Y to a leaf of the TCPT for X, when Y has not yet been split in any of the other nodes in the TCPT, this effectively adds an edge from Y to X in the network (the AddEdge operation in the original search). Similarly, removing the last occurrence of Y from the TCPT of X is equivalent to removing the edge from Y to X (DeleteEdge).

In our representation, each node X has a list of *RefCandidates*(X), which are the leaves of T_X, and are potential candidates for further refinement. Each leaf node l of the TCPT has an associated set of *SplitCandidates*(l), which specifies the candidate variables that are available for further splitting. When refining the TCPT, the algorithm tries all possible variables in each leaf's *SplitCandidates* set, and then selects the refinement that offers the largest improvement in the MDL score. Initially, a variable X with no parents starts with a single root node in its TCPT which is the only *RefCandidates*; all other variables Y are in the root's *SplitCandidates* set. As the TCPT is refined, the procedure AddSplit (Figure 5) propagates these candidates to the new leaf nodes after removing the candidate associated with the selected refinement. The need for separate *SplitCandidates* sets for each leaf node will become obvious in Section 6 when we explain how abstraction hierarchies are incorporated into TCPTs.

The optimal single-step refinement can be found by evaluating all possible refinements for all variables in the BN, and choosing the one that most improves the MDL score. However, this is computationally infeasible; therefore, we instead randomly choose the next variable to refine, and use the stochastic procedure ExtendTree (Figure 5) to select a leaf node to

[5] ReverseEdge is not included as an operator in the TCPT learning algorithm, because it would be inconsistent with the incremental approach.

refine. The probability of a leaf node being selected for refinement is proportional to the size of the *SplitCandidates* set at that node. The selected node is then refined using the candidate split (if one exists) that leads to the largest improvement in the MDL score of the tree. This process of selecting a variable and then applying AddSplit and RemoveSplit to modify its TCPT is performed until a local minimum is reached in the MDL score for the BN.

5. LEARNING WITH ABSTRACT CPTs

Next we describe how structure learning is done for BNs with abstract CPTs. The ABS algorithm (desJardins et al. 2000) extends the standard search over network structures as follows. When an edge is added to the network, the parent X_i is added at its most abstract level, using the top-level values in the AVH. For example, if *Model* is chosen as a parent of another node, the initial abstraction level would be {*sportscar*, *sedan*, *SUV*}.

ABS extends the standard set of BN search operators—AddEdge, DeleteEdge, and ReverseEdge—with two new operators: Refine(Y, X, a) and Abstract(Y, X, a). Refine(Y, X, a) replaces the abstract value a of Y in X's CPT with a's children, spec(a). If a is a leaf node, then Refine has no effect. Similarly, Abstract(Y, X, a) replaces the value a of Y, and its siblings, in X's CPT with a's parent, gen(a). If a's parent is the root node, then Abstract has no effect. Notice that we only abstract the values of a parent; the values of the child (X_i) are never abstracted.

The search process is a greedy search algorithm that repeatedly applies these five operators to the current network, evaluates the resulting network using the MDL score, and replaces the current network with the new one if the latter outscores the former.

6. THE TCPT ABS (TABS) LEARNING ALGORITHM

TCPTs take advantage of the fact that the value of a node can be independent of the values of a subset of its parents, given a local context. By incorporating AVHs into TCPT learning, we can also take advantage of the fact that certain parent *values* may have similar influences on the conditional probabilities of the child node. This reduces the branching factor of nodes with AVHs, and allows the decision about whether to make a distinction between certain values to be postponed until it is required. As a result, the number of parameters needed for the learned BN is reduced.

Our TCPT ABS (TABS) learning algorithm extends the TCPT learning algorithm described in Section 4, by allowing the nodes in the TCPT to be modeled at different levels of abstraction. In basic incremental tree learning, there are two operators, AddSplit and RemoveSplit. In TABS, we also have two operators: RefineSplit and AbstractSplit. RefineSplit(Y, X, l) specializes the leaf node l in X's TCPT by introducing a single-step refinement of the parent variable Y. If Y does not appear in the context for l, then this refinement represents adding an edge from Y to X at the top-level abstraction of Y. Similarly, AbstractSplit(Y, X, l) removes the split node l, from X's TCPT. If this is the last split that mentions Y, this effectively removes Y as a parent of X.

Example 4. Consider the abstract TCPT in Figure 4. In this example, learning might have begun with

RefineSplit *(Hybrid,FuelEfficiency,*l_1*)*,

then applied

Procedure BuildAVH(X_i,C)
for c from 1 to $r_i - 1$ do
1. find the clusters C_p, C_q in C such that $Dist(C_p, C_q)$ is minimized
2. create a new cluster $C_r = C_p \bigcup C_q$, and make this new cluster the parent of C_p and C_q in the hierarchy
3. remove C_p and C_q from C for the next iteration

FIGURE 6. Clustering algorithm for deriving AVHs from training data.

