Learning Bayesian Networks From Dependency Networks: A Preliminary Study

Geoff Hulten
University of Washington
Seattle, WA

David Maxwell Chickering
Microsoft Research
Redmond, WA

David Heckerman
Microsoft Research
Redmond, WA

Abstract

In this paper we describe how to learn Bayesian networks from a summary of complete data in the form of a *dependency network* rather than from data directly. This method allows us to gain the advantages of both representations: scalable algorithms for learning dependency networks and convenient inference with Bayesian networks. Our approach is to use a dependency network as an "oracle" for the statistics needed to learn a Bayesian network. We show that the general problem is NP-hard and develop a greedy search algorithm. We conduct a preliminary experimental evaluation and find that the prediction accuracy of the Bayesian networks constructed from our algorithm almost equals that of Bayesian networks learned directly from the data.

1 Introduction

In this paper we describe how to learn Bayesian networks from a summary of data in the form of a *dependency network* rather than from data directly. The data is assumed to be complete—that is, contain no missing values. Dependency networks, described by Heckerman, Chickering, Meek, Rounthwaite and Kadie (2000), are graphical models that are similar to Bayesian networks. They differ in that their graphical structures are not required to be acyclic. Each node in a dependency network contains a conditional probability given its parents in the network, and a dependency network defines a joint probability distribution over the corresponding domain by means of Gibbs sampling (as described by Heckerman et al., 2000).

An advantage of dependency networks is that—using the approximate method described in Heckerman et al. (2000)—they are generally easier to learn from complete data than are Bayesian networks. Namely, we can learn the conditional probability distribution for each node in isolation, using any standard classification or regression algorithm (possibly in conjunction with explicit feature selection) to select parents for a node and populate the associated parameter values. Furthermore, there are many classification/regression learning algorithms that have been scaled up for large data sets. These algorithms can easily be applied to produce scalable algorithms for learning dependency networks with the associated local models. Notice that it is usually not obvious how these scalable algorithms can be modified to respect the graphical constraints imposed by Bayesian networks. There has been some work on scaling up learning Bayesian networks for the case when local models contain complete conditional probability tables. These include the Sparse Candidate Algorithm by Friedman, Nachman, and Peér (1999) and VFBN by Hulten and Domingos (2002). The scalability of these algorithms is hampered by the fact that the size of a conditional probability table grows exponentially with the number of parents of a node.

Bayesian networks have some advantages over dependency networks: the factored form of their joint distribution leads to efficient inference algorithms, whereas the Gibbs sampling that is often needed to extract probabilities of interest from a dependency network is slow. Sometimes, when the Bayesian network structures are reasonably simple, algorithms for exact inference can be applied. For more complex structures, there are many well-established approximate inference techniques such as loopy propagation (e.g. Murphy, Weiss, and Jordan, 1999) and variational methods (e.g. Jordan, Ghahramani, and Jaakola, 1999).

Learning Bayesian networks from dependency networks provides the advantages of both representations. We learn a dependency network from (complete) data using a well-known and scalable algorithm, and then construct a Bayesian network for more convenient inference. Once the dependency network is available, the computational complexity of the algorithm that produces the Bayesian network is independent of the size of the original data set.

There are a number of different approaches that can be used to construct a Bayesian network from a dependency network. The core idea they share is that the dependency network is used as an "oracle" for the sufficient statistics needed to learn a Bayesian network. When a needed statistic is explicitly encoded in the dependency network, it is simply returned. When a needed statistic is not explicitly encoded in the dependency network it can be generated via Gibbs sampling, approximated, or ignored. In this preliminary study, we describe an algorithm that considers only Bayesian network structures whose corresponding sufficient statistics are explicitly encoded in the conditional probability distributions of the dependency network. This approach produces Bayesian network structures that are acyclic sub-graphs of the dependency networks.

One drawback of any approach that constructs a Bayesian network from a dependency network is that any relation in the data that is not represented in the dependency network has little chance of being present in the Bayesian network learned from it. An alternate approach is to learn a model from sufficient statistics encoded in a Dynamic AD-tree (Komarek and Moore, 2000). This representation contains enough information to calculate exact values for all of the joint statistics, and it has been used to learn Bayesian networks. Nonetheless, AD-trees typically use substantially more memory than dependency networks, learning an AD-tree typically requires multiple scans of the data whereas learning a dependency network does not, and AD-trees are often forced to perform extra data scans at query time to save memory.

This paper is organized as follows. In Section 2, we describe our notation and present relevant background material. In Section 3, we argue that our learning problem is NP-hard, present the DN2BN algorithm, and provide a detailed implementation for DN2BN for the special case when all conditional probability distributions are decision trees. In Section 4, we provide experimental results showing that the Bayesian networks learned using DN2BN are competitive (in terms of prediction accuracy) with Bayesian networks learned directly from data. Finally, in Section 5, we conclude with a summary and discussion of future work.

2 Background

Throughout the paper, we use the following syntactical conventions. We denote a variable by an upper-case token (e.g. A, B_i, Y, Θ) and a state or value of that variable by the same token in lower case (e.g. a, b_i, y, θ). We denote sets with bold-face capitalized tokens (e.g. $\mathbf{A}, \mathbf{Pa_i}$) and corresponding sets of values by bold-face lower case tokens (e.g. $\mathbf{a}, \mathbf{pa_i}$). Finally, we use calligraphic tokens (e.g. $\mathcal{G}, \mathcal{B}$) to denote statistical models and graphs.

Consider a domain of n variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Dependency networks and Bayesian networks both (1) encode assertions about the dependencies and independencies that hold among the variables in $\mathbf{X}$ and (2) contain local probability distributions that characterize a joint probability distribution over $\mathbf{X}$. More specifically, both are directed graphical models $(\mathcal{S}, \mathbf{\Theta})$ where $\mathcal{S}$ is the structure of the model and $\mathbf{\Theta}$ its corresponding set of parameters. The structure $\mathcal{S}$ contains both (1) a directed graph, whose nodes are in one-to-one correspondence with the variables in $\mathbf{X}$, and whose (lack of) edges represent the *global* (independence) constraints among those variables and (2) any *local* constraints that exist in the conditional distributions corresponding to each variable (e.g. the constraints imposed by the structure of a decision-tree). The parameters $\mathbf{\Theta}$, combined with the global and local constraints, define a joint probability distribution $p(\mathbf{X})$. We use X_i to denote both the variable and the corresponding node in a graphical model. We use $\mathbf{Pa_i}$ to denote the parents of X_i in a graphical model.

A Bayesian network $\mathcal{B} = (\mathcal{S}_\mathcal{B}, \mathbf{\Theta}_\mathcal{B})$ is a directed graphical model for which the associated directed graph is acyclic. The model encodes the conditional independence constraints that each node is independent of its non-descendants given its parents. The result of these constraints is that the joint distribution over $\mathbf{X}$ can be factored as follows:

$$p(x_1, ..., x_n|\mathcal{S}_\mathcal{B}) = \prod_{i=1}^{n} p(x_i|\mathbf{pa_i}, \mathbf{\Theta}_\mathcal{B}) \qquad (1)$$

where $\mathbf{pa_i}$ is the set of values for $\mathbf{Pa_i}$ within $x_1, ..., x_n$.

A dependency network $\mathcal{D} = (\mathcal{S}_\mathcal{D}, \mathbf{\Theta}_\mathcal{D})$ is similar to a Bayesian network, except that the associated directed graph is not necessarily acyclic. The model encodes the conditional independence constraints that each node is independent of *all other nodes* in $\mathbf{X}$ given its parents. The model stores, for each node X_i, the probability distribution

$$p(X_i|\mathbf{Pa_i}, \mathbf{\Theta}_\mathcal{B}) = p(X_i|\mathbf{X} \setminus X_i, \mathbf{\Theta}_\mathcal{B}) \qquad (2)$$

As shown by Heckerman et al. (2000), the set of all conditional probability distributions defined by Equation 2 collectively define a joint probability distribution $p(\mathbf{X}|\mathcal{S}_\mathcal{D}, \mathbf{\Theta}_\mathcal{D})$ by means of Gibbs sampling.

There are many well-known algorithms for learning Bayesian networks from data. Buntine (1996) provides a good review of the literature, Heckerman (1995) presents a tutorial on the topic, and Jordan (1998) contains some introductory articles and more recent advances. In one popular class of algorithms, the so called *search-and-score algorithms*, a search algorithm is used in conjunction with a scoring criterion to evaluate the fit of candidate models to the data. Once a good structure is identified, the corresponding parameters are estimated in a straightforward manner. The methods described in this paper are related to this class of algorithm.

The scoring criterion and parameter estimates of search-and-score algorithms are based on sufficient statistics of the data. The *sufficient statistics of the data for a given model* are a summary of the data that is sufficient to both compute the score and estimate the parameters of a model. For example, if all variables in $\mathbf{X}$ are discrete and the graph is empty, then the sufficient statistics for the model are the marginal counts of each variable of X_i.

An assumption made by many Bayesian network learning algorithms is that the parameters associated with each variable are mutually independent. Given this assumption and a complete data set, the parameters remain mutually independent a posteriori, and the structure score can be written as the sum of independent sub-scores, one for each conditional distribution in the model. In addition, the sufficient statistics can be decomposed into sufficient statistics for individual conditional probability distributions. For example, when learning a Bayesian network for discrete variables, the counts needed to evaluate the entire network can be written as the union of the counts needed to estimate each conditional distribution. In general, when these properties hold, the scoring criterion is said to be *decomposable.*

Heckerman et al. (2000) describe an approach for learning dependency networks from complete data. The basic idea behind their approach is to learn each conditional distribution associated with the dependency network separately. That is, $p(X_i|\mathbf{X} \setminus X_i)$ for each X_i is learned independently using some probabilistic classification/regression model. An important issue with this approach is the consistency of the learned distributions. The set of conditional distributions associated with a dependency network is said to be *consistent* if there exists a joint distribution for $\mathbf{X}$ from which each conditional can be obtained via the rules of probability. It is not difficult to see that the independent learning of conditional distributions may lead to inconsistencies. For example, due to small-data effects, a learned decision tree that predicts Y may not split on X, whereas a learned decision tree for X may split on Y. Nonetheless, Heckerman et al. (2000) argue that, for reasonably large data sets, the local distributions will be "almost consistent" (described formally in their paper), because each is learned from the same set of joint data.

This heuristic method for learning dependency networks is the key to our approach for constructing a Bayesian network from a dependency network. Because each conditional distribution in the dependency network is learned separately, we can associate with each distribution a set of sufficient statistics that may then be applied to the construction of the Bayesian network using a decomposable scoring criterion. Note that the space needed to store sufficient statistics for each conditional distribution is typically on the same order as the space needed to store the parameter values themselves.

In the next section, we will present the details of a learning algorithm under the assumption that the conditional distributions in both dependency networks and Bayesian networks are represented with decision trees. A decision tree $\mathcal{T}_i$ is a tree-structured local model that represents the conditional probability for a singleton target variable X_i given its parents $\mathbf{Pa_i}$. Each internal node in the tree contains a test on one of parent variables and has one child for each possible outcome of the test. Corresponding to each leaf is a probability distribution over X_i that is specified by the parameters $\mathbf{\Theta}$ of the Bayesian network or dependency network. Every path from the root of a decision tree to a leaf passes through some number of internal nodes. Combining these node tests selects a subspace of the space defined by the values of $\mathbf{Pa_i}$, and each leaf models the distribution for X_i in one of these subspaces. Notice that these subspaces are non-overlapping and that their union is the space spanned by the values of $\mathbf{Pa_i}$.

Throughout our remaining discussion, we assume that the parameters associated with each leaf of a decision tree are mutually independent. This assumption will typically be incoherent for dependency networks, but will produce additional decomposition of the model scores.

3 Learning Bayesian Networks From Dependency Networks

There are many ways to use the information in a dependency network to learn a Bayesian network. In this section we will restrict ourselves to ones that consider only Bayesian networks whose sufficient statistics are explicitly represented in the dependency network. This restriction limits the Bayesian network structures that we can score, and thus limits the search space used for learning. The set of Bayesian networks that can be scored can be determined by examining the conditional probability distributions used by the dependency network. For example, when all variables in the domain are discrete and the conditional distributions are unconstrained multinomials we can evaluate any Bayesian network that is an acyclic sub-graph of the dependency network. (The sufficient statistics for $p(X_i|\mathbf{Pa_i})$ can be derived easily from the sufficient statistics for $p(X_i|\mathbf{Pa'_i})$ if $\mathbf{Pa_i} \subseteq \mathbf{Pa'_i}$). When the conditional distribution has local structure (e.g., a decision tree), additional restrictions apply.

The resulting set of scored Bayesian networks can be explored in many ways. In Section 3.1, we argue that finding the optimal one is NP-hard, and consequently it is appropriate to apply heuristic search techniques. Two obvious search strategies are to start from an empty Bayesian network and add edges from the dependency network until further additions create cycles; and to start from the dependency network and remove edges until there are no cycles. In this paper, as described in Section 3.2, we take the latter approach.

3.1 Complexity Result

In this section, we provide a simple reduction from a known NP-hard problem to the decision problem corresponding to learning a Bayesian network from a dependency network. Following is the decision problem corresponding to our learning task:

DNET-TO-BNET
INSTANCE: Dependency network $\mathcal{D} = (\mathcal{S}_\mathcal{D}, \Theta_\mathcal{D})$ and scoring criterion $S(\mathcal{S}_\mathcal{B}, \mathcal{D})$ that evaluates Bayesian-network structures.
QUESTION: Does there exist a Bayesian network structure $\mathcal{S}_\mathcal{B}$ such that $S(\mathcal{S}_\mathcal{B}, \mathcal{D}) \geq s$?

The reduction is from the following decision problem that was proved to be NP-hard by Karp (1972):

FEEDBACK ARC SET
INSTANCE: Directed graph $\mathcal{G} = (\mathbf{V}, \mathbf{A})$, positive integer $k \leq |\mathbf{A}|$.
QUESTION: Does there exist a subset $\mathbf{A}' \subseteq \mathbf{A}$ with $|\mathbf{A}'| \leq k$ such that $\mathbf{A}'$ contains at least one arc from every directed cycle in $\mathcal{G}$?

Lemma 1 *DNET-TO-BNET is NP-Hard.*

Proof: We prove the result by providing a polynomial reduction from FEEDBACK ARC SET. The dependency network $\mathcal{D}$ is defined such that (1) the directed graph is precisely $\mathcal{G}$ in the instance of FEEDBACK ARC SET, and (2) the set of parameters and sufficient statistics are empty. The scoring criterion for the instance of DNET-TO-BNET returns zero if $\mathcal{S}_\mathcal{B}$ contains any edges not in $\mathcal{G}$, and otherwise simply returns the number of edges in $\mathcal{S}_\mathcal{B}$. s is set to the number of edges in $\mathcal{G}$ minus the value k from the instance of FEEDBACK ARC SET.

Clearly the reduction is polynomial. We now show that there exists a Bayesian network structure with score $\geq s$ if and only if there is arc set of size $\leq k$. Given a valid solution $\mathbf{A}'$ from FEEDBACK ARC SET, we create a (necessarily acyclic) Bayesian network structure $\mathcal{S}$ by removing all of these edges from $\mathcal{G}$. Clearly $\mathcal{S}$ is a subgraph of $\mathcal{G}$ with precisely s edges, resulting in a score equal to s. Given a Bayesian network structure $\mathcal{S}$ with score $\geq s$, we define $\mathbf{A}'$ to be the set of arcs in the dependency-network graph that are not in $\mathcal{S}$. By definition of the scoring function, we know that there are at most k edges in $\mathbf{A}'$; furthermore, because $\mathcal{S}$ is acyclic, every directed cycle in the dependency-network graph $\mathcal{G}$ must contain at least one edge in $\mathbf{A}'$. $\Box$

For the proof of Lemma 1, we took advantage of the somewhat arbitrary definition of the scoring criterion S. In practice, the values that most real-world criteria assign to network structures are going to be constrained by the sufficient statistics in the dependency network, which in turn are determined by some data set. Although we have no proof, we conjecture that when DNET-TO-BNET is restricted to such real-word criteria, the problem remains hard.

3.2 The DN2BN Algorithm

Given the result from the previous section, it is appropriate to apply a heuristic search algorithm to identify a high-scoring Bayesian network structure from the dependency network. In this section, we describe a greedy implementation of the learning algorithm for the special case when the conditional distributions are decision trees.

Although there are numerous greedy approaches that could be applied to our problem, in this preliminary study we consider a simple algorithm that repeatedly removes edges from the dependency network (along with the associated splits from the decision trees) until the resulting graph is acyclic. That is, our approach simplifies the dependency network until the *first* valid Bayesian network structure is encountered, and that first structure is returned by the algorithm.

We call our algorithm *DN2BN*. DN2BN takes as input a dependency network $\mathcal{D}$ and outputs a Bayesian net-

Table 1: Pseudo-code for DN2BN.

```
Input: A dependency network $\mathcal{D} = (\mathcal{S}_\mathcal{D}, \Theta_\mathcal{D})$
Output: A Bayesian-network structure $\mathcal{S}_\mathcal{B}$
Let $\mathcal{S}_\mathcal{B} = \mathcal{S}_\mathcal{D}$
(Let $\mathcal{T}_i$ denote the decision tree corresponding to
        node $X_i$ in $\mathcal{S}_\mathcal{B}$)
While there are cycles in $\mathcal{S}_\mathcal{B}$
    Let $\mathbf{E}_{cycle}$ be the set of edges in cycles in $\mathcal{S}_\mathcal{B}$
    Find the cost of removing each edge in $\mathbf{E}_{cycle}$
    Let $X_i \rightarrow X_j$ have lowest removal cost in $\mathbf{E}_{cycle}$
    Remove $X_i \rightarrow X_j$ from $\mathcal{S}_\mathcal{B}$
        Update $\mathcal{T}_j$ by pruning splits on $X_i$
        Let $\mathbf{Par}'(\mathbf{j})$ be the set of parents of $X_j$
            that do not have splits in the new $\mathcal{T}_j$
        For every $X_k \in \mathbf{Par}'(\mathbf{j})$
            remove $X_k \rightarrow X_j$ from $\mathcal{S}_\mathcal{B}$
Return $\mathcal{S}_\mathcal{B}$
```

work. In order to remove the edge $X_i \rightarrow X_j$, the decision tree $\mathcal{T}_j$ must be modified so that none of its internal nodes test X_i. This modification will presumably reduce the quality of $\mathcal{T}_j$—measured by the component of the decomposable scoring function local to X_j—by forcing it to take advantage of less information. The goal of DN2BN is to remove the minimum-cost set of edges from $\mathcal{S}_\mathcal{D}$ so that it no longer contains cycles.

Table 1 contains pseudo-code for DN2BN. Note that the structures of the input dependency network and the output Bayesian network specify the graphical structure and the structure of the decision trees (but not their parameters). The algorithm identifies all edges that are involved in cycles in $\mathcal{S}_\mathcal{B}$. Removing edges that are not involved in cycles makes no sense: there is a cost as it reduces the quality of some node's decision tree, but there is no benefit as it does not make progress towards removing cycles from $\mathcal{S}_\mathcal{B}$. Therefore DN2BN does not consider removing non-cycle edges.

The cost of removing each cycle edge is calculated in the following manner. When $X_i \rightarrow X_j$ is removed from $\mathcal{S}_\mathcal{B}$, $\mathcal{T}_j$ must be modified so that none of its internal nodes test the value of X_i. DN2BN accomplishes this by identifying every subtree of $\mathcal{T}_j$ that is rooted by a node splitting on X_i, and replacing these subtrees with leaves. The sufficient statistics for the new leaves are computed by combining the sufficient statistics corresponding to the leaves of their deleted subtrees. Given the decomposable scoring criterion used in Heckerman et al.'s (2000) heuristic learning method, we can evaluate each edge deletion by taking the difference in local score between the new and old trees.

After scoring all of the edges, DN2BN selects the edge with lowest cost, removes it from $\mathcal{S}_\mathcal{B}$, and updates the appropriate decision tree. Notice that the new tree may be substantially smaller than the old one. In the worst case, when the edge $X_i \rightarrow X_j$ is removed and the root node of the original $\mathcal{T}_j$ tests X_i, the resulting tree will be a single leaf. In this and many other cases, some variables from $\mathbf{Pa_j}$—other than the explicitly deleted X_i—may no longer participate in splits in $\mathcal{T}_j$. The edges associated with these *pruned parents* are also removed from $\mathcal{S}_\mathcal{B}$ by the algorithm.

A simple extension to this basic algorithm takes advantage of an easily computed property: when X_i is in $\mathbf{Pa_j}$ and X_j is in $\mathbf{Pa_i}$, these two nodes form a cycle of length two and either $X_i \rightarrow X_j$ or $X_j \rightarrow X_i$ must be removed in order to make $\mathcal{S}_\mathcal{B}$ acyclic. DN2BN first greedily breaks all of these length-two cycles before considering any other edge removals. This modification has the potential to simplify the problem and avoid removing some spurious edges, because removing a single edge from $\mathcal{S}_\mathcal{B}$ potentially breaks many cycles. In fact, in our experiments, we found that breaking these trivial cycles (and removing pruned parents) sometimes removed all cycles from the dependency network.

DN2BN repeats these steps until $\mathcal{S}_\mathcal{B}$ is acyclic and then terminates. Notice that DN2BN removes one edge in each iteration and will require $O(c)$ removals to break all cycles, where c is the number of edges involved in cycles in the input graph. We will now examine the complexity of the operations required in each of these iterations. The first operation in each iteration finds the set of edges involved in cycles in $\mathcal{S}_\mathcal{B}$. This operation can be done by finding the strongly connected components (SCCs) of the graph. It is well known (see Cormen, Leiserson, and Rivest, 1990) that this can be accomplished in $O(n + e)$ time. Notice, however, that this can be accelerated in iterations after the first by maintaining an auxiliary graph containing just the cycle-edges and information about each node's SCC membership. Then, when an edge is removed, the SCC algorithm need only be run on the SCC that contained the removed edge. Thus, maintaining E_{cycle} takes $O(n_{max} + c_{max})$ time, where n_{max} and c_{max} are the number of nodes and cycle edges (respectively) in the largest SCC. The second operation in each iteration scores the cost of removing each cycle edge. Scoring an edge removal requires time proportional to the number of leaves in the subtrees it prunes. This scoring can be accelerated using the decomposability of our scoring metric. In particular, only the scores of edges pointing to X_j must be updated after removing edge $X_i \rightarrow X_j$. The number of such affected edges is bounded by the number of nodes in X_j's SCC. Therefore, the time for this step is bounded by $O(n_{max} * l_{max})$, where l_{max} is the number of leaves in the largest decision tree. These two operations dominate the time of the remainder of DN2BN's operations; and the total time complexity is bounded by $O(c * (c_{max} + n_{max} * l_{max}))$. Notice that this is independent of the size of available training data. Existing scalable decision tree induction algorithms, such at VFDT by Domingos and Hulten (2000), can be used to learn dependency networks in time that is independent of the size of available training data. Thus, Bayesian networks with decision trees for conditional distributions can be learned in a scalable manner.

4 Experimental Evaluation

In this section we compare the predictive accuracy and learning times of Bayesian networks built with DN2BN with those learned directly from data. The following describes the four (real) data sets used in our comparison. Table 2 provides additional properties of these data sets.

1. **Media Metrix**

   The Media Metrix data set contains demographic and internet-use data for individuals during the month of January 1997.

2. **Nielsen**

   The Nielsen data set contains data about television-watching behavior during a two-week period in 1995. The data was made available courtesy of Nielsen Media Research. The data records whether or not each user watched five or more minutes of network TV shows aired during the given time period.

3. **EachMovie**

   The EachMovie data set consists of viewer ratings on movies and was collected during an 18-month period beginning in 1995. The rating is a discrete variable that is either missing, or is provided as an integer from one to five.

4. **Census**

   The Census data set, extracted from the United States Census Bureau, contains demographic information about United States citizens.

Note that the EachMovie and Census data sets contain missing entries. The data was completed by making "missing" an explicit state for each variable. In addition, all of the data sets except for EachMovie contains both discrete and continuous variables.

Table 2: Number of variables (n) and number of samples for the evaluation data sets.

Data Set	n	# Samples
Media Metrix	37	4808
Nielsen	407	3550
EachMovie	1625	72916
Census	37	299285

Table 3: The log-score of the produced model on the test set. Empty is the score of the marginal model.

Data Set	BNET	DN2BN	Empty
Media Metrix	-0.7620	-0.7934	-1.1894
Nielsen	-0.1038	-0.1055	-0.1265
EachMovie	-0.0862	-0.0906	-0.1216
Census	-1.3355	-1.4471	-1.8633

30% of each data set was randomly reserved for testing. We learned a dependency network and a Bayesian network from the remaining data—where the learned conditional distributions in both models were decision trees—using the algorithms described in Heckerman et al. (2000). In particular, we used a Bayesian scoring criterion in conjunction with a forward greedy search. The leaves of the decision trees contained either multinomial distributions (for discrete target variables) or univariate-Gaussian distributions (for continuous targets). We refer to the dependency-network algorithm as *DNET* and to the Bayesian network algorithm as *BNET* below. We then converted the dependency network into a Bayesian network using DN2BN. The quality of the resulting models were measured on the reserved test data by calculating the average over cases of

$$\frac{\log p(\text{test case})}{n}$$

(recall that n is the number of variables in the domain).

The same Bayesian criterion was used for DNET, BNET, and DN2BN. We used a structure prior of 0.001^f, where f is the number of free parameters in the structure. This prior favors simpler models, and the value 0.001 has proven effective over a wide range of situations. We used a diffuse parameter prior.

Table 4: Number of edges in the graphical structure.

Data Set	BNET	DNET	DN2BN
Media Metrix	85	124	61
Nielsen	626	784	466
EachMovie	6959	7437	5367
Census	306	402	77

Table 5: Details collected from the run of DN2BN. *l2 edges remvd* is the number of edges removed to break length two cycles. *other edges remvd* is the number of edges removed after breaking length two cycles. *parents pruned* is the number of edges removed because some other removal pruned their effect from a decision tree. *trees pruned* is the number of decision trees that were pruned down to a single leaf. *leaves remvd* is the difference in the number of leaves between input dependency network and output Bayesian network.

Data Set	l2 edges remvd	other edges remvd	pruned parents	trees pruned	leaves remvd
Media Metrix	43	7	14	7	131
Nielsen	260	0	58	95	355
EachMovie	1345	9	716	102	3261
Census	325	14	104	156	1806

Table 6: Run times of the algorithms in minutes. DNET+DN2BN is the sum of the time needed by DNET and DN2BN.

Data Set	BNET	DNET	DN2BN	DNET+DN2BN
Media Metrix	< 1	< 1	< 1	< 1
Nielsen	< 1	< 1	< 1	< 1
EachMovie	97	48	2	50
Census	10	3	1	4

Namely, for multinomial distributions, we used a uniform Dirichlet parameter prior (also commonly referred to as the "K2 prior"), and for the Gaussian distributions, we used a Normal-Wishart parameter prior having a prior mean of zero (equivalent sample size one) and a prior precision of one (equivalent sample size two).[1]

The predictive accuracy of learned models are shown in Table 3. *Empty* is a Bayesian network with no edges. Overall, the Bayesian networks produced by DN2BN were almost as accurate as those produced by BNET. Quantitatively, the gains of the models produced by DN2BN over Empty were 79% to 93% of those produced by BNET over Empty.

Table 4 describes statistics of the structures produced. The models produced by DNET always had the most edges, and those produced by DN2BN always had the fewest. The edge counts of DN2BN's models were 25% to 77% of those of BNET's models. (Excluding Census the range was 72% to 77%.) Notice that DN2BN achieved nearly the same scores as BNET while producing networks with many fewer edges.

We instrumented DN2BN to keep a detailed record of the operations it performed during its runs. Table 5 contains this information. Notice that the majority of the edge removals were performed to break length two cycles. In fact, breaking length two cycles and removing pruned parents nearly completed the conversion on all four data sets. There were a large number of pruned parents, suggesting that DN2BN would be improved by recovering some of this pruned structure—that is, by replacing pruned subtrees with structures more complex than leaves.

Finally, we compared the run times of the algorithms. Table 6 contains the results of this comparison. DN2BN's run times were quite short, two minutes or less on every data set. In fact, learning a dependency network and then running DN2BN has a speed advantage over BNET even on these relatively small data sets (that fit in RAM) and without the use of a scalable decision tree learning algorithm.

5 Summary

Algorithms that can convert models of one type into models of another type without accessing data allow us to do more with existing models and learning algorithms. We developed an algorithm, DN2BN, capable of converting dependency networks into Bayesian networks. We showed experimentally that our algorithm produced Bayesian networks with prediction accuracy almost equaling that of Bayesian networks learned from data directly. The advantages of our algorithm are that (1) it can create Bayesian networks without accessing data, (2) it is often faster than learning Bayesian networks directly, and (3) it can exploit mature and scalable decision tree induction algorithms.

We plan to improve DN2BN by allowing it to replace

[1]Before learning a decision tree for a continuous target variable, we first standardized the data so that the target had mean zero and variance one; thus the Normal-Wishart parameter prior is actually an empirical prior.

the subtrees it prunes with structure more complex than single leaves. In general, this requires access to sufficient statistics that are not explicitly encoded in the dependency network. We will explore a range of alternatives including approximating the needed statistics, inferring them from the complete dependency network using Gibbs sampling, and estimating them from training data. We plan to evaluate other search strategies including look-ahead and forward search (adding edges from the dependency network to an empty network until further additions introduce cycles). More generally, we plan to explore the use of dependency networks and Bayesian networks as alternatives for existing cached sufficient statistics structures in other settings.

References

[1] W. L. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8:195–210, 1996.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT press, 1990.

[3] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, 2000. ACM Press.

[4] N. Friedman, I. Nachman, and D. Peér. Learning Bayesian network structure from massive datasets: the “sparse candidate” algorithm. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 206–215. Morgan Kaufmann, 1999.

[5] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, October 2000.

[6] D. Heckerman. A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1996.

[7] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 525–531, Edmonton, Alberta, Canada, 2002. ACM Press.

[8] M. Jordan, editor. *Learning in Graphical Models*, volume 89. Kluwer, Boston, MA, NATO ASI, Series D: Behavioural and Social Sciences edition, 1998.

[9] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In M. I. Jordan, editor, *Learning in Graphical Models.* MIT Press, Cambridge, MA, 1999.

[10] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Communications*, pages 85–103. Plenum Press, 1972.

[11] P. Komarek and A. Moore. A dynamic adaptation of AD-trees for efficient machine learning on large data sets. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 495–502. Morgan Kaufmann, 2000.

[12] K. P. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *In Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence.* Morgan Kaufmann, 1999.