On the Relationship between Sum-Product Networks and Bayesian Networks

Han Zhao HAN.ZHAO@UWATERLOO.CA
Mazen Melibari MMELIBAR@UWATERLOO.CA
Pascal Poupart PPOUPART@UWATERLOO.CA
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Abstract

In this paper, we establish some theoretical connections between Sum-Product Networks (SPNs) and Bayesian Networks (BNs). We prove that every SPN can be converted into a BN in linear time and space in terms of the network size. The key insight is to use Algebraic Decision Diagrams (ADDs) to compactly represent the local conditional probability distributions at each node in the resulting BN by exploiting context-specific independence (CSI). The generated BN has a simple directed bipartite graphical structure. We show that by applying the Variable Elimination algorithm (VE) to the generated BN with ADD representations, we can recover the original SPN where the SPN can be viewed as a history record or caching of the VE inference process. To help state the proof clearly, we introduce the notion of *normal* SPN and present a theoretical analysis of the consistency and decomposability properties. We conclude the paper with some discussion of the implications of the proof and establish a connection between the depth of an SPN and a lower bound of the tree-width of its corresponding BN.

1. Introduction

Sum-Product Networks (SPNs) have recently been proposed as tractable deep models (Poon & Domingos, 2011) for probabilistic inference. They distinguish themselves from other types of probabilistic graphical models (PGMs), including Bayesian Networks (BNs) and Markov Networks (MNs), by the fact that inference can be done exactly in linear time with respect to the size of the network. This has generated a lot of interest since inference is often a core task for parameter estimation and structure learning, and it typically needs to be approximated to ensure tractability since probabilistic inference in BNs and MNs is #P-complete (Roth, 1996).

The relationship between SPNs and BNs, and more broadly with PGMs, is not clear. Since the introduction of SPNs in the seminal paper of Poon & Domingos (2011), it is well understood that SPNs and BNs are equally expressive in the sense that they can represent any joint distribution over discrete variables[1], but it is not clear how to convert SPNs into BNs, nor whether a blow up may occur in the conversion process. The common belief is that there exists a distribution such that the smallest BN that encodes this distribution is exponentially larger than the smallest SPN that encodes the same distribution. The key behind this belief lies in SPNs' ability to exploit context-specific independence (CSI) (Boutilier et al., 1996).

While the above belief is correct for classic BNs with tabular conditional probability distributions (CPDs) that ignore CSI, and for BNs with tree-based CPDs due to the replication problem (Pagallo, 1989), it is not clear whether it is correct for BNs with more compact representations of the CPDs. The other direction is clear for classic BNs with tabular representation: given a BN with tabular representation of its CPDs, we can build an SPN that represents the same joint probability distribution in time and space complexity that may be exponential in the tree-width of the BN. Briefly, this is done by constructing a junction tree and translating it into an SPN[2]. However, to the best of our knowledge, it is still unknown how to convert an SPN into a BN and whether the conversion will lead to a blow up when more compact representations than tables and trees are used for the CPDs.

In this paper, we prove that by adopting Algebraic Decision Diagrams (ADDs) (Bahar et al., 1997) to represent the CPDs at each node in a BN, every SPN can be converted into a BN in linear time and space complexity in the size of the SPN. The generated BN has a simple bipartite structure,

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

[1] Joint distributions over continuous variables are also possible, but we will restrict ourselves to discrete variables in this paper.

[2] http://spn.cs.washington.edu/faq.shtml

which facilitates the analysis of the structure of an SPN in terms of the structure of the generated BN. Furthermore, we show that by applying the Variable Elimination (VE) algorithm (Zhang & Poole, 1996) to the generated BN with ADD representation of its CPDs, we can recover the original SPN in linear time and space with respect to the size of the SPN.

Our contributions can be summarized as follows. First, we present a constructive algorithm and a proof for the conversion of SPNs into BNs using ADDs to represent the local CPDs. The conversion process is bounded by a linear function of the size of the SPN in both time and space. This gives a new perspective to understand the probabilistic semantics implied by the structure of an SPN through the generated BN. Second, we show that by executing VE on the generated BN, we can recover the original SPN in linear time and space complexity in the size of the SPN. Combined with the first point, this establishes a clear relationship between SPNs and BNs. Third, we introduce the subclass of *normal* SPNs and show that every SPN can be transformed into a normal SPN in quadratic time and space. Compared with general SPNs, the structure of normal SPNs exhibit more intuitive probabilistic semantics and hence normal SPNs are used as a bridge in the conversion of general SPNs to BNs.