RefineSplit *(Model,FuelEfficiency,l_2)*

(where l_2 is the leaf in the context with $hybrid = no$), and finally applied

RefineSplit *(Model,FuelEfficiency,l_4)*

(where l_4 is the context with *hybrid* = *no* and *model* = *sedan*).

7. GENERATING AVHs USING AGGLOMERATIVE CLUSTERING

Agglomerative clustering methods can easily be used to construct AVHs, because the iterative process of grouping (agglomerating) clusters intrinsically corresponds to a bottom-up hierarchy construction process.[6] Therefore, we have developed two agglomerative clustering-based methods—*global clustering* and *local clustering*—for deriving AVHs from the training data used to build the BN. Global clustering uses all of the training data to derive a single AVH for each variable. Local clustering builds AVHs based only on the BN neighborhood for each variable, which can change during search, and builds a different AVH for each local neighborhood. One way to view the AVH construction is as the introduction of new hidden variables; we discuss that connection in Section 9.

7.1. Global Clustering

Given a set of instances D, our goal is to find a hierarchical clustering of the values (H_{X_i}) for each variable X_i. We begin by constructing r_i clusters; each cluster contains all of the instances d_j that have a particular value v_{ik} for X_i. $C = \{C_1, C_2, \ldots, C_{r_i}\}$, where $C_k = \{d_j \mid x_{ij} = v_{ik}\}$. These correspond to the initial "leaf clusters."

The BuildAVH procedure shown in Figure 6 uses hierarchical agglomerative clustering (Jain, Murty, and Flynn 1999) to iteratively merge pairs of clusters. (After creating the initial clusters, the variable X_i is ignored in computing the distances.) The resulting AVH will be a binary hierarchy, because the merging step always combines two clusters.

Because our data sets contain only unordered categorical variables, we cannot use the mean or median to compute the centroid (average) of a cluster. Therefore, we use the mode (most frequently occurring value) of each variable. Similarly, there is no notion of the "distance" between any pair of values for given variable. Therefore, distance measures such as Euclidean or Manhattan distance cannot be used. Instead, we use Hamming distance to measure the distance between two cluster centroids for the merging step.

[6]Other popular clustering approaches, such as K-means or spectral clustering, could be used recursively to repeatedly split a set of values into a hierarchy, but they are not designed for this purpose. Similarly, other agglomerative clustering methods could be applied, but the straightforward algorithm we use here seems to work well in practice, as seen in the experimental results.

The Hamming distance between two instances d_i and d_j is simply the number of positions (variables) for which the values differ in the two instances:

$$Dist(d_i, d_j) = \sum_{l=1}^{n} diff(d_{il}, d_{jl}) \tag{17}$$

where

$$diff(d_{il}, d_{jl}) = \begin{cases} 0 & \text{if } d_{il} = d_{jl} \\ 1 & \text{otherwise.} \end{cases} \tag{18}$$

The distance $Dist(C_p, C_q)$ is defined to be the Hamming distance between the centroids of the clusters, C_p and C_q.

7.2. Local Clustering

In local clustering, each time a new edge is added to the network, the AVH associated with the parent node is recalculated for that particular context. Only the child node and the other parents of the child are used in the clustering process. As a result, a node may have a different AVH in the CPT of each node for which it is a parent. The same BuildAVH algorithm is used here as for global clustering (Figure 6). The only difference is that the representation of the clusters C_p has fewer than n variables (specifically, the node itself, its new child, and the parents of the new child).

In ABS, the local clusters can be computed either "one-time" or "frequently." In "one-time" clustering, $A(Y:X)$ is learned only once, when Y is added as a parent of X. If other parents are later added, then Y's AVH will not change. Note that this means the resulting clustering is very order-sensitive, because it is entirely determined by the local context when the first parent is added. In theory, the first parent should be the most "influential" parent, so the AVHs may in fact capture the most important distinctions. However, to explore the sensitivity of the clustering to the this ordering effect, we also implemented "frequent" clustering, in which all parents' AVHs are recomputed every time the local context (i.e., X's parents) changes.

With TCPT learning, a node's AVH is learned only when the node is added to the TCPT for the first time. (Otherwise, the TCPT would need to be recalculated from scratch every time one of the AVHs change, and we would no longer have an incremental algorithm.) Initially, when the TCPT does not contain any nodes, and a new parent, Y is being added to X_i, only the variable X_i is taken into consideration when constructing the AVH of Y. When the TCPT does contain other nodes $\boldsymbol{Z}$, when Y is initially added to X_i, Y's AVH is constructed, based on $\boldsymbol{Z}$ and X_i values.

