Compiling Bayesian Network Classifiers into Decision Graphs

Andy Shih, Arthur Choi, Adnan Darwiche
Computer Science Department
University of California, Los Angeles
{andyshih,aychoi,darwiche}@cs.ucla.edu

Abstract

We propose an algorithm for compiling Bayesian network classifiers into decision graphs that mimic the input and output behavior of the classifiers. In particular, we compile Bayesian network classifiers into *ordered* decision graphs, which are tractable and can be exponentially smaller in size than decision trees. This tractability facilitates reasoning about the behavior of Bayesian network classifiers, including the explanation of decisions they make. Our compilation algorithm comes with guarantees on the time of compilation and the size of compiled decision graphs. We apply our compilation algorithm to classifiers from the literature and discuss some case studies in which we show how to automatically explain their decisions and verify properties of their behavior.

1 Introduction

Bayesian network classifiers have been used extensively in the literature (Friedman, Geiger, and Goldszmidt 1997; Ng and Jordan 2002; Pernkopf and Bilmes 2005; Roos et al. 2005). These classifiers encode a probability distribution over a set of features and a class variable, and then classify an observation on features based on the posterior marginal of the class variable. Although the classification process computes probabilities, the classifier's behavior can be captured as a discrete mapping from the states of features (instances) to the states of the class variable (classes). We refer to this mapping as the classifier's *decision function.* Obtaining the decision function in a symbolic, tractable form was recently shown to be useful for explaining the decisions and formally verifying the properties of classifiers (Shih, Choi, and Darwiche 2018a; 2018b).

Previous work proposed algorithms for compiling Naive Bayes and latent tree classifiers into decision graphs (Chan and Darwiche 2003; Shih, Choi, and Darwiche 2018b). In this paper, we propose an algorithm that compiles a Bayesian network classifier with an arbitrary structure. We then illustrate the utility of our compilation algorithm through case studies, in which we symbolically reason about the behavior of some Bayesian network classifiers from the literature. This follows a recent trend in analyzing machine learning models using symbolic approaches (Narodytska et al. 2018; Katz et al. 2017).

In particular, our algorithm compiles a Bayesian network classifier into an *ordered decision graph,* a well-known and tractable class of decision graphs in which features are tested in the same order, along each path from the root of the graph to any of its leaves. When the features and the class variable are binary, the ordered decision graph is known in the literature as an *Ordered Binary Decision Diagram (OBDD)* (Bryant 1986; Meinel and Theobald 1998; Wegener 2000). When only the class variable is binary, it is known as an *Ordered Decision Diagram (ODD)* (Chan and Darwiche 2003). In this paper, we focus on the case of binary classifiers, but discuss an adaptation of our proposed compilation algorithm to support multi-class classifiers.

This paper is structured as follows. Section 2 provides an introduction to Bayesian network classifiers and ODDs. Sections 3 and 4 provide key results on Bayesian network classifiers, which are utilized by our compilation algorithm in Section 5. Experimental results relating to scalability are then given in Section 6, followed by a case study in Section 7. We finally conclude in Section 8.

2 Background

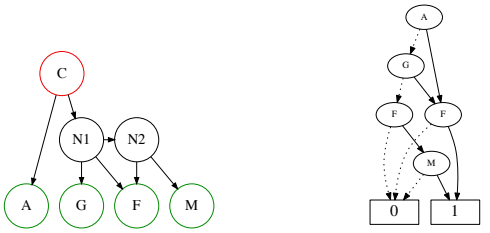
In this section, we describe Bayesian network classifiers and Ordered Decision Diagrams in more detail.

Bayesian Network Classifiers

A *Bayesian network classifier* is a Bayesian network containing a special set of variables: a single *class* variable C and n *feature* variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. The class C is usually a root in the network and the features $\mathbf{X}$ are usually leaves. An instantiation of variables $\mathbf{X}$ is denoted $\mathbf{x}$ and called an *instance.* When the class variable is binary, its two values c and $\bar{c}$ correspond to 1 and 0 decisions. A binary Bayesian network classifier specifying probability distribution $Pr(.)$ will classify an instance $\mathbf{x}$ as 1 iff $Pr(c \mid \mathbf{x}) \geq T$, where T is called the classification *threshold.* We denote the decision function of a classifier B as F_B.

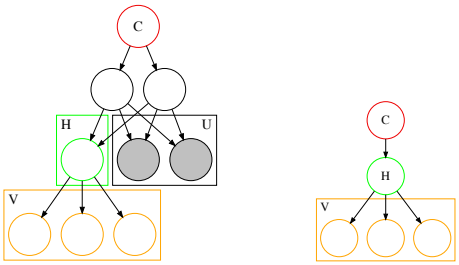
Ordered Decision Diagrams

An *Ordered Binary Decision Diagram* (OBDD) is a *tractable* representation of a Boolean function over variables $\mathbf{X} = X_1, \ldots, X_n$ (Bryant 1986; Meinel and Theobald 1998; Wegener 2000). An OBDD is a rooted, directed acyclic



(a) Bayesian network classifier (b) Ordered Decision Diagram

Figure 1: A Bayesian network classifier and its corresponding decision graph. The details of the Bayesian network classifier are provided in Table 3 in the Appendix.



(a) Classifier B. (b) Subclassifier $B^{\mathbf{H}}_{\mathbf{u}}$.

Figure 2: Variable H splits features into $(\mathbf{U}, \mathbf{V})$. When given an instantiation $\mathbf{u}$ on the variables $\mathbf{U}$, we can construct a subclassifier that has the same decision function as the original classifier over variables $\mathbf{V}$.

graph with two sinks called the 1-sink and 0-sink. Every node (except the sinks) in the OBDD is labeled with a variable X_i and has two labeled outgoing edges: the 1-edge and the 0-edge. The labeling of the OBDD nodes respects some global ordering of the variables $\mathbf{X}$: if there is an edge from a node labeled X_i to a node labeled X_j, then X_i must come before X_j in the global ordering. To evaluate the OBDD on an instance $\mathbf{x}$, start at the root node of the OBDD and let x_i be the label of the current node. Repeatedly follow the x_i-edge of the current node, until a sink node is reached. Reaching the 1-sink means $\mathbf{x}$ is evaluated to 1 and reaching the 0-sink means $\mathbf{x}$ is evaluated to 0 by the OBDD.

An OBDD is defined over binary variables, but can be extended to discrete variables with arbitrary values. This is called an ODD: a node labeled with variable X_i has one outgoing edge for each value of variable X_i. Hence, an OBDD/ODD can be viewed as representing a function $f(\mathbf{X})$ that maps instances $\mathbf{x}$ into $\{0, 1\}$.

Consider the Bayesian network classifier in Figure 1a, which classifies a movie as being a box office success or not. It has four binary features: A (*Adapted Screenplay*), G (*Great Cinematography*), F (*Famous Cast*), and M (*Marketing*). The ODD in Figure 1b captures the input and output behavior of the classifier in a tractable form. That is, given an instantiation of the features $\{A$=1,G=1,F=1,M=1$\}$, we start from the root node of the ODD and repeatedly follow the solid edge if the feature of the current node is set to 1, and follow the dotted edge otherwise. We will reach either the 0-sink or the 1-sink, which tells us if the instance is classified as 0 or 1. Moreover, the reached decision is guaranteed to match the one obtained from the Bayesian network classifier in Figure 1a. Given this tractable representation, we follow the approach of (Shih, Choi, and Darwiche 2018b) to efficiently explain the decisions of the Bayesian network classifier.

Consider a movie that is an adapted screenplay, has great cinematography, a famous cast, heavy marketing, and is classified as being a box office success. This movie corresponds to features $\{A$=1,G=1,F=1,M=1$\}$ and a classification of 1. Using the ODD in Figure 1b we can deduce, in time linear on the size of the ODD, that the movie could have had poor cinematography and low marketing, and would still be classified as being a box office success. In fact, the partial instantiation $\{A$=1,F=1$\}$ completely determines that the movie will be classified as being successful, regardless of how the remaining features are set. Furthermore, we can formally verify that the classifier is monotonic. That is, any instance classified as 1 remains classified as 1 even if some of its features are flipped from 0 to 1.

The above analysis is exemplary of many more queries that one can efficiently answer about a Bayesian network classifier, once it has been compiled into an ODD. We describe below some more queries that can also be efficiently answered (Shih, Choi, and Darwiche 2018a).

- Robustness: Given an instance, what is the least number of features we need to flip to change its classification?
- Irrelevant Features: Are there features that do not impact any classification?
- If-Then Rules: Does the classifier satisfy some given rules of thumb?

The above queries can be computed in linear or quadratic time in the size of the compiled ODD. This highlights the significance of compiling Bayesian network classifiers into ODDs. Next, we present some key observations about Bayesian network classifiers that we exploit later.

3 Subclassifiers

Our compilation algorithm is based on recursively decomposing into smaller classifiers and identifying those that are equivalent to avoid the compilation of a classifier if an equivalent one has already been compiled. In this section we present the method of decomposing into smaller classifiers, which can be described at a high level as follows. Given a Bayesian network classifier B, let $\mathbf{U}$ and $\mathbf{V}$ be a partition of its features $\mathbf{X}$ and let H be a variable not in $\mathbf{X}$. Suppose now that we are only interested in classifying instances $\mathbf{x}$ that set features $\mathbf{U}$ to some state $\mathbf{u}$. When certain conditions hold, we can perform the classification using a *smaller* classifier, which is obtained from classifier B as follows:

– Node H is disconnected from its parents and C, the class node, is added as the new parent of H.

– A CPT is assigned to H based on inference on B.
– A prior is assigned to C based on inference on B.
– Leaves are repeatedly removed from classifier B as long as they are not in $C \cup H \cup \mathbf{V}$.

The resulting classifier is called a *subclassifier*. Figure 2 depicts an example of the structural changes needed to construct a subclassifier. The main insight regarding these subclassifiers is that they compute the same decision function over the remaining variables $\mathbf{V}$ as the original classifier. Once we fix $\mathbf{u}$ and obtain a subclassifier $B^H_{\mathbf{u}}$ from an original classifier B, the decision function $F_{B^H_{\mathbf{u}}}(\mathbf{v})$ will output the same classification as $F_B(\mathbf{uv})$. This key property will be used in the algorithm to reduce the compilation of a classifier into the compilation of subclassifiers.

We will now spell out the above result on subclassifiers. We first state the conditions under which a subclassifier can be constructed.

Definition 1 *Let $(\mathbf{U}, \mathbf{V})$ be a partition of the features $\mathbf{X}$ in a Bayesian network classifier B, and let H be a variable outside $\mathbf{X}$. We say that H **splits** features $\mathbf{X}$ into $(\mathbf{U}, \mathbf{V})$ iff H d-separates features $\mathbf{V}$ from C and $\mathbf{U}$.*

Recall that $\mathbf{Z}$ d-separates $\mathbf{X}$ from $\mathbf{Y}$ if $\mathbf{X}$ and $\mathbf{Y}$ are independent given $\mathbf{Z}$ (Darwiche 2009). We are now ready to define subclassifiers.

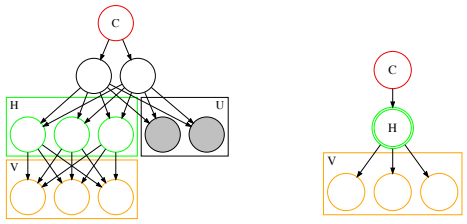
Definition 2 *Let B be a Bayesian network classifier, H be a variable that splits features into $(\mathbf{U}, \mathbf{V})$, and $\mathbf{u}$ be an instantiation of features $\mathbf{U}$. The **subclassifier** for H and $\mathbf{u}$, denoted $B^H_{\mathbf{u}}$, is obtained from classifier B as follows:*

1. *Disconnect node H from its parents.*
2. *Make H a child of class variable C, and set its Conditional Probability Table (CPT) to $P(H|C\mathbf{u})$.*
3. *Set the CPT of C to $P(C|\mathbf{u})$.*
4. *Repeatedly remove every leaf node from B that is not in $C \cup H \cup \mathbf{V}$.*

Constructing a subclassifier requires some computational work on the original classifier B. First, we need to identify a variable H that satisfies the condition of Definition 1. This can be done in polynomial time as it only involves reasoning about d-separation (Darwiche 2009). Second, we need to determine the CPTs of H and C, which require the computation of posteriors on the H and C, given the state $\mathbf{u}$ of features $\mathbf{U}$. This requires exact inference on the classifier B. We will later provide a bound on the number of inference calls made by our compilation algorithm for this purpose. The next theorem formalizes the property of subclassifiers.

Theorem 1 *Let B be a Bayesian network classifier and let H be a variable that splits features into $(\mathbf{U}, \mathbf{V})$. For a subclassifier $B^H_{\mathbf{u}}$, we have $F_B(\mathbf{uv}) = F_{B^H_{\mathbf{u}}}(\mathbf{v})$ for all instantiations $\mathbf{v}$ of features $\mathbf{V}$.*

According to this theorem, the classification of an instance $\mathbf{uv}$ by classifier B will match the classification of $\mathbf{v}$ by subclassifier $B^H_{\mathbf{u}}$. As we shall see, when our compilation algorithm fixes the state of features $\mathbf{U}$ to $\mathbf{u}$, it will construct and recursively compile the subclassifier $B^H_{\mathbf{u}}$.



(a) Classifier B. (b) Subclassifier $B^{\mathbf{H}}_{\mathbf{u}}$.

Figure 3: A construction of a subclassifier using a set of splitting nodes $\mathbf{H}$.

Multiple Splitting Nodes

For Bayesian network classifiers with dense structure, there may not exist a single node H that splits the features into two parts. In such cases, we can actually relax the splitting node to the case of a set of nodes, using the formulation of a meganode (Russell and Norvig 2016). More specifically, a set of nodes in a Bayesian network can sometimes be replaced by a single meganode whose state space is the Cartesian product of the state spaces of the set of nodes (Figure 3). To simplify the discussion, for the rest of the paper we will only focus on the case where H is a single node.

4 Equivalent Classifiers

We will now introduce the second key result that will form the basis of our algorithm for compiling a Bayesian network classifier into an ODD. This result provides a method for detecting when two "similar" Bayesian network classifiers induce the same decision function. In this section we assume, without loss of generality, that the classifier has a class node C which has a single child H. This assumption is satisfied by all subclassifiers and can easily be satisfied for any classifier by adding a dummy class node with the original class node as its single child. First we define when two classifiers are considered similar.

Definition 3 *Let B be a Bayesian network classifier with a class node C which has a single child node H. A second Bayesian network classifier is **similar** to B if it has the same structure as B and differs only in the CPTs of C and H.*

Let $\mathbf{X}$ be the feature variables of two similar classifiers B and B'. Note that $P(\mathbf{X}|H)$ is the same across the two classifiers, and H d-separates C from $\mathbf{X}$ by our earlier assumption. Thus, we can rewrite $P(C|\mathbf{X})$ as follows:

$$\begin{aligned} P(C|\mathbf{x}) &= \sum_h P(C|h)P(h|\mathbf{x}) \\ &= \sum_h P(C,h)P(\mathbf{x}|h)/P(\mathbf{x}) \end{aligned} \quad (1)$$

So far, the results from Section 3 and this section have not assumed that the class variable C and the variable H are binary. These results will be used in the context of the compiling multi-class classifiers in Section 5. For the rest of this

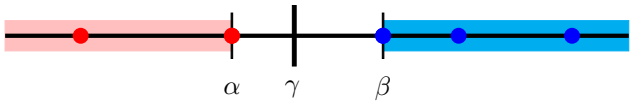


Figure 4: Visualization of the equivalence interval of a classifier B. The red dots represent instances classified as 1 and the blue dots represent instances classified as 0. A classifier that is similar to B shares the same decision function as B if its coefficient γ falls within the equivalence interval of B, depicted by the white region between α and β.

section, we will assume that nodes C and H are binary, and the classification threshold is t. In this setting, we have an efficient way of detecting when two similar classifiers share the same decision function, in time linear in the number of features $\mathbf{X}$. We present the details next.

Setting $a_h = P(c, H = h) - tP(H = h)$, we can rewrite the classification as a linear inequality.

$$\sum_h P(c,h)P(\mathbf{x}|h) \geq tP(\mathbf{x})$$

$$\sum_h (P(c,h) - tP(h))P(\mathbf{x}|h) \geq 0$$

$$\sum_h a_h P(\mathbf{x}|h) \geq 0 \qquad (2)$$

For two similar classifiers, the values a_h vary but $P(\mathbf{x}|h)$ is the same. To detect if two similar classifiers share the same decision function, we just need to verify that the two sets of values a_h classify all instances $\mathbf{x}$ the same way. To do so, we define the sign, margin, and coefficient of such classifiers.

Definition 4 *Let B be a non-trivial[1] Bayesian network classifier with a threshold t and a binary class node C which has a single binary child node H. Let σ denote the **sign** of the classifier, which is defined to be 1 if $P(c|H = 1) \geq t$ and 0 otherwise. The **margin** α, β and **coefficient** γ of B are defined as follows:*

$$\alpha = \max_{\mathbf{x}:F_B(\mathbf{x})=1} P(\mathbf{x}|H = 1-\sigma)/P(\mathbf{x}|H = \sigma)$$

$$\beta = \min_{\mathbf{x}:F_B(\mathbf{x})=0} P(\mathbf{x}|H = 1-\sigma)/P(\mathbf{x}|H = \sigma)$$

$$\gamma = -1 \cdot \frac{P(c, H = \sigma) - tP(H = \sigma)}{P(c, H = 1-\sigma) - tP(H = 1-\sigma)}$$

That is, α is the largest value of $P(\mathbf{x}|H = 1-\sigma)/P(\mathbf{x}|H = \sigma)$ attained by any instance classified as 1, and β is the smallest such value attained by any instance classified as 0 (see Figure 4). The values α, β, and γ come from a rearrangement of Equation 2 for the case of a binary H variable. The notion of a margin was actually identified by (Chan and Darwiche 2003) in connection to Naive Bayes classifiers, and turns out to apply to general Bayesian network classifiers.

The next result was proven only for Naive Bayes classifiers in (Chan and Darwiche 2003). Our generalization is phrased differently and is more succinct.

[1] A non-trivial classifier with a binary class node classifies at least one instance as 1 and at least one instance as 0.

Theorem 2 *Let B be a non-trivial Bayesian network classifier with a binary class node C and a single binary child node H. Let B' be a second classifier that is similar to B and has the same sign as B. Let t be their threshold, (α, β) be the margin of classifier B, and γ be the coefficient of B'. The two classifiers have the same decision function, $F_B = F_{B'}$, iff γ belongs to the interval $[\alpha, \beta)$. This is called the equivalence interval of classifier B.*

The above theorem enables us to perform binary search over equivalence intervals to identify equivalent subclassifiers: ones that lead to the same decision function and hence the same compilation. This technique avoids the compilation of a subclassifier if an equivalent one has already been compiled.

5 From Numeric Bayesian Network Classifiers to Symbolic ODDs

We now present our algorithm for compiling a Bayesian network classifier B into an ODD. We address the case of binary classifiers first and then describe the adaptation of our algorithm to multi-class classifiers.

The overall structure of the algorithm is pretty simple. We first identify a binary variable H that splits the features into $(\mathbf{U}, \mathbf{V})$. We then start enumerating over the values of features in $\mathbf{U}$ as if we are building a decision tree (in a depth-first manner). Each leaf of this decision tree corresponds to a distinct instantiation $\mathbf{u}$ and a subclassifier $B^H_{\mathbf{u}}$ with its equivalence interval. These subclassifiers are similar to one another, since they differ only in the CPTs of class C and variable H. Our algorithm will then compile these subclassifiers recursively using the same technique, except that it will avoid compiling a subclassifier if it already compiled an equivalent one—as determined by Theorem 2.

The efficiency of this algorithm depends on the choice of variable H and the corresponding feature decomposition $(\mathbf{U}, \mathbf{V})$ as we want $\mathbf{U}$ to be as small as possible. We identify such feature decompositions in a preprocessing step. That is, after first decomposing features into $(\mathbf{U}, \mathbf{V})$, using an appropriate H, we follow by decomposing $\mathbf{V}$ recursively. This leads us to the notion of a *block ordering* of features.

Definition 5 *Given a Bayesian network classifier, a **block ordering** of its features $\mathbf{X}$ is a sequence $\pi = (\mathbf{X}_1, \ldots, \mathbf{X}_m)$ such that $\mathbf{X}_1, \ldots, \mathbf{X}_m$ is a partition of features $\mathbf{X}$, and for each $0 < k < m$, there exists a binary variable H that splits features $\mathbf{X}$ into $(\mathbf{X}_1 \cup \ldots \cup \mathbf{X}_k, \mathbf{X}_{k+1} \cup \ldots \cup \mathbf{X}_m)$.*

Each element $\mathbf{X}_i$ is called a *block* of the block ordering π. We will assume that the features in a block are ordered (arbitrarily). As such, we will refer to features by their position in the block ordering π.

We will later discuss a heuristic for obtaining a block order, which we used in our experiments. But for now, we will discuss Algorithms 1 and 2. Algorithm 1 is passed a Bayesian network classifier B and a block ordering π of features. It creates the sinks of the ODD and calls Algorithm 2.

Algorithm 2 implements the proposal we discussed earlier. It maintains a cache that stores tuples of the form (D, I, σ, k), where D is an ODD node, I is an equivalence

Algorithm 1 `compile-classifier`(B, π)

input: Bayesian network classifier B and block ordering π of features

output: ODD for the decision function of classifier B

main:

```
0-sink ← terminal ODD node labeled with 0
1-sink ← terminal ODD node labeled with 1
D ← compile-subclassifier(B, {}, π, 0)
return reduced form of D
```

Algorithm 2 `compile-subclassifier`$(B, \mathbf{u}, \pi, k)$

input: Bayesian network classifier B, instantiation $\mathbf{u}$ of some features, block ordering π of features, integer k

output: ODD for the decision function of classifier B

main:

```
if u is an instantiation of a block in ordering π then
    B ← get-subclassifier(B, u, π, k)
    if B has no feature variables then
        return get-sink(B)
    γ, σ ← coefficient and sign of B
    D ← find-in-cache(γ, σ, k)
    if D = null then
        D ← compile-subclassifier(B, {}, π, k)
        I ← equivalence-interval(D)
        store-in-cache(D, I, σ, k)
    return D
X ← feature at position k in ordering π
S ← {}
for each state x of feature X do
    C ← compile-subclassifier(N, u ∪ x, π, k + 1)
    add (C, x) to set S
return get-odd-node(S)
```

interval, σ is a boolean, and k is an integer. Such a cache entry means that ODD D is the result of compiling a subclassifier $B_{\mathbf{u}}$ that has equivalence interval I and sign σ. It also means that the last feature in block $\mathbf{U}$ is at position $k-1$ in the block ordering π. The cache is fetched based on a coefficient γ, a sign σ and a level k. That is, it returns ODD D if $\gamma \in I$ for the same σ and same k.

Algorithm 2 makes use of four auxiliary functions. First, `get-subclassifier`$(B, \mathbf{u}, \pi, k)$ constructs a subclassifier and requires a constant number of calls to an exact inference algorithm to get the coefficients of the subclassifier. `get-sink`(B) takes in a subclassifier with no more features, and returns either the `0-sink` or the `1-sink` based on a simple check. `equivalence-interval`(D) computes the equivalence interval of the classifier leading to ODD D. This is done in constant time using the equivalence intervals for the children of D (Chan and Darwiche 2003). Finally, `get-odd-node`(S) returns an ODD node, which is defined by the set S that specifies the node's children and the labels of edges pointing to these children.

Algorithm 3 implements a simple, greedy heuristic for obtaining a block ordering of features. Its running time is $O(n^4)$, where n is the number of features, which was sufficient for our experiments.

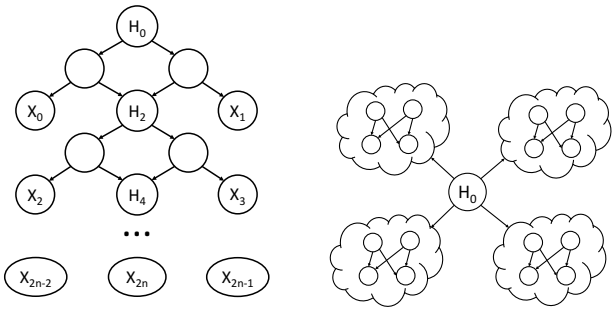
Algorithm 3 `block-order`$(B, \mathbf{X})$

input: A Bayesian network classifier B with features $\mathbf{X}$

output: A block ordering π of the features $\mathbf{X}$

main:

```
H, U, V ← class variable of B, X, ∅
for each variable H' that splits features X into (U', V') do
    if |U'| ≤ |U| then
        H, U, V ← H', U', V'
u ← some instantiation of U
return U, block-order(B_u^H, V)
```



(a) Ladder classifier (variables X_i are features)

(b) Cluster classifier (clouds contain arbitrary structure)

Figure 5: Examples of classifier families with improved compilation time.

We close this section by providing time and size bounds on our compilation algorithm. We later show that for certain classes of Bayesian networks, these bounds can be as tight as the bounds provided by (Chan and Darwiche 2003) for compiling Naive Bayes classifiers into ODDs.

Definition 6 *Let $\pi = (\mathbf{X}_1, \ldots, \mathbf{X}_m)$ be a block ordering of the features in a Bayesian network classifier B. Let p_i denote the size of the state space of the features in block i, and let $s(i,j) = \log_2(p_i \times \ldots \times p_j)$. The **width** w_π of this order and the **compilation width** w_B of classifier B are defined as:*

$$w_\pi = \max_{i \in \{1,\ldots,m\}} \lceil p_i \cdot \min(s(1, i-1), s(i, m)) \rceil$$
$$w_B = \min_\pi w_\pi.$$

We now have the following bounds on Algorithm 1.

Theorem 3 *The number of nodes in the ODD returned by Algorithm 1 is $O(2^{w_\pi})$, where w_π is the width of order π. Moreover, the running time of the algorithm is $O(PT + w_\pi 2^{w_\pi})$, where P is the sum of the state space sizes of blocks in π and T is the time of an inference call on the classifier.*

Consider the family of ladder classifiers depicted in Figure 5a, which has $n = 2m + 1$ features. We can use the sequence of nodes $H_2, H_4, \ldots, H_{2m}$ to decompose features, leading to the block ordering

$$[X_0, X_1], [X_2, X_3], \ldots, [X_{2m-2}, X_{2m-1}, X_{2m}],$$

which has width $n/2$, assuming binary features. The size of the ODD is $O(2^{n/2})$ and the running time is $O(nT + n2^{n/2})$.

The family of cluster classifiers in Figure 5b has similar bounds. Assume that we have n features and k clusters, with each cluster having n/k features. We can repeatedly use the node H_0 to split features into k blocks of size n/k, leading to a block order width $n/2$, assuming binary features, and a largest block size n/k. The size of the ODD is $O(2^{n/2})$ and the running time is $O(k2^{n/k}T + n2^{n/2})$.

What is interesting about these bounds is that they match the ones for compiling Naive Bayes classifiers to ODDs (Chan and Darwiche 2003)—an NP-hard problem as shown by (Shih, Choi, and Darwiche 2018b). In practice, however, the time and size complexity of Algorithm 1 can be significantly better as we show in Section 6.

Multi-class Classifiers

We can adjust our algorithm to support the compilation of multi-class classifiers. For a multi-class classifier B, an instance is classified based on the highest posterior probability among all states of the class variable. That is, $F_B(\mathbf{x}) = \text{argmax}_c P(c|\mathbf{x})$. Since the output is non-binary, we use a variation of the ODD called the Algebraic Decision Diagram (ADD), which has multiple sinks, one for each state of the class node (Bahar et al. 1997). ADDs are tractable, so we can still explain and perform verification on the behavior of multi-class classifiers efficiently.

The construction of a subclassifier in Section 3 generalizes directly to multi-class classifiers. Equation 1 holds for multi-class classifiers as well. Our current formulation of equivalence intervals is designed for binary classifiers, so for future work we will look into generalizing equivalence intervals to multi-class classifiers.

We can still obtain a bound on the compilation time using a naive method of detecting equivalent subclassifiers. First, we modify Algorithm 1 to support multiple sinks. We modify Algorithm 2 so that instead of finding an equivalent subclassifier using binary search on equivalence intervals, we enumerate over all instances of the subclassifier if the number of instances is small enough. Using Equation 1 to share inference calls among similar subclassifiers, we can get a bound on the running time of our adapted compilation algorithm on multi-class classifiers.

Theorem 4 *The running time of the adapted version of Algorithm 1 on a multi-class Bayesian network classifier with a block ordering π is $O(PT + 2^s)$, where P is the sum of the state space sizes of all blocks in π, T is the time of an inference call on the classifier, and s is the log of the product of the state space sizes of blocks in π.*

Reducing the classifier into subclassifiers provides significant savings on the number of exact inference calls. The number of computations may be greatly reduced once we develop equivalence intervals for multi-class classifiers.

6 Experiments

Table 1 summarizes compilation experiments we ran on three binary Bayesian networks using all leaf nodes as features. For each network we included a number of classifiers, each corresponding to one root of the network, using a threshold of $\frac{1}{2}$. Table 2 provides similar results on two other binary networks, except in this case we sampled some of the leaf nodes to include as features (the networks were too large to fully compile). Inference calls were performed using the SamIAm library.[2]

Table 1: win95pts has 76 nodes, 16 features and width 9. Andes has 223 nodes, 24 features and width 18. cpcs54 has 54 nodes, 13 features and width 14. Width refers to the network tree-width, approximated by the minfill heuristic.

network	class	block order width	largest / # of blocks	ODD size	compile time (s)
win95pts	GRDS	15	15 / 2	498	21
win95pts	NP	14	14 / 3	413	11
win95pts	NC	14	14 / 3	430	12
win95pts	PO	15	15 / 2	291	21
win95pts	POOK	15	15 / 2	285	21
win95pts	PAT	13	13 / 4	636	5
win95pts	PDrvr	14	14 / 3	352	11
win95pts	PMem	13	13 / 4	890	5
win95pts	POn	14	14 / 3	31	11
win95pts	PPpr	14	14 / 3	31	10
Andes	TK	23	18 / 7	47	11,708
Andes	VKE	19	19 / 6	2,107	27,495
Andes	CNBG	19	19 / 6	2,893	24,374
Andes	MDA	19	19 / 6	5,454	26,614
cpcs54	x3	12	12 / 2	25	69
cpcs54	x4	12	12 / 2	92	69
cpcs54	x7	12	12 / 2	13	69
cpcs54	x8	12	12 / 2	20	69
cpcs54	x9	11	11 / 3	13	35

The actual sizes of the resulting ODDs are much smaller than the theoretical upper bounds. For example, the size of the ODD of the Andes classifier with root ValueKnownEq(VKE) is less than 1% of the bound given by the block order width. The limiting factor for the compilation algorithm is the compilation time, which depends on the treewidth of the network and scales exponentially with respect to the largest feature block. The treewidth affects the time of each inference call, and the largest feature block bounds the number of inference calls made by the compilation algorithm. For example, the two emdec6g experiments in Table 2 with 27 and 30 features differ only by 2 in block order width. But, since they differ by 11 in the largest feature block, we notice a large jump in the compilation time for these two experiments (two orders of magnitude). On the other hand, the experiment with 30 and 33 features also differ by 2 in block order width. Since their largest feature blocks differ only by 2, the compilation times are comparable (factor of 2). The ODD size and compilation time are also significantly affected by the threshold of the classifier. A heavily biased threshold can lead to a very small ODD and a short compilation time, while a balanced threshold generally leads to larger ODDs.

Our results suggest that the compilation algorithm can be used to implement and deploy Bayesian network classifiers

[2] Available at http://reasoning.cs.ucla.edu/samiam/

Table 2: Network tcc4e has 98 nodes and width 10. Network emdec6g has 168 nodes and width 7. We use t27 as the class node for tcc4e, and x29 as the class node for emdec6g.

network	# sampled features	block order width	largest / # of blocks	ODD size	compile time (s)
tcc4e	21	15	11 / 7	167	4
tcc4e	26	19	14 / 8	930	11
tcc4e	30	30	20 / 6	3,057	1,873
tcc4e	37	37	25 / 8	10,442	39,705
tcc4e	38	38	26 / 8	22,508	91,332
emdec6g	24	24	7 / 13	115	6
emdec6g	27	27	10 / 10	122	11
emdec6g	30	29	21 / 7	4,154	2,487
emdec6g	33	31	22 / 8	3,855	5,308

more effectively, given that they may induce small ODDs that can be evaluated in linear time, without requiring floating point computations. In the next section, we show another major application: the ability to reason *symbolically* about the behavior of a Bayesian network classifier.

7 Case Study: Explaining Classifiers

We illustrate the utility our compilation algorithm, showing how the resulting ODDs can be used to explain and verify a given classifier. We consider two networks from the literature: win95pts and Andes; see Table 1. We treat each network as a set of classifiers, taking each root node as a class variable; each leaf node is treated as a feature. We assume a threshold of $\frac{1}{2}$.

We compile an ODD for each classifier and then explain their decisions using two types of explanations that were proposed in previous work: minimum cardinality (MC) explanations and prime implicant (PI) explanations (Shih, Choi, and Darwiche 2018b). An MC-explanation of a positive decision identifies a minimal set of 1-features that are responsible for the decision. That is, the decision will stick even if all other features are set to 0. In contrast, a PI-explanation identifies a minimal set of features (regardless of their state) that are responsible for the current decision. That is, one can toggle all other features in any fashion without changing the decision. Given an ODD representing the decision function of a classifier, many types of explanations become tractable (in contrast to the case of unordered decision graphs). Moreover, certain types of formal verification become tractable as well, e.g., testing if the classifier is monotonic (Shih, Choi, and Darwiche 2018a).

The win95pts network is used to diagnose why a printing job has failed (Breese and Heckerman 1996). It has 76 binary variables, 16 of which are leaves which we take as the features of the classifier. One of its root nodes PtrOffline (PO) represents a failure mode (the printer is offline), and has two states: Online (0) and Offline (1). An instance is classified positively if the the probability of being Offline is $\geq \frac{1}{2}$. We first consider a positively classified instance (indicating a printing failure) that sets 7 of 16 features as 1. The unique MC-explanation for this decision consists of a *single* feature set to 1: the printer icon is grayed out (PrtIcon is GrayedOut (1)). That is, observing this one symptom positively is sufficient for a positive decision (printer is offline), even if all of the other 6 features were observed as 0 instead of 1. For a technician using such a classifier for decision support, this suggests that the troubleshooting of a printer failure should focus on this observation, as it is the most pertinent among all positively observed features.

Consider the shortest PI-explanation of this positive instance, which consists of three features: the printer icon is grayed out (PrtIcon is GrayedOut), the problem is repeatable (NotRepeat is No), and the graphics are not distorted (GraphicsDistorted is No). With just these three observations, the classifier will always decide that the printer is offline (PtrOffline is Offline), no matter how the other features are observed. Such a guarantee can help users trust the classifier, especially if its behavior matches the users' intuition. In fact, users can even enter their own partial observation of interest (say, PrtIcon is Normal and GraphicsDistorted is Yes) and check if the classifier is guaranteed to behave according to their expectations (say, decide that PtrOffline is Online) regardless of how the remaining features are set.

Next, we consider the Andes network, which models students' problem-solving skills in physics (Gertner, Conati, and VanLehn 1998). We consider the class node TryKinematics (TK), which has two states: false (0) and true (1). This class predicts whether a student has developed problem-solving skills in kinematics, and assesses the student positively if the probability of true is $\geq \frac{1}{2}$. This classifier has 24 binary features. First, we verify whether the classifier is monotonic or not: it is indeed monotonic. Next, we consider a positively classified instance that observed 5 of these features as 1. The MC-explanation tells us that 3 of these 5 features are responsible for the decision: TryKinematicsForAccel, TryKinematicsForDuration, and TryKinematicsForDisplacement. That is, we can flip the other two features to 0 and still maintain a positive classification. We can also efficiently test whether the classification of this instance is robust, given an ODD of the classifier's decision function (Shih, Choi, and Darwiche 2018a). In our example, it only takes a single feature to be flipped (from 1 to 0) to flip the decision to negative.

8 Conclusion

We presented an algorithm for compiling Bayesian network classifiers with arbitrary structure into tractable decision graphs in the form of ODDs. These decision graphs capture the input and output behavior of the classifiers and can be used to efficiently reason about a classifier's behavior. We provided a bound on the time complexity of our algorithm and a bound on the resulting ODD size, showing they can be as tight as previous bounds for compiling Naive Bayes classifiers. We managed to compile Bayesian network classifiers with over a hundred nodes and thirty features into compact ODDs of only a few thousand nodes. We also presented a case study where we examined the compiled decision graphs of classifiers from literature and symbolically reasoned about their behavior.

Acknowledgments This work has been partially supported by NSF grant #IIS-1514253, ONR grant #N00014-18-1-2561 and DARPA XAI grant #N66001-17-2-4032.

A Appendix

Table 3: Details of Bayesian network classifier in Figure 1a

Threshold	0.5
$C=1$	0.5

C	0	1
$N1=1$	0.3	0.9

$N1$	0	1
$G=1$	0.5	0.6

C	0	1
$A=1$	0.6	0.7

$N1$	0	1
$N2=1$	0.3	0.7

$N2$	0	1
$M=1$	0.2	0.9

$N1,N2$	00	01	10	11
$F=1$	0.1	0.8	0.3	0.9

B Proofs

Proof of Theorem 1 We want to show that $F_B(\mathbf{uv}) = F_{B^{\mathbf{H}}_{\mathbf{u}}}(\mathbf{v})$. We let $P(.)$ denote the probability distribution of the original classifier B and let $P'(.)$ denote the probability distribution of the subclassifier $B^{\mathbf{H}}_{\mathbf{u}}$. First, we work out the following equalities:

$$\begin{aligned}P(\mathbf{h}|\mathbf{u}) &= \sum_c P(\mathbf{h}|c\mathbf{u})P(c|\mathbf{u}) \\ &= \sum_c P'(h|c)P'(c) = P'(h)\end{aligned}$$

$$\begin{aligned}P(\mathbf{v}|\mathbf{u}) &= \sum_{\mathbf{h}} P(\mathbf{v}|\mathbf{hu})P(h|\mathbf{u}) \\ &= \sum_{\mathbf{h}} P(\mathbf{v}|\mathbf{h})P(\mathbf{h}|\mathbf{u}) \\ &= \sum_h P'(\mathbf{v}|h)P'(h) = P'(\mathbf{v})\end{aligned}$$

This gives us the following list of identities:

$$P(c|\mathbf{u}) = P'(c) \qquad P(\mathbf{h}|c\mathbf{u}) = P'(h|c)$$
$$P(\mathbf{h}|\mathbf{u}) = P'(h) \qquad P(\mathbf{v}|\mathbf{u}) = P'(\mathbf{v})$$

Next, we will show the main property we are after: for any c and $\mathbf{v}$, $P(c|\mathbf{uv}) = P'(c|\mathbf{v})$.

$$\begin{aligned}P(c|\mathbf{uv}) &= \sum_{\mathbf{h}} P(c|\mathbf{huv})P(\mathbf{h}|\mathbf{uv}) \\ &= \sum_{\mathbf{h}} P(c|\mathbf{hu})P(\mathbf{h}|\mathbf{vu}) \\ &= \sum_{\mathbf{h}} \frac{P(\mathbf{h}|c\mathbf{u})P(c|\mathbf{u})}{P(\mathbf{h}|\mathbf{u})} \frac{P(\mathbf{v}|\mathbf{hu})P(\mathbf{h}|\mathbf{u})}{P(\mathbf{v}|\mathbf{u})} \\ &= \sum_h \frac{P'(h|c)P'(c)}{P'(h)} \frac{P'(\mathbf{v}|h)P'(h)}{P'(\mathbf{v})} \\ &= \sum_h P'(c|h)P'(h|\mathbf{v}) = P'(c|\mathbf{v})\end{aligned}$$

Since the threshold is the same for the classifier B and the subclassifier $B^{\mathbf{H}}_{\mathbf{u}}$, it follows that $F_B(\mathbf{uv}) = F_{B^{\mathbf{H}}_{\mathbf{u}}}(\mathbf{v})$ for all $\mathbf{v}$. □

Proof of Theorem 2 We let $P(.)$ denote the probability distribution of the classifier B and let $P'(.)$ denote the probability distribution of the classifier B'. Using Equation 2 we can rewrite the classification decision of classifier B as $\sum_h a_h P(\mathbf{x}|h) \geq 0$, where $a_h = P(c, H = h) - tP(H = h)$. Since h is binary, we can expand the summation.

$$a_0 P(\mathbf{x}|H=0) + a_1 P(\mathbf{x}|H=1) \geq 0$$

Since the classifier B is nontrivial, we know that $a_0/a_1 < 0$. Suppose that the sign σ of B is 1, and thus $a_1 > 0$. Rearranging, we get:

$$-1 \cdot \frac{a_1}{a_0} \geq \frac{P(\mathbf{x}|H=0)}{P(\mathbf{x}|H=1)} \tag{3}$$

Now suppose $F_B(\mathbf{x}) = 1$ for some $\mathbf{x}$. Recall that α is the maximum value of $\frac{P(\mathbf{x}|H=0)}{P(\mathbf{x}|H=1)}$ attained by any instance classified as 1. Let $a'_h = P'(c, H = h) - tP'(H = h)$.

$$-1 \cdot \frac{a'_1}{a'_0} = \gamma \geq \alpha \geq \frac{P(\mathbf{x}|H=0)}{P(\mathbf{x}|H=1)}$$

So we have that $F'_B(\mathbf{x}) = 1$. The proof is analogous for instances classified as 0, as well as for classifiers with sign 0, thus $F_B(\mathbf{x}) = F'_B(\mathbf{x})$ for all $\mathbf{x}$. □

Proof of Theorem 3 Let $\pi = (X_1, \ldots, X_m)$ be the block ordering of the features, and let $s(i, j) = \log_2(p_i \times \ldots p_j)$, where p_l denotes the size of the state space of the features in block l. Let t_k be the number of ODD nodes from level $s(1, k-1)$ to level $s(1, k) - 1$, and note that $\sum_k t_k$ is the total number of nodes in the ODD.

We will bound the number of nodes in the ODD by bounding the number of nodes in each feature block. The number of ODD nodes on level $s(1, k-1)$ is bounded from above by $2^{s(1,k-1)}$ (by decision trees) and also by $2 \cdot 2^{s(k,m)}$ (by equivalence intervals). The bound of $2 \cdot 2^{s(k,m)}$ from equivalence intervals is due to the following observation. For the subclassifiers stored in cache of sign 1 and level $s(1, k-1)$, their classification decision on an instance $\mathbf{x}$ can be written as in Equation 3. Since the RHS of the inequality of Equation 3 is the same among all subclassifiers of sign 1 and level $s(1, k-1)$, and there are $2^{s(k,m)}$ distinct instances for such subclassifiers, there are at most $2^{s(k,m)}$ equivalence classes of subclassifiers of sign 1 and level $s(1, k-1)$. The analysis for subclassifiers of sign 0 and level $s(1, k-1)$ is the same, so we have at most $2 \cdot 2^{s(k,m)}$ equivalence classes for subclassifiers on level $s(1, k-1)$.

From level $s(1, k-1)$ to level $s(1, k) - 1$, the algorithm constructs the ODD in a decision tree manner. Therefore, we have that t_k is bounded by $p_k \cdot \min(2^{s(1,k-1)}, 2 \cdot 2^{s(k,m)})$. So, $t_j = O(2^{w_\pi})$, where $j = \text{argmax}_k(t_k)$. To finish, observe that both sequences $t_{j+1}, t_{j+2}, \ldots$ and $t_{j-1}, t_{j-2}, \ldots$ on either side of j decay exponentially fast, so we have that the total number of nodes is $\sum_k t_k = O(t_j) = O(2^{w_\pi})$.

Next we will bound the time complexity of the algorithm. We start by showing that the number of exact inference calls is $P = p_1 + \ldots + p_m$. This number is much smaller than the number of subclassifiers constructed, which is $O(2^{w_\pi})$, because we can share the results of inference calls across different subclassifier constructions.

We want to show that for multiple classifiers that are similar, the construction of their subclassifiers can reuse the same inference calls. For a set of similar classifiers, let H' be the child of class node C and let H be the new splitting node used to construct the subclassifiers. Note that to construct a subclassifier, we need the values $P(h|\mathbf{u}c)$ and $P(c|\mathbf{u})$.

$$P(h|\mathbf{u}c) = \sum_{h'} P(h|\mathbf{u}ch')P(h'|c)$$

The terms $P(h|\mathbf{u}ch')$ are actually the same across similar classifiers, since similar classifiers only differ in the CPTs of C and H' and those variables are fixed in these terms. As well, the terms $P(h'|c)$ do not require any inference at all, since these are just the CPTs encoded in the network. A similar analysis shows that inference calls can also be shared when calculating the value of $P(c|\mathbf{u})$. Therefore, the total number of inference calls for the i-th feature block is $O(p_i)$. Finally, computing equivalence intervals in the algorithm can be done without any inference calls using the equivalence intervals of subclassifiers. So, the total number of inference calls is $O(P)$.

As for the number of computations of the algorithm, observe that the most expensive operation is finding and storing equivalent subclassifiers in cache, which requires binary search on $O(2^{w_\pi})$ intervals. This gives us $O(w_\pi 2^{w_\pi})$ computations and a time complexity of $O(PT + w_\pi 2^{w_\pi})$. □

Proof of Theorem 4 The bound on the number of inference calls uses the same analysis as Theorem 3. As well, the bound on the number of computations in the algorithm also follows the same analysis, except that we cannot use equivalence intervals so we do not get the same bound on the number of ODD nodes. We only know that there are at most $O(2^s)$ ODD nodes due to the bound from decision tree sizes, so we get a time complexity of $O(PT + 2^s)$. □

References

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebric decision diagrams and their applications. *Formal methods in system design* 10(2-3):171–206.

Breese, J. S., and Heckerman, D. 1996. Decision-theoretic troubleshooting: A framework for repair and experiment. In *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, 124–132. Morgan Kaufmann Publishers Inc.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35:677–691.

Chan, H., and Darwiche, A. 2003. Reasoning about Bayesian network classifiers. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 107–115.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Friedman, N.; Geiger, D.; and Goldszmidt, M. 1997. Bayesian network classifiers. *Machine Learning* 29(2-3):131–163.

Gertner, A. S.; Conati, C.; and VanLehn, K. 1998. Procedural help in andes: Generating hints using a bayesian network student model. *AAAI/IAAI* 1998:106–11.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, 97–117. Springer.

Meinel, C., and Theobald, T. 1998. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer.

Narodytska, N.; Kasiviswanathan, S. P.; Ryzhyk, L.; Sagiv, M.; and Walsh, T. 2018. Verifying properties of binarized deep neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*.

Ng, A. Y., and Jordan, M. I. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems (NIPS)*, 841–848.

Pernkopf, F., and Bilmes, J. 2005. Discriminative versus generative parameter and structure learning of Bayesian network classifiers. In *Proceedings of the 22nd international conference on Machine learning*, 657–664. ACM.

Roos, T.; Wettig, H.; Grünwald, P.; Myllymäki, P.; and Tirri, H. 2005. On discriminative bayesian network classifiers and logistic regression. *Machine Learning* 59(3):267–296.

Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Shih, A.; Choi, A.; and Darwiche, A. 2018a. Formal verification of Bayesian network classifiers. In *Proceedings of the 9th International Conference on Probabilistic Graphical Models (PGM)*.

Shih, A.; Choi, A.; and Darwiche, A. 2018b. A symbolic approach to explaining Bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*.

Wegener, I. 2000. *Branching Programs and Binary Decision Diagrams*. SIAM.