Bayesian Network Learning with Abstraction Hierarchies and Context-Specific Independence

Marie desJardins[1], Priyang Rathod[1], and Lise Getoor[2]

[1] Department of Computer Science and Electrical Engineering,
University of Maryland, Baltimore County
[2] Computer Science Department,
University of Maryland, College Park

Abstract. Context-specific independence representations, such as tree-structured conditional probability tables (TCPTs), reduce the number of parameters in Bayesian networks by capturing local independence relationships and improve the quality of learned Bayesian networks. We previously presented Abstraction-Based Search (ABS), a technique for using attribute value hierarchies during Bayesian network learning to remove unimportant distinctions within the CPTs. In this paper, we introduce TCPT ABS (TABS), which integrates ABS with TCPT learning. Since expert-provided hierarchies may not be available, we provide a clustering technique for deriving hierarchies from data. We present empirical results for three real-world domains, finding that (1) combining TCPTs and ABS provides a significant increase in the quality of learned Bayesian networks (2) combining TCPTs and ABS provides a dramatic reduction in the number of parameters in the learned networks, and (3) data-derived hierarchies perform as well or better than expert-provided hierarchies.

1 Introduction

Bayesian networks (BNs) are a widely used representation for capturing probabilistic relationships among variables in a domain of interest [12]. 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 inductively learned from data [5, 8].

The conditional probability distributions associated with discrete variables in a BN are most commonly represented as explicit conditional probability tables (CPTs), which specify a multinomial distribution over the values of a variable for each combination of values of its parents. However, researchers have found that explicitly representing *context-specific independence* (CSI) relationships in the BN can reduce the number of parameters required to describe the BN [2]. In particular, learning methods have been developed that use tree-structured CPTs [7] and graph-structured CPTs [4] to represent the CSI relationships among the variables.

In previous work, we presented Abstraction-Based Search (ABS) [6], which used background knowledge in the form of expert-provided attribute value hierarchies (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 and improving the quality of the learned BN.

The abstractions provided by AVHs in ABS are complementary to those provided by TCPTs. Therefore, in this paper, we provide a new version of abstraction-based search, TCPT ABS (TABS), which integrates ABS with TCPTs. Our empirical results show that TABS significantly 1) improves the quality of the learned BN and 2) reduces the number of parameters required to represent a learned BN, compared to standard BN learning, ABS, or TCPT learning alone.

A second contribution of this paper is an agglomerative clustering-based method for deriving AVHs from the training data used for learning the BN. Our original motivation for developing this technique 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 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.

The remainder of the paper is organized as follows. We first give background on BNs, TCPTs, and learning methods for BNs with TCPTs. Next, we introduce the ABS and TABS methods for incorporating AVHs into the learning process, and the clustering algorithm for deriving AVHs from training data. We then provide experimental results in three real-world domains, and finally present related work, conclusions, and future work.

2 Bayesian Networks

We assume that the reader has some familiarity with basic concepts of BNs [12] and local search-based methods for learning BNs [8]. In this section, we briefly introduce the notation, learning methods, and scoring functions that are used in the remainder of the paper.

A BN is a 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 these variables are all discrete and finite. The domain of variable x_i is given by the set $\{v_{i1}, v_{i2}, \ldots, v_{ir_i}\}$, where r_i is the number of values that x_i may take. A BN over the set of variables $\mathbf{X}$ is represented as a pair, $B = (G, \Theta)$. G, the BN structure, 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 independent of its non-descendents in the network, given its parents π_i. Using this conditional independence assumption, the joint probability distribution can be factored as:

$$P(\mathbf{X}) = \prod_i P(x_i|\pi_i). \tag{1}$$

Given a set of m instances, $D = \{d_1, d_2, \ldots, d_m\}$, where each d_j is an attribute vector $\langle d_{1j}, d_{2j}, \ldots, d_{nj}\rangle$, we would like to learn a BN that best matches the data. Since this problem is NP-hard [3], typically a simple greedy hill-climbing