8. EXPERIMENTAL RESULTS

In this section, we describe results of experiments on three real-world data sets from the UC Irvine Machine Learning Repository (Hettich, Blake, and Merz 1998): Mushroom, Nursery, and U.S. Census (ADULT).

8.1. Data Sets

Because our research represents one of the first attempts to handle many-valued nominal variables, most existing data sets have been manually constructed to include only a small

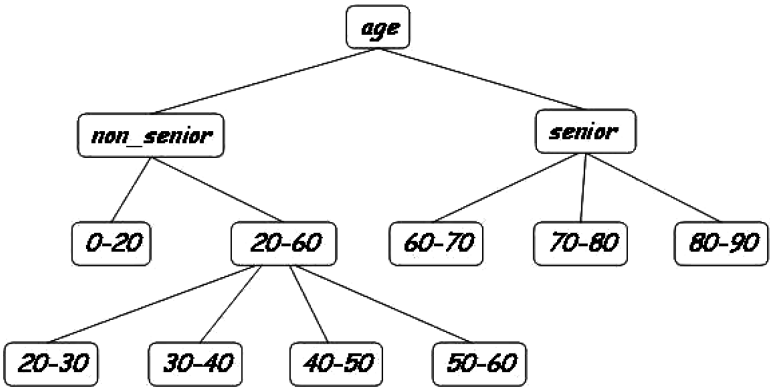


FIGURE 7. AVH for the *Age* variable of the Adult data set.

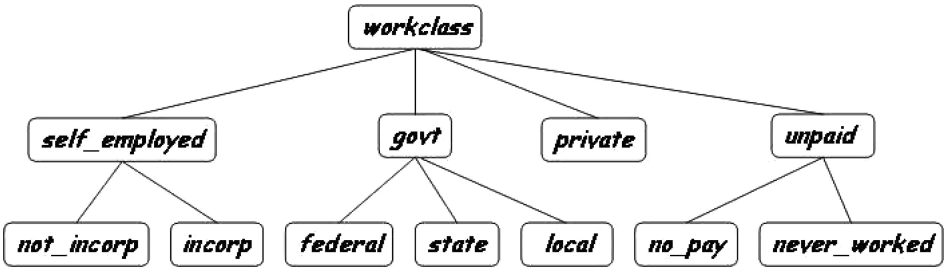


FIGURE 8. AVH for the *Work Class* variable of the Adult data set.

number of values for nominal variables. Such domains do not benefit significantly from the addition of AVHs, because the number of distinctions that can be made is already small. We use three data sets from the UC Irvine repository that have nominal variables with varying domain sizes.[7]

The Nursery data set (12,960 instances) has nine nominal variables, six of which have associated expert-provided AVHs. These AVHs are the shallowest of the data sets that we tested. Most of the AVHs have depth two, with a maximum branching factor of three.

The Mushroom data set (5644 instances) has 23 variables, 17 of which have associated expert-provided AVHs. This data set also has very shallow AVHs, but some variables have a higher branching factor (up to five).

The ADULT data set (45,222 instances), which is derived from U.S. Census data, has 13 variables (four continuous and nine nominal). We discretized the continuous variables, and created AVHs by hand for nine of the variables; these AVHs have depth 2 or 3, and a typical average branching factor of around 3. Figures 7 and 8 show the "expert" AVHs for the *Age* and *Work Class* variables, respectively.

We also generated data-derived AVHs using the clustering techniques described in Section 7. Because these AVHs are binary trees, they are generally much deeper than the expert-provided AVHs.

[7]Note that although these data sets are often used for supervised learning (classification), we are using them for density estimation, and therefore do not give our results in terms of precision (classification accuracy).

8.2. Experiments

We compared eight different learning algorithms:

- FLAT: Hill-climbing with "flat" CPTs—i.e., without abstraction or TCPTs
- ABS-E: ABS using expert-provided AVHs
- ABS-G: ABS using global clustering to learn the AVHs
- ABS-L1: ABS using "one-time" local clustering
- ABS-Lf: ABS using "frequent" local clustering
- TCPT: TCPT learning
- TABS-E: TABS with expert-provided AVHs
- TABS-G: TABS using global clustering to learn the AVHs
- TABS-L: TABS using ("one-time") local clustering to learn the AVHs

Fivefold cross-validation was used to estimate performance. For each fold, the data set was randomly divided into two parts: 80% training data and 20% test data. Each algorithm was then run five times on the training data, starting with a different random network. From these five, the network with the best MDL score on the training data was retained. The five networks thus obtained were evaluated on their respective test data. Because we are interested in measuring how well the learned BNs correspond to the joint probability distribution on the test data, we use log likelihood as our accuracy measure. Log likelihood will be maximized for test data that is highly probable, given the BN. The results reported below are the average of these five evaluations. In addition, we are interested in whether the learned BNs are qualitatively different in terms of complexity for the different algorithms, so we also measure the number of parameters and edges in the learned BNs.