2. Related Work

The SPN, as an inference machine, has a close connection with the broader field of knowledge representation and knowledge compilation. In knowledge compilation, the reasoning process is divided into two phases: an offline compilation phase and an online query-answering phase. In the offline phase, the knowledge base, either propositional theory or belief network, is compiled into some tractable target language. In the online phase, the compiled target model is used to answer a large number of queries efficiently. The key motivation of knowledge compilation is to shift the computation that is common to many queries from the online phase into the offline phase. As an example, Arithmetic Circuits (ACs) have been studied and used extensively in both knowledge representation and probabilistic inference (Darwiche, 2000; Huang et al., 2006; Chavira et al., 2006). Rooshenas & Lowd (2014) recently showed that ACs and SPNs can be converted mutually without an exponential blow-up in both time and space. As a direct result, ACs and SPNs share the same expressiveness for probabilistic reasoning. Another representation closely related to SPNs in propositional logic and knowledge representation is the deterministic-Decomposable Negation Normal Form (d-DNNF) (Darwiche & Marquis, 2001). Like SPNs, d-DNNF formulas can be queried to answer satisfiability and model counting problems. We refer interested readers to Darwiche & Marquis (2001) and Darwiche (2001) for more detailed discussions.

Since their introduction by Poon & Domingos (2011), SPNs have generated a lot of interest as a tractable class of models for probabilistic inference in machine learning. Discriminative learning techniques for SPNs have been proposed and applied to image classification (Gens & Domingos, 2012). Later, automatic structure learning algorithms were developed to build tree-structured SPNs directly from data (Dennis & Ventura, 2012; Peharz et al., 2013; Gens & Domingos, 2013; Rooshenas & Lowd, 2014). SPNs have also been applied to various fields and have generated promising results, including activity modeling (Amer & Todorovic, 2012), speech modeling (Peharz et al., 2014) and language modeling (Cheng et al., 2014). Theoretical work investigating the influence of the depth of SPNs on expressiveness exists (Delalleau & Bengio, 2011), but is quite limited. As discussed later, our results reinforce previous theoretical results about the depth of SPNs and provide further insights about the structure of SPNs by examining the structure of equivalent BNs.

3. Preliminaries

We start by introducing the notation used in this paper. We use $1 : N$ to abbreviate the notation $\{1, 2, \ldots, N\}$. We use a capital letter X to denote a random variable and a bolded capital letter $\mathbf{X}_{1:N}$ to denote a set of random variables $\mathbf{X}_{1:N} = \{X_1, \ldots, X_N\}$. Similarly, a lowercase letter x is used to denote a value taken by X and a bolded lowercase letter $\mathbf{x}_{1:N}$ denotes a joint value taken by the corresponding vector $\mathbf{X}_{1:N}$ of random variables. We may omit the subscript $1 : N$ from $\mathbf{X}_{1:N}$ and $\mathbf{x}_{1:N}$ if it is clear from the context. For a random variable X_i, we use $x_i^j, j \in 1 : J$ to enumerate all the values taken by X_i. For simplicity, we use $\Pr(x)$ to mean $\Pr(X = x)$ and $\Pr(\mathbf{x})$ to mean $\Pr(\mathbf{X} = \mathbf{x})$. We use calligraphic letters to denote graphs (e.g., $\mathcal{G}$). In particular, BNs, SPNs and ADDs are denoted respectively by $\mathcal{B}$, $\mathcal{S}$ and $\mathcal{A}$.

To ensure that the paper is self contained, we briefly review some background material about Algebraic Decision Diagrams and Sum-Product Networks.

3.1. Algebraic Decision Diagram

We first give a formal definition of Algebraic Decision Diagrams (ADDs) for variables with Boolean domains and then extend the definition to domains corresponding to arbitrary finite sets.

Definition 1 (Algebraic Decision Diagram (Bahar et al., 1997))**.** An Algebraic Decision Diagram (ADD) is a graphical representation of a real function with Boolean input variables: $f : \{0, 1\}^N \mapsto \mathbb{R}$, where the graph is a rooted

DAG. There are two kinds of nodes in an ADD. Terminal nodes, whose out-degree is 0, are associated with real values. Internal nodes, whose out-degree is 2, are associated with Boolean variables $X_n, n \in 1:N$. For each internal node X_n, the left out-edge is labeled with $X_n = \texttt{FALSE}$ and the right out-edge is labeled with $X_n = \texttt{TRUE}$.