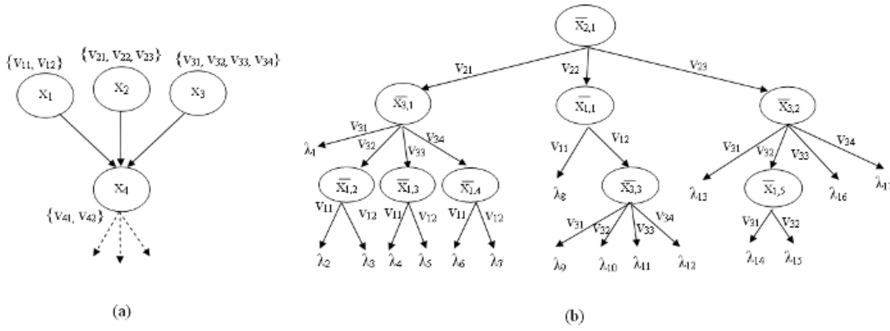

Fig. 1. (a) A partial view of the Bayesian network; (b) TCPT at variable x_4

search is used. The local search operators are $Add(x_i, x_j)$, which adds x_i as a parent of x_j; $Delete(x_i, x_j)$, which removes x_i from the parent set of x_j; and $Reverse(x_i, x_j)$, which reverses the edge between x_i and x_j. 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 scoring function can be decomposed locally, so that when an edge is added, modified or deleted, only the score of the variable x_i whose parent set π_i changed needs to be re-scored.

In our work, we use the Minimum Description Length (MDL) scoring criteria which attempts to select the hypothesis (i.e., the BN B) that minimizes the description length of the data D encoded using the BN. There are two components of the encoded data: the BN itself and the data encoding using the BN. The description length (DL) of the BN can be further decomposed into the DL of the structure, plus the DL of the (maximum likelihood) parameters. The DL of the data is given by its conditional entropy. The mathematical derivation of the MDL score is given by Bouckaert and others [1, 11].

3 Tree-Structured CPTs

In the TCPT representation, each variable in the BN has an associated tree-structured CPT. These trees specify the conditional probability of the values of a variable x_i, given its parents π_i. The leaves represent different conditional distributions over the values of x_i; the path from the root to a leaf defines the parent context for that distribution. In many cases, given a particular context of a subset of π_i, the value of x_i is independent of the rest of the parents in π_i. TCPTs can capture this local dependency structure.

We use the notation T_{x_i} to refer to the TCPT associated with the BN variable x_i. Each variable x_j that is a parent of x_i in the BN will have one or more corresponding nodes in T_{x_i}. We will refer to these tree nodes as $\bar{X}_j$ (to distinguish them from the variable x_j in the BN). Let $\#(\bar{X}_j)$ be the number of times that $\bar{X}_j$ appears in the tree; we will refer to these tree nodes as $\bar{X}_{j,p}$ $(p = 1, 2, \ldots, \#(\bar{X}_j))$. Each such tree node appears in a different *context* in the tree. The context of a

```
Procedure RefineTree($\lambda$, $\bar{X}_{new}$)
    1. remove $\lambda$ from $RefCand$
    2. replace $\lambda$ with $\bar{X}_{new}$, and create $|Val(\bar{X}_{new})|$ new leaf nodes below $\bar{X}_{new}$
    3. for each child $\lambda'$ of $\bar{X}_{new}$,
        a. set $Candidates(\lambda') = Candidates(\lambda)$ - $\{\bar{X}_{new}\}$
        b. add $\lambda'$ to $RefCand$
Procedure ExtendTree($x_i$)
    1. select a leaf node $\lambda$ for refinement from $RefCand$ with probability
            $prob \propto |Candidates(\lambda)|$
    2. $\bar{X}_{new} = argmin_t\{Score(T_{x_i} \cup \bar{X}_t)\}$
            where $\bar{X}_t$ is an instance of the $t^{th}$ candidate in $Candidates(\lambda)$
    3. RefineTree($\lambda, \bar{X}_{new}$)
```

Fig. 2. TCPT learning algorithm: The RefineTree and ExtendTree procedures

node $\bar{X}_{j,p}$ is defined by the branches (variable/value pairs) along the path from $\bar{X}_{j,p}$ to the root of the TCPT. We use $\Upsilon(\bar{X}_{j,p})$ to denote the context of $\bar{X}_{j,p}$.

For example, Figure 1(b) shows the TCPT at node x_4 of the BN shown in Figure 1(a). T_{x_4} includes three instances of $\bar{X}_3$: (1) $\bar{X}_{3,1}$, whose context is $\Upsilon(\bar{X}_{3,1}) = [(\bar{X}_{2,1} = v_{21})]$, (2) $\bar{X}_{3,2}$, whose context is $\Upsilon(\bar{X}_{3,2}) = [(\bar{X}_{2,1} = v_{23})]$, and (3) $\bar{X}_{3,3}$, with context, $\Upsilon(\bar{X}_{3,3}) = [(\bar{X}_{2,1} = v_{22}), (\bar{X}_{1,1} = v_{12})]$.