8.3. Discussion

Table 1 shows the average log likelihood and standard deviations for all of the experiments. (Log likelihoods of smaller magnitude indicate a better fit to the data.) TCPT learning consistently yields better log likelihood than FLAT. In general, the use of expert AVHs does not significantly change the log likelihood scores in either direction, whether TCPTs are used or not. Data-derived AVHs do sometimes yield improvement for non-TCPT ABS learning. However, none of these variations reach even the performance of basic TCPT learning,

TABLE 1. Average Log Likelihood on the Three Data Sets, with Standard Deviations

	Log Likelihood Score					
	Nursery		Mushroom		Adult	
FLAT	−21769	±1.6	−8680.2	±5.6	−86740	±105.5
ABS-E	−21770	±3.7	−8683.4	±8.2	−86741	±138.9
ABS-G	−21762	±29.0	−8668.8	±8.0	−84058	±152.2
ABS-L1	−21846	±40.8	−8671.0	±28.2	−86770	±66.3
ABS-Lf	−21827	±24.6	−8644.2	±18.2	−86552	±139.3
TCPT	−21736	±1.6	−8594.5	±22.6	−86180	±21.9
TABS-E	−21735	±2.9	−8557.0	±15.6	−86165	±24.4
TABS-G	−21634	±24.1	−8610.4	±11.7	−83670	±105.2
TABS-L	**−19957**	±12.4	**−7839.3**	±26.4	**−80996**	±342.9

TABLE 2. Average Number of Parameters and Edges for Learned BNs

	Nursery		Mushroom		Adult	
	Params	Edges	Params	Edges	Params	Edges
FLAT	272.0	11.9	2280.2	43.3	2991.4	22.3
ABS-E	263.6	12.2	2114.8	45.1	2729.6	23.9
ABS-G	234.6	12.6	2446.2	44.9	2496.6	22.1
ABS-L1	251.2	20.1	2363.6	45.82	2884.8	25.0
ABS-Lf	280.7	21.7	2496.2	49.3	3045.3	29.3
TCPT	369.4	29.3	1838.2	75.3	2591.2	44.6
TABS-E	286.6	**32.0**	1304.2	89.2	3236.6	**52.8**
TABS-G	182.2	29.4	915.4	**106.4**	**1763.4**	49.9
TABS-L	**161.5**	26.4	**891.8**	103.6	1848.0	47.2

with one exception (Adult with ABS-G (global clustering)). There is no consistent pattern for which of the AVHs is the best: on Nursery and Adult, the global AVHs give the best performance; on Mushroom, the frequent local AVHs are best.

TCPT-E (TCPT learning with expert AVHs) gives mixed results: sometimes it performs slightly better than not using AVHs; sometimes it performs slightly worse.

TABS-L (TCPT learning with local clustering of AVHs) gives by far the best performance, across all three data sets. This is true even in the case of the Mushroom data set, which has fairly small domain sizes, and therefore yields relatively shallow AVHs. We conclude that learning local AVHs in conjuction with the use of TCPT learning is a highly effective learning method for categorical domains, in terms of accuracy.

Table 2 shows the average number of parameters and average number of edges in the learned BNs. For all variations and all data sets, the BNs learned by TCPT have more edges, on average, than those learned by FLAT. Similarly, ABS and TABS nearly always result in BNs with more edges than FLAT and TCPT, respectively (although in some cases, the increase is small). Given this observation, one would expect these more structurally complex BNs to require more parameters. However, in most cases, the resulting networks have *fewer* parameters. The most notable exception is ABS-Lf, which results in more edges, but uses significantly more parameters as well. Other exceptions include all of the ABS-based methods on the mushroom data set and TABS-E on the Adult domain, which all result in BNs with more edges but also somewhat more parameters.

The most important result is that TABS-L consistently and significantly decreases the number of parameters in all three domains, with an average compression ratio of 1.95 across the three domains. The BNs learned by TABS-L also have more edges than TCPT in two of the domains, and nearly as many edges as TCPT in the third (Nursery). Interestingly, in the Nursery domain, the TCPT BN has more parameters than FLAT (most likely because of the additional edges introduced by TCPT learning), so the TABS-L BN (with a compression ratio of 1.68 relative to FLAT, and 2.29 relative to TCPT) in some sense represents the best of both worlds.