We extend the original definition of an ADD by allowing it to represent not only functions of Boolean variables, but also any function of discrete variables with a finite set as domain. This can be done by allowing each internal node X_n to have $|\mathcal{X}_n|$ out-edges and label each edge with $x_n^j, j \in 1:|\mathcal{X}_n|$, where $\mathcal{X}_n$ is the domain of variable X_n and $|\mathcal{X}_n|$ is the number of values X_n takes. Such an ADD represents a function $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_N \mapsto \mathbb{R}$, where $\times$ means the *Cartesian product* between two sets. Henceforth, we will use our extended definition of ADDs throughout the paper.

For our purpose, we will use an ADD as a compact graphical representation of local CPDs associated with each node in a BN. This is a key insight of our constructive proof presented later. Compared with a tabular representation or a decision tree representation of local CPDs, CPDs represented by ADDs can fully exploit CSI (Boutilier et al., 1996) and effectively avoid the replication problem (Pagallo, 1989) of the decision tree representation.

We give an example in Fig. 1 where the tabular representation, decision-tree representation and ADD representation of a function of 4 Boolean variables is presented. Another advantage of ADDs to represent local CPDs is that arithmetic operations such as multiplying ADDs and summing-out a variable from an ADD can be implemented efficiently in polynomial time. This will allow us to use ADDs in the Variable Elimination (VE) algorithm to recover the original SPN after its conversion to a BN with CPDs represented by ADDs. Readers are referred to Bahar et al. (1997) for a more detailed and thorough discussion about ADDs.

3.2. Sum-Product Network

Before introducing SPNs, we first define the notion of *network polynomial*, which plays an important role in our proof. We use $\mathbb{I}[X = x]$ to denote an indicator that returns 1 when $X = x$ and 0 otherwise. To simplify the notation, we will use $\mathbb{I}_x$ to represent $\mathbb{I}[X = x]$.

Definition 2 (Network Polynomial (Poon & Domingos, 2011))**.** Let $f(\cdot) \geq 0$ be an unnormalized probability distribution over a Boolean random vector $\mathbf{X}_{1:N}$. The network polynomial of $f(\cdot)$ is a multilinear function $\sum_{\mathbf{x}} f(\mathbf{x}) \prod_{n=1}^{N} \mathbb{I}_{\mathbf{x}_n}$ of indicator variables, where the summation is over all possible instantiations of the Boolean random vector $\mathbf{X}_{1:N}$.

Intuitively, the network polynomial is a Boolean expansion (Boole, 1847) of the unnormalized probability distribution $f(\cdot)$. For example, the network polynomial of a BN $X_1 \rightarrow X_2$ is $\Pr(x_1, x_2)\mathbb{I}_{x_1}\mathbb{I}_{x_2} + \Pr(x_1, \bar{x}_2)\mathbb{I}_{x_1}\mathbb{I}_{\bar{x}_2} + \Pr(\bar{x}_1, x_2)\mathbb{I}_{\bar{x}_1}\mathbb{I}_{x_2} + \Pr(\bar{x}_1, \bar{x}_2)\mathbb{I}_{\bar{x}_1}\mathbb{I}_{\bar{x}_2}$.

Definition 3 (Sum-Product Network (Poon & Domingos, 2011))**.** A Sum-Product Network (SPN) over Boolean variables $\mathbf{X}_{1:N}$ is a rooted DAG whose leaves are the indicators $\mathbb{I}_{x_1}, \ldots, \mathbb{I}_{x_N}$ and $\mathbb{I}_{\bar{x}_1}, \ldots, \mathbb{I}_{\bar{x}_N}$ and whose internal nodes are sums and products. Each edge (v_i, v_j) emanating from a sum node v_i has a non-negative weight w_{ij}. The value of a product node is the product of the values of its children. The value of a sum node is $\sum_{v_j \in Ch(v_i)} w_{ij} val(v_j)$ where $Ch(v_i)$ are the children of v_i and $val(v_j)$ is the value of node v_j. The value of an SPN $\mathcal{S}[\mathbb{I}_{x_1}, \mathbb{I}_{\bar{x}_1}, \ldots, \mathbb{I}_{x_N}, \mathbb{I}_{\bar{x}_N}]$ is the value of its root.

The *scope* of a node in an SPN is defined as the set of variables that have indicators among the node's descendants: For any node v in an SPN, if v is a terminal node, say, an indicator variable over X, then $\text{scope}(v) = \{X\}$, else $\text{scope}(v) = \bigcup_{\tilde{v} \in Ch(v)} \text{scope}(\tilde{v})$. Poon & Domingos (2011) further define the following properties of an SPN:

Definition 4 (Complete)**.** An SPN is *complete* iff each sum node has children with the same scope.

Definition 5 (Consistent)**.** An SPN is consistent iff no variable appears negated in one child of a product node and non-negated in another.

Definition 6 (Decomposable)**.** An SPN is decomposable iff for every product node v, $\text{scope}(v_i) \bigcap \text{scope}(v_j) = \varnothing$ where $v_i, v_j \in Ch(v), i \neq j$.

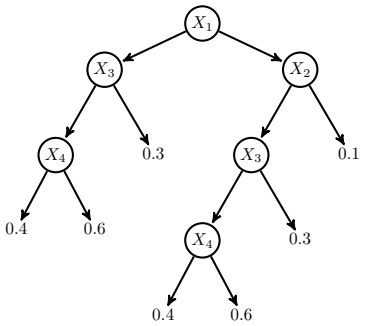
Clearly, decomposability implies consistency in SPNs. An SPN is said to be *valid* iff it defines a (unnormalized) probability distribution. Poon & Domingos (2011) proved that if an SPN is complete and consistent, then it is valid. Note that this is a sufficient, but not necessary condition. In this paper, we focus only on complete and consistent SPNs as we are interested in their associated probabilistic semantics. For a complete and consistent SPN $\mathcal{S}$, each node v in $\mathcal{S}$ defines a network polynomial $f_v(\cdot)$ which corresponds to the sub-SPN rooted at v. The network polynomial defined by the root of the SPN can then be computed recursively by taking a weighted sum of the network polynomials defined by the sub-SPNs rooted at the children of each sum node and a product of the network polynomials defined by the sub-SPNs rooted at the children of each product node. The probability distribution induced by an SPN $\mathcal{S}$ is defined as $\Pr_{\mathcal{S}}(\mathbf{x}) \triangleq \frac{f_{\mathcal{S}}(\mathbf{x})}{\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x})}$, where $f_{\mathcal{S}}(\cdot)$ is the network polynomial defined by the root of the SPN $\mathcal{S}$.

4. Main Results

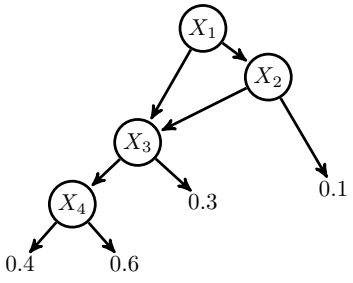
In this section, we state the main results obtained in this paper and then provide a discussion. Due to the space limit,

X_1	X_2	X_3	X_4	$f(\cdot)$	X_1	X_2	X_3	X_4	$f(\cdot)$
0	0	0	0	0.4	1	0	0	0	0.4
0	0	0	1	0.6	1	0	0	1	0.6
0	0	1	0	0.3	1	0	1	0	0.3
0	0	1	1	0.3	1	0	1	1	0.3
0	1	0	0	0.4	1	1	0	0	0.1
0	1	0	1	0.6	1	1	0	1	0.1
0	1	1	0	0.3	1	1	1	0	0.1
0	1	1	1	0.3	1	1	1	1	0.1

(a) Tabular representation.



(b) Decision-Tree representation.



(c) ADD representation.

Figure 1. Different representations of the same Boolean function. The tabular representation cannot exploit CSI and the Decision-Tree representation cannot reuse isomorphic subgraphs. The ADD representation can fully exploit CSI by sharing isomorphic subgraphs, which makes it the most compact representation among the three representations. In Fig. 1(b) and Fig. 1(c), the left and right branches of each internal node correspond respectively to `FALSE` and `TRUE`.

we refer the reader to the supplementary material for all the proof details. To keep the presentation simple, we assume without loss of generality that all the random variables are Boolean unless explicitly stated. It is straightforward to