We use $\text{Val}(\bar{X}_{j,p}) = \{\bar{v}_{j,p1}, \bar{v}_{j,p2}, \ldots\}$ to denote the set of values $\bar{X}_j$ can take in the tree. In the standard TCPT representation, $|\text{Val}(\bar{X}_{j,p})| = r_j$, the domain size of x_j, and every non-leaf instance of $\bar{X}_j$ will have r_j outgoing edges. (When using AVHs, the branches can be associated with abstract values, so the branching factor will depend on the context of the node (Section 4.3).) The set of all $(\Upsilon(\lambda), \lambda)$ pairs associated with leaf nodes in T_{x_i} defines the CPT of x_i.

Learning TCPTs. Boutilier et al. [2] present a method for learning TCPTs by applying a recursive tree building algorithm each time a parent is added to a variable x_i in the BN. The TCPTs are generated using an MDL-based scoring function that trades off the complexity of the tree structure with the information gain that it provides. The tree building process is followed by a post-pruning step to remove unnecessary distinctions in the tree. Tree learning is a sub-step of the BN structure learning algorithm: each time a variable x_j is added to the parent set of another variable x_i, the TCPT for x_i is re-learned.

We propose an alternative TCPT learning approach, in which the BN structure learning process is redefined as the process of learning the TCPTs associated with individual variables. Instead of adding or removing edges from the network structure, we use adding and removing nodes in the individual TCPTs as the basic operations in the hill-climbing search. If the TCPT of x_i does not already contain an instance of x_j, then adding such an instance has the effect of adding an edge from x_j to x_i in the network. Similarly, removing the last occurrence of x_j from the TCPT of x_i is equivalent to removing the edge from x_j to x_i.

In our representation, each leaf node λ has an associated set of *Candidates*, which specifies the candidate variables that are available for further splitting the tree at that node. When refining the TCPT, the algorithm tries all possible variables in this set and then selects the refinement that offers the largest improvement in the MDL score.

Initially, the root starts with all $\bar{X}_j$s except itself in its *Candidates* set. As the TCPT is refined, the procedure RefineTree (Figure 2) propagates these candidates to the new leaf nodes after removing the candidate associated with the selected refinement. In Figure 1, the root $\bar{X}_{2,1}$, initially has two candidates for each of its outgoing edge: $Candidates = \{\bar{X}_1, \bar{X}_3\}$. When the candidate $\bar{X}_3$ is selected for refinement at edge v_{21}, the leaf is replaced with $\bar{X}_{3,1}$, and the candidate set of each of $\bar{X}_{3,1}$'s child leaf node is set to $\{\bar{X}_1\}$. On the other hand, when $\bar{X}_1$ is selected for refinement at edge v_{22}, the leaf node at that edge is replaced with $\bar{X}_{1,1}$, and each of its child leaf node's candidate set is set to $\{\bar{X}_3\}$.

The BN structure is learned by recursively splitting leaf nodes in the TCPTs. Each TCPT maintains a list $RefCand$, which lists the candidate nodes in the tree that can be further refined—i.e., leaf nodes whose *Candidates* sets are non-empty. Initially, this set contains only the root of the tree; it is updated as branches are added to the tree during the splitting process.

The optimal single-step refinement can be found by evaluating all possible refinements ($RefCand$s) for all variables in the BN, and choosing the one that most improves the MDL score. However, this is computationally intensive; therefore, we instead randomly choose the next BN variable to refine, and use the procedure ExtendTree (Figure 2) to select a leaf node to refine. The probability of a leaf node being selected for refinement is proportional to the size of the *Candidates* set at that node. The selected node is then refined using the candidate split that leads to the largest improvement in the MDL score of the tree. This process of selecting a variable and then refining its TCPT is performed until a local minimum is reached in the MDL score for the BN.

4 Learning BNs Using Abstraction Value Hierarchies

We next discuss AVHs in more detail, and then describe the ABS and TABS learning algorithms.

4.1 Attribute Value Hierarchies

An attribute value hierarchy (AVH) defines an IS-A hierarchy for a categorical feature value. The leaves of the AVH describe base-level values; these are the values that occur in the training set. The interior nodes describe abstractions of the base-level values. The intent is that the AVH is designed to define useful and meaningful abstractions in a particular domain.

Figure 3 shows an AVH for the Workclass attribute in a U.S. Census domain, which describes an individual's employer type. At the root, all workclass types are grouped together. Below this are three abstract workclass values—Self-employed, Government, and Unpaid—and one base-level value—Private. Each of the abstract values is further subdivided into the lower-level. As shown by

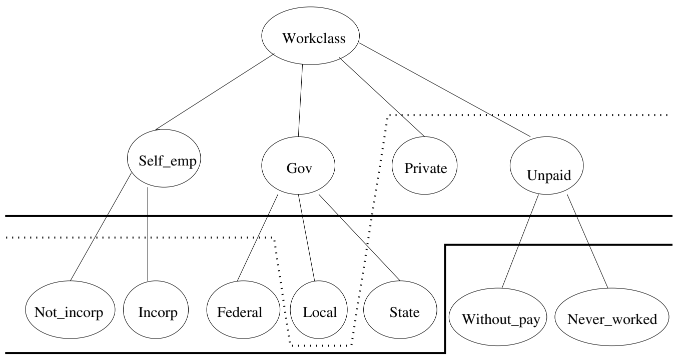

Fig. 3. AVH for the Workclass attribute, with legal (solid) and illegal (dotted) abstraction levels

this example, an AVH need not be balanced (i.e., path length from leaf values to the root can vary), and the branching factor (number of children) can vary within the hierarchy.

A cut through the tree defines an *abstraction level*, which is equivalent to a mutually exclusive and complete set of abstract attribute values. Figure 3 shows three different abstraction levels for the workclass attribute. Each abstraction level contains the set of values immediately above the cut line. The solid lines correspond to *legal* abstraction levels. The upper abstraction level includes the values Self_emp, Gov, Private, and Unpaid. The lower abstraction level includes the values Not_incorp, Incorp, Federal, Local, State, Private, and Unpaid. In this case, the lower abstraction level makes more distinctions than the upper abstraction level. The dotted line corresponds to an *illegal* abstraction level: for example, it includes both Gov and Local, which are not mutually exclusive.

The AVH helps to bias our search over appropriate abstractions for a categorical attribute. Without the hierarchy to guide us, we would need to consider arbitrary subsets of the base-level values for abstractions. Here, the AVH tells us which combinations of the values are meaningful (and, hopefully, useful in density estimation).

4.2 Abstraction-Based Search

There are two key tasks to be performed when learning a probabilistic model: scoring a candidate model and searching the space of possible models. We describe how these are done when CPTs associated with nodes can be represented at different abstraction levels.

The original ABS algorithm [6] extended the standard search over network structures as follows. When an edge is added to the network, the parent is added at its most abstract level (i.e., using the top-level values in the AVH). For example, if Workclass is chosen as a parent of another node, the initial abstraction level would be {Self_emp, Gov, Private, Unpaid}.

ABS extends the standard set of BN search operators—arc addition, arc deletion, and arc reversal—with two new operators: $\mathsf{Refine}(x_i, x_j, l)$ and $\mathsf{Abstract}$

$(\mathsf{x_i}, \mathsf{x_j}, \mathsf{l})$. The search process is a greedy search algorithm that repeatedly applies these five operators to the current network, evaluates the resulting network using the Bayesian score based on the MDL approach, and replaces the current network with the new one if the latter outscores the former.

If x_i is the parent of x_j, and its current abstraction level is $\{v'_{i1}, \ldots, v'_{ik'}\}$, $\mathsf{Refine}(\mathsf{x_i}, \mathsf{x_j}, \mathsf{l})$ refines the lth value of the abstraction, v'_l by replacing v'_l with the set of values of its children in the AVH. During the search process, ABS attempts to apply Refine to each value of each abstraction in the current network. Refine only succeeds if the value it is applied to is an abstract value (i.e., if the value has children in the AVH).

Similarly, if x_i is the parent of x_j, and its current abstraction level is $\{v'_{i1}, \ldots, v'_{ik'}\}$, $\mathsf{Abstract}(\mathsf{x_i}, \mathsf{x_j}, \mathsf{l})$ abstracts the lth value of the abstraction, v'_l by replacing v'_l and its siblings with the value of their parent in the AVH. Again, during search, ABS attempts to apply Abstract to each value of each abstraction level. Abstract only succeeds if the parent value is below the root node of the AVH and all of the value's siblings appear in the abstraction level. For example, in the lower abstraction level shown in Figure 3, neither condition is satisfied for the value Unpaid: its parent value is the root node of the hierarchy, and Unpaid's siblings Self_emp and Gov do not appear in the abstraction level.

4.3 The TCPT ABS 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, 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, we are able to reduce the number of parameters that are required to be learned for the BN.

Our TCPT ABS (TABS) learning algorithm (Figure 4) extends the TCPT refining algorithm RefineTree described in Section 3, with some provisions for adding nodes from AVHs into the TCPT.

```
Procedure RefineTreeAbs($\lambda$, $\bar{X}_{new}$)
    1. remove $\lambda$ from $RefCand$
    2. replace $\lambda$ with $\bar{X}_{new}$, and create $|Val(\bar{X}_{new})|$ new leaf nodes below $\bar{X}_{new}$
    3. for each child $\lambda'$ of $\bar{X}_{new}$,
          a. set $Candidates(\lambda') = Candidates(\lambda)$ - $\{\bar{X}_{new}\}$
          b. for each child value $v_{ij}$ of $\lambda'$ in the AVH of $\bar{X}_{new}$,
                (i). if $v_{ij}$ is an abstract value,
                        add a new candidate $V'_{ij}$ to $Candidates(\lambda')$
          c. add $\lambda'$ to $RefCand$
```

Fig. 4. TABS learning algorithm: The RefineTreeAbs procedure

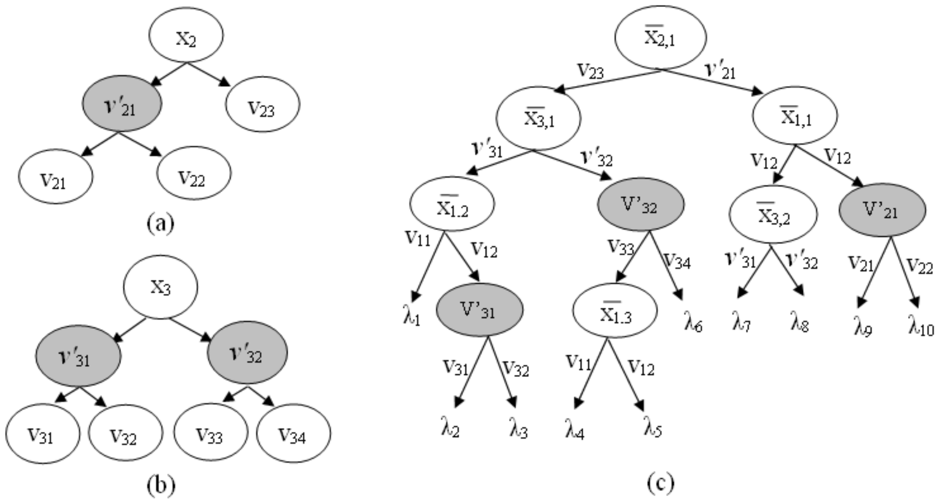

Fig. 5. (a) AVH of x_2; (b) AVH of x_3; (c) TCPT at variable x_4 learned with TABS-E

Suppose we have an AVH for x_i; the leaves of the AVH are the r_i base values of x_i, $\{v_{i1}, v_{i2}, \ldots, v_{ir_i}\}$. The internal, or abstract, values in the AVH are given by $\{v'_{i1}, v'_{i2}, \ldots\}$. Each abstract value corresponds to a set of base values. In TABS, when a new tree node is added to a TCPT, it is added at its most abstract level in the AVH. In other words, when a node $\bar{X}_{j,p}$ is added to a TCPT, its $Val(\bar{X}_{j,p})$ includes the set of values in the AVH of x_j that are the immediate children of the root node.

For example, suppose that the BN variables x_2 and x_3 in the BN of Figure 1(a) have associated AVHs, as shown in Figures 5(a) and (b). When these variables are used to split a TCPT node, the set of new branches will be $Val(\bar{X}_{2,1}) = \{v'_{21}, v_{23}\}$ and $Val(\bar{X}_{3,1}) = \{v'_{31}, v'_{32}\}$, as shown in Figure 5(c). Since the nodes associated with the abstract values can be further refined, we create candidate splits for the abstract values, V'_{jk}, and add these to $Candidates$ set of the leaf node associated with the abstract value v'_{jk}. This candidate refinement can be instantiated later through a refine step to split the tree. $Val(V'_{jk})$ is set to the set of values that are immediate descendants of v'_{jk} in the AVH of x_j.

In essence, this enables us to avoid making unnecessary distinctions among similar values until it is deemed necessary. For example, in Figure 5(c), when we add node $\bar{X}_{3,1}$, making a distinction between values v_{31} and v_{32} does not offer any gain in terms of the MDL score. However, once the tree is further split and node $\bar{X}_{1,2}$ is added, the distinction between v_{31} and v_{32} become more prominent and so we add the split V'_{31} in the next step.

5 Generating AVHs Using Agglomerative Clustering

It is not always possible to have access to sufficient domain knowledge or the services of a domain expert to create hierarchies of attribute values. Therefore,

```
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
```

Fig. 6. Agglomerative clustering algorithm for deriving AVHs from training data

we have developed an agglomerative clustering-based method for deriving AVHs from the training data used to build the BN.

Given a set of instances D, our goal is to find a hierarchical clustering of the values for each variable x_i. First, we put all of the instances d_j that have a particular value for x_i into a cluster, resulting in r_i clusters: $C = \{C_1, C_2, \ldots, C_{r_i}\}$, where $C_k = \{d_j | x_{ij} = v_{ik}\}$. These are the initial "leaf clusters."

The BuildAVH procedure shown in Figure 6 uses average-link agglomerative clustering to iteratively merge pairs of clusters. The distance $Dist(C_p, C_q)$ is defined to be the distance between the centroids of the clusters. (Note that variable x_i is ignored in computing the centroids and distances when deriving the AVH for x_i.) For nominal attributes, we use the most frequently occurring value (mode) of the attribute as the centroid. We use the Hamming distance for measuring the distance between two centroids. The above technique is repeated for each attribute x_i, yielding a binary AVH.

6 Experimental Results

In this section, we describe results of experiments on three real-world data sets from the UC Irving Machine Learning Repository [9]: Mushroom, Nursery, and U.S. Census (ADULT).

Data Sets. The nursery data set (12960 instances) has nine nominal variables, six of which have associated expert-provided AVHs. This data set is the shallowest data set that we tested. Most of the AVHs have depth one with a maximum branching factor of three, except for the class variable, which has five values. The Mushroom data set (5644 instances) has 23 variables, 17 of which have associated expert-provided AVHs. This data set also has variables with hierarchies that are very shallow, but some variables have a higher branching factor (up to five). The ADULT data set (45222 instances), which is derived from U.S. Census data, has 14 variables (five continuous and nine nominal). We discretized the continuous variables manually, and created AVHs by hand for nine of the variables.

We also generated data-derived AVHs using the clustering technique described in Section 5. Since these AVHs are binary trees, they are much deeper than the expert-provided AVHs.

Experiments. We compared six different learning algorithms: (1) FLAT: Hill-climbing with "flat" CPTs—i.e., without abstraction or TCPTs; (2) ABS-E:

Table 1. Average log likelihood on the three data sets

	Log Likelihood Score					
	Nursery		Mushroom		Adult	
FLAT	-21769	±1.57	-8680.2	±5.58	-86740	±105.48
ABS-E	-21770	±3.72	-8683.4	±8.18	-86741	±138.92
ABS-D	-21762	±29.02	-8668.8	±7.95	-84058	±152.18
TCPT	-21736	± 1.55	-8594.5	±22.55	-86180	±21.91
TABS-E	-21735	±2.92	**-8557.0**	±15.58	-86165	±24.40
TABS-D	**-21634**	±24.11	-8610.4	±11.72	**-83670**	±105.24

ABS using expert-provided AVHs; (3) ABS-D: ABS using data-derived AVHs; (4) TCPT: TCPT learning; (5) TABS-E: TABS with expert-provided AVHs; and (6) TABS-D: TABS with data-derived AVHs. Five-fold cross-validation was used to estimate performance. Each algorithm was run five times on each split and the network with the best MDL score from these five was retained. The results reported below are the average results over test sets for the different cross-validation splits.

Discussion. Table 1 shows the average log likelihood and standard deviations for the 18 experiments that we ran. TCPTs consistently improve the log likelihood of learned BNs, relative to FLAT. (Note that log likelihoods of smaller magnitude correspond to a better fit to the data.) The use of AVHs alone improves the score although this is not true in all cases. However, note that the combination of the two, the TABS algorithms, are always the best performers. In two cases, on Nursery and Adult, the TABS-D algorithm gives significant improvement, while for Mushroom, the TABS-E algorithm gives the best improvement. The results on the most complex data set, ADULT, are the most striking.

Table 2 shows the average number of parameters and average number of edges in the learned BNs. The BNs learned by TCPT have more edges on average than those learned by FLAT. Similarly, ABS/TABS result in more edges than FLAT/TCPT (although in some cases, the increase is small). One would expect these more structurally complex BNs to require a larger number of parameters. However, in most cases, the resulting networks have more edges and *fewer* parameters. In particular, TABS-D consistently yields an equal or greater number of edges as TCPT, while significantly decreasing the number of parameters in all three domains. (On average, TABS-D uses 55.6% fewer parameters than TCPT.)

Table 2. Average number of parameters and edges for learned BNs

	Nursery		Mushroom		Adult	
	Params	Edges	Params	Edges	Params	Edges
FLAT	272.0	11.91	2280.2	43.33	2991.4	22.26
ABS-E	263.6	12.16	2114.8	45.06	2729.6	23.88
ABS-D	234.6	12.63	2446.2	44.93	2496.6	22.13
TCPT	369.4	29.25	1838.2	75.33	2591.2	44.6
TABS-E	286.6	32	1304.2	89.17	3236.6	52.84
TABS-D	**182.2**	29.4	**915.4**	106.4	**1763.4**	49.9

7 Related Work

Zhang and Honavar have presented methods for using AVHs to learn decision trees [13] and Naïve Bayes models [14]. 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 density estimation. Zhang and Honavar allow the data to be represented with partially specified attribute values—that is, an attribute 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 attribute.

Kang et al. [10] give a method for generating AVTs using a hierarchical agglomerative clustering approach. However, since 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 (density estimation) tasks.

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 naïve 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 CSI representations, such as decision graphs [4] and decision tables [7] 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.

8 Conclusions and Future Work

We have presented TABS, an extension to Abstraction-Based Search that integrates attribute value hierarchies (AVHs) with tree-structured conditional probability tables (TCPTs). We also described a clustering-based algorithm for constructing AVHs from the training data used for learning a BN. We showed that the use of AVHs significantly improves the accuracy of the learned BN and reduces the number of parameters required to represent the learned BN. In particular, TABS with the data-derived AVHs consistently yields BNs with the smallest number of 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.

Acknowledgements

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 by UMBC's NSF ADVANCE grant. Lise Getoor was funded by NSF #0423845, with additional support from the National Geospatial Agency and the ITIC KDD program.

References

1. R. R. Bouckaert. Probabilistic network construction using the minimum description length principle. Technical Report RUU-CS-94-27, Utrecht University, 1994.
2. C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of UAI-96*, pages 115–123, 1996.
3. D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian Networks is NP-Hard. Technical Report MSR-TR-94-17, Microsoft Research, November 1994.
4. D. M. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of UAI-97*, pages 80–89, 1997.
5. G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.
6. M. desJardins, L. Getoor, and D. Koller. Using feature hierarchies in Bayesian network learning. In *Proceedings of SARA-02*, pages 260–270, 2000.
7. N. Friedman and M. Goldszmidt. Learning bayesian networks with local structure. In *Proceedings of UAI-96*, pages 252–262, 1996.
8. D. Heckerman. A Tutorial on Learning Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, March 1995.
9. S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.
10. D.K. Kang, A. Silvescu, J. Zhang, and V. Honavar. Generation of attribute value taxonomies from data for data-driven construction of accurate and compact classifiers. In *Proceedings of ICDM-04*, 2004.
11. W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, pages 269–293, 1994.
12. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
13. J. Zhang and V. Honavar. Learning decision tree classifiers from attribute value taxonomies and partially specified data. In *Proceedings of ICML-03*, 2003.
14. J. Zhang and V. Honavar. AVT-NBL: An algorithm for learning compact and accurate naive Bayes classifiers from attribute value taxonomies and data. In *Proceedings of ICDM-04*, 2004.