Figure 9 and Table 3 show learning performance on different training set sizes in the Adult domain.[8] TABS-L consistently gives the best performance, down to 20% of the original data

[8]These experiments were run separately from those described earlier, with different random folds, so the results on the full data set are different.

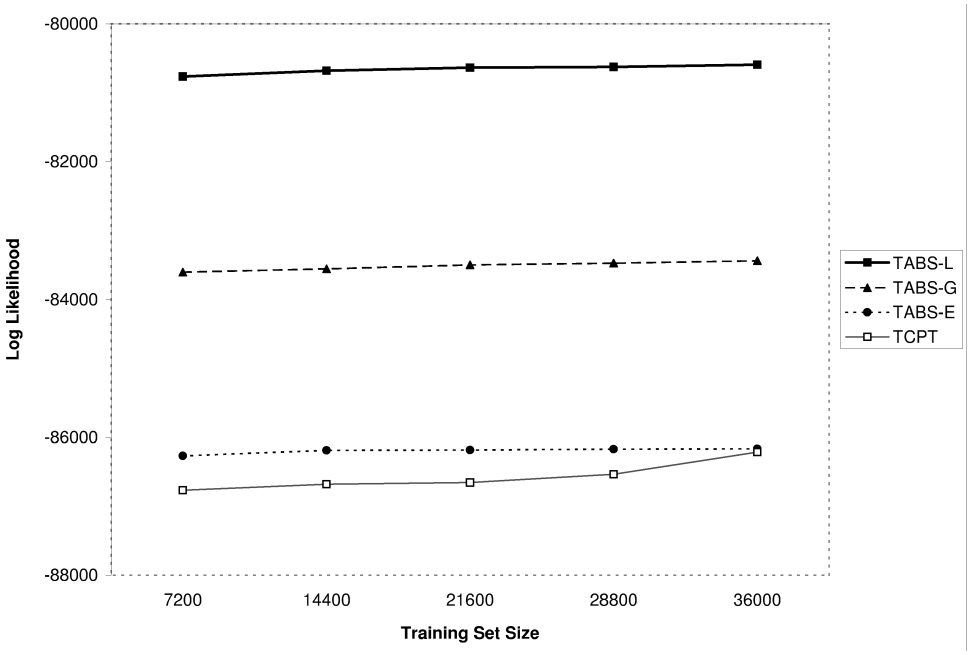


FIGURE 9. Log likelihood of learned BNs for Adult domain, for different training set sizes.

set (7200 instances, the smallest training set size that we tested), and TABS-G is consistently better than TABS. Interestingly, although TABS-E does not outperform TABS significantly on the full data set, it does yield statistically significantly better performance for small training set sizes. One can interpret this result as indicating that given enough data, expert hierarchies are not useful, but with less data, they provide valuable background information.

Figures 11 and 10 and Table 4 show the average number of parameters and edges for varying training set sizes. Not surprisingly, more data results in more edges and more parameters for any given algorithm. TABS-L consistently yields BNs with the fewest parameters of any algorithm. It always results in more edges than TCPT, and all three TABS algorithms yield a similar number of edges. However, TABS-E, which results in slightly more edges than the other two, also consistently uses many more parameters. TABS-G is nearly as good as TABS-L, but consistently uses more parameters (although not as many as TABS-E).

TABLE 3. Average Log Likelihood Scores on Adult for Different Training Set Sizes

	Log Likelihood Score			
Training Set	TCPT	TABS-E	TABS-G	TABS-L
7200	−86765.8	−86268.6	−83601.6	**−80765.8**
14400	−86679.6	−86187.8	−83555.6	**−80679.6**
21600	−86655.3	−86184.2	−83496.9	**−80636.2**
28800	−86536.2	−86170.7	−83474.2	**−80625.3**
36000	−86213.3	−86167.3	−83437.1	**−80593.3**

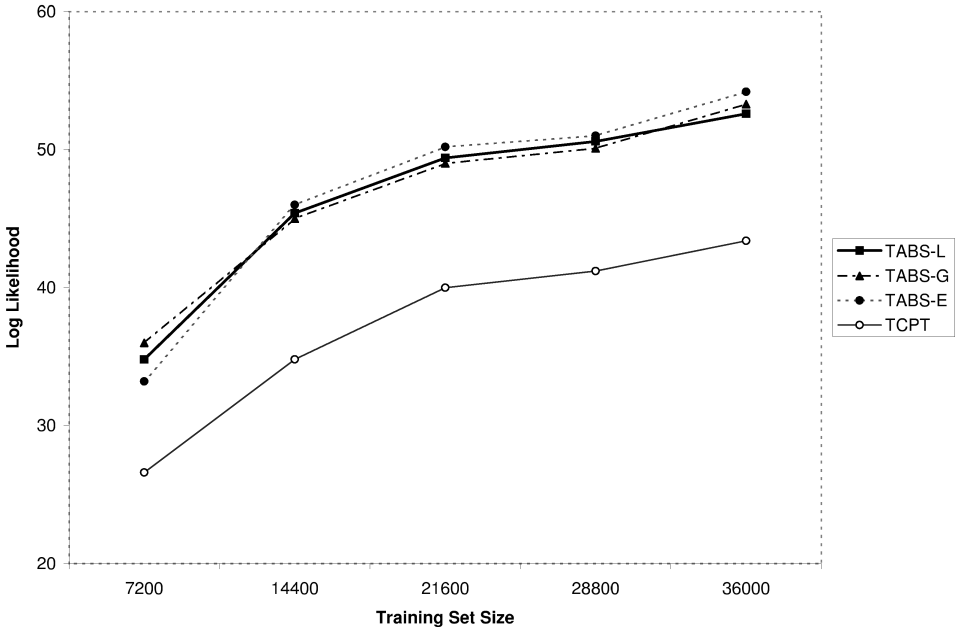


FIGURE 10. Number of edges in learned BNs for Adult domain, for different training set sizes.

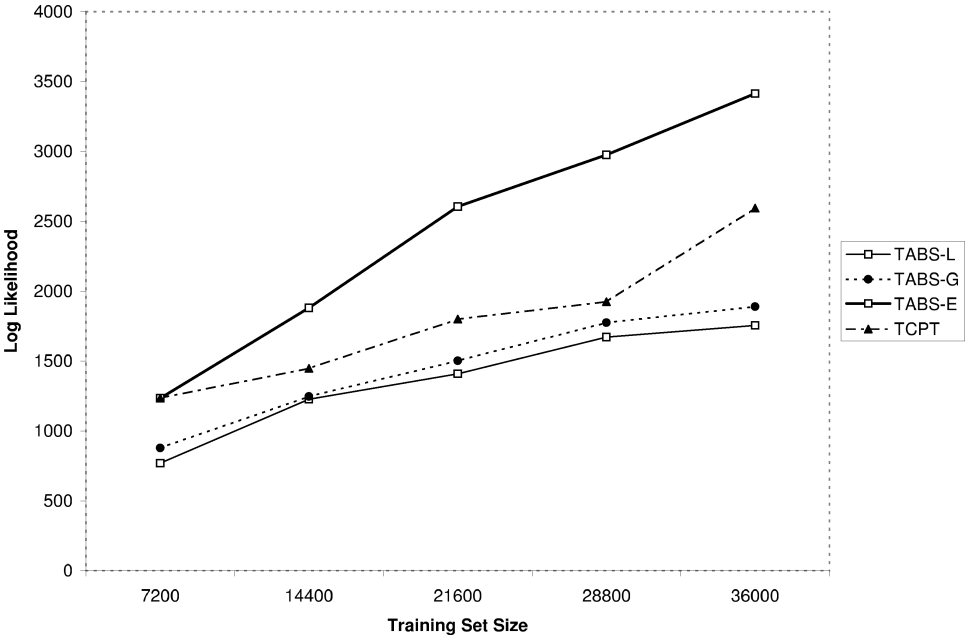


FIGURE 11. Number of parameters in learned BNs for Adult domain, for different training set sizes.

The variations in the performance of different AVHs warrants further investigation into the problem of how to determine the best AVH for a given learning problem. However, we can conclude that TABS with local clustering is consistently the best approach across all three metrics: log likelihood, number of edges, and number of parameters.

TABLE 4. Average Number of Parameters and Edges on Adult for Different Training Set Sizes

	TCPT		TABS-E		TABS-G		TABS-L	
	Params	Edges	Params	Edges	Params	Edges	Params	Edges
7200	1235.6	26.6	1234.6	33.2	878.2	**36.0**	**770.2**	34.8
14400	1448.0	34.8	1881.4	**46.0**	1246.8	45.0	**1227.6**	45.4
21600	1800.4	40.0	2606.2	**50.2**	1502.6	49.0	**1408.8**	49.4
28800	1924.2	41.2	2976.2	**51.0**	1775.2	50.1	**1672.2**	50.6
36000	2595.0	43.4	3414.8	**54.2**	1889.8	53.3	**1755.6**	52.6

9. RELATED WORK

Zhang and Honavar have presented methods for using AVHs to learn decision trees (Zhang and Honavar 2003) and naive Bayes models (Zhang and Honavar 2004). Their decision tree learning method has some similarities to our TCPT construction process, in that it maintains local contexts at each tree node, and always uses the "most abstract" split available at a given point in the tree. However, their scoring method is based on information gain rather than an MDL score, and is applied to classification problems rather than more general probability estimation. Zhang and Honavar allow the data to be represented with partially specified variable values—that is, a variable can take on any value in the AVH, not just leaf values. They impute leaf values probabilistically, based on global frequency counts. Our work could potentially be extended to permit partially specified values using a similar method. Alternatively, one might wish to use local frequency counts instead (i.e., impute values based on context-dependent counts), or to explicitly use "fractional instances" rather than imputing a single value to each partially specified variable.

Kang et al. (2004) give a method for generating AVHs using a hierarchical agglomerative clustering approach. However, because they are focused on pure classification tasks, the similarity measure for merging clusters is based only on the class distributions of the instances associated with a given group of values. (They use Jensen-Shannon divergence on these distributions to measure distance, although they point out that many other divergence measures are possible.) In contrast, we use a measure of distance in attribute space, making our similarity measure appropriate for non-classification (probability estimation) tasks.

Sharma and Poole (2003) have developed efficient methods for inference in BNs containing CPTs that are represented as decision trees or decision graphs, in which the decision (splitting) criteria can use intensional functions (procedural calls) to group values. Although we do not focus on inference in our work, this method could be applied to perform inference with our AVH-derived, extensional value groupings.

Previous methods for learning TCPTs typically allow each split in the tree to be either a *full split* (which includes a branch for each of the associated variable's values) or a *binary split* (which includes one branch for a selected value, and groups the remaining values into a second branch). The binary split is a type of naive abstraction—but this abstraction is purely local (i.e., it does not take into account expert-provided knowledge or global knowledge about the similarity of attribute values), and is costly in terms of the number of possible abstractions that must be tested. Although we have focused on TCPTs in this paper, other variations of context-specific independence representations, such as decision graphs (Chickering et al. 1997) and decision tables (Friedman and Goldszmidt 1996) could also benefit from AVHs. In effect, AVHs provide additional knowledge—either from an expert in the case of expert-provided AVHs, or from the entire data set in the case of data-derived data—that can be used to identify groups of values that are likely to behave similarly.

The agglomerative clustering for the construction of AVHs is a form of hidden variable introduction. While there has been some work on learning Bayesian networks with hidden variables (notably Friedman 1997), it is still an area that has many interesting open research problems. Some of the approaches to learning hidden variables involve a form of clustering; for example, module networks (Segal et al. 2005) cluster random variables into modules where all of the random variables in a module have the same CPTs. Burge and Lane (2006) describe an approach that uses *aggregation hierarchies* to structure the search space for learning a BN. In the latter work, the hierarchies are formed by grouping variables in the Bayes net together to form a hierarchical Bayesian network (Gyftodimos and Flach 2002). Hierarchical Bayesian networks allow one to group together probabilistically related *variables* of a single instance; this work is complementary to our work on grouping together *values* of a single variable that behave similarly across instances.

Boullé (2005) described a method for value grouping for categorical attributes in naive Bayes classification. This method selects a Bayes-optimal grouping by measuring the discriminatory value of sets of attribute values with respect to the class variable. Similarly, Kass (1980) uses a chi-square test to iteratively group values that are "similar" (with respect to the class variable) in decision tree learning. Neither of these approaches consider the possibility of changing the grouping dynamically during learning, and the resulting groupings are not hierarchical (although a binary AVH could be generated by recording the sequence of value groupings). The groupings are also specific to the class variable, so the method is not appropriate for more general density estimation problems. However, either of these grouping methods could potentially be adapted for use in the local clustering step of our algorithm, by applying them iteratively to construct a hierarchy for the local context.

10. CONCLUSIONS AND FUTURE WORK

We have presented ABS, which uses AVHs in learning BNs, and TABS, an extension to ABS that integrates AVHs with TCPTs. We also described two clustering-based algorithms for constructing AVHs from the training data used for learning a BN: a global method that uses all of the data, and a local method that uses only the local structure in the BN to learn each AVH. We showed that the use of AVHs significantly reduces the number of parameters required to represent the learned BN; in particular, TABS with locally clustered AVHs consistently yields BNs with the fewest parameters.

In future work, we plan to investigate variations to the tree-learning algorithm and to the clustering techniques for deriving AVHs, including non-binary agglomerative clustering. We also plan to extend our learning methods to permit partially specified (abstract) values in the data, and to support a decision graph representation of local structure.

ACKNOWLEDGMENTS

Thanks to Jun Zhang of Iowa State University for providing the AVHs for Nursery and Mushroom data sets. This work was partially funded by NSF #0325329 and NSF #0423845. Student support was also provided by the NSF ADVANCE Program at the University of Maryland, Baltimore County (UMBC), grant number SBE0244880.

REFERENCES

BOUCKAERT, R. R. 1994. Probabilistic Network Construction using the Minimum Description Length Principle. Technical Report RUU-CS-94-27, Utrecht University.

BOULLÉ, M. 2005. A Bayes optimal approach for partitioning the values of categorical attributes. Journal of Artificial Intelligence Research, **6**:1431–1452.

BOUTILIER, C., N. FRIEDMAN, M. GOLDSZMIDT, and D. KOLLER. 1996. Context-specific independence in Bayesian networks. *In* Proceedings of UAI-96, pp. 115–123.

BURGE, J., and T. LANE. 2006. Improving Bayesian network structure search with random variable aggregation hierarchies. *In* Proceedings of the European Conference on Machine Learning, Vol. LNAI 4212. *Edited by* J. Furnkranz, T. Scheffer, and M. Spiliopoulou. Springer-Verlag, Berlin, Heidelberg, pp. 66–77.

CHICKERING, D. M., D. GEIGER, and D. HECKERMAN. 1994. Learning Bayesian Networks is NP-Hard. Technical Report MSR-TR-94-17, Microsoft Research.

CHICKERING, D. M., D. HECKERMAN, and C. MEEK. 1997. A Bayesian approach to learning Bayesian networks with local structure. *In* Proceedings of UAI-97. Morgan Kaufmann Publishers, San Francisco, CA, pp. 80–89.

COOPER, G. F., and E. HERSKOVITS. 1992. A Bayesian method for the induction of probabilistic networks from data. Machine Learning, **9**(4):309–347.

DESJARDINS, M., L. GETOOR, and D. KOLLER. 2000. Using feature hierarchies in Bayesian network learning. *In* Proc. of SARA '02. Springer-Verlag, pp. 260–276.

DESJARDINS, M., L. GETOOR, and P. RATHOD. 2005. Bayesian network learning with abstraction hierarchies and context-specific independence. *In* Proceedings of the 16th European Conference on Machine Learning (ECML-2005).

FRIEDMAN, N. 1997. Learning belief networks in the presence of missing values and hidden variables. *In* International Conference on Machine Learning, pp. 125–133.

FRIEDMAN, N., and M. GOLDSZMIDT. 1996. Learning Bayesian networks with local structure. *In* Proceedings of UAI-96. Morgan Kaufmann Publishers, San Francisco, CA, pp. 252–262.

FRIEDMAN, N., and Z. YAKHINI. 1996. On the sample complexity of learning bayesian networks. *In* Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence.

GYFTODIMOS, E., and PETER A. FLACH. 2002. Hierarchical Bayesian networks: A probabilistic reasoning model for structured domains. *In* Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002).

HECKERMAN, D. 1995. A Tutorial on Learning Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research.

HETTICH, S., C. L. BLAKE, and C. J. MERZ. 1998. UCI machine learning repository.

JAIN, A. K., M. N. MURTY, and P. J. FLYNN. 1999. Data clustering: A review. ACM Computing Surveys, **31**(3):263–323.

KANG, D.-K., A. SILVESCU, J. ZHANG, and V. HONAVAR. 2004. Generation of attribute value taxonomies from data for data-driven construction of accurate and compact classifiers. *In* Proceedings of ICDM-04.

KASS, G. V. 1980. An exploratory technique for investigating large quantities of categorical data. Applied Statistics, **29**(2):119–127.

LAM, W., and F. BACCHUS. 1994. Learning Bayesian belief networks: An approach based on the MDL principle. Computational Intelligence, **10**(3):269–293.

PEARL, J. 1988. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco, CA.

POOLE, D., and N. L. ZHANG. 2003. Exploiting contextual independence in probabilistic inference. Journal of Artificial Intelligence Research, **18**:263–313.

SEGAL, E., DANA PE'ER, AVIV REGEV, DAPHNE KOLLER, and N. FRIEDMAN. 2005. Learning module networks. Journal of Machine Learning Research, **6**:556–588.

SHARMA, R., and D. POOLE. 2003. Efficient inference in large discrete domains. *In* Proceedings of UAI-03. Morgan Kaufmann, San Francisco, CA, pp. 535–542.

ZHANG, J., and V. HONAVAR. 2003. Learning decision tree classifiers from attribute value taxonomies and partially specified data. *In* Proceedings of ICML-03.

ZHANG, J., and V. HONAVAR. 2004. AVT-NBL: An algorithm for learning compact and accurate naive Bayes classifiers from attribute value taxonomies and data. *In* Proceedings of ICDM-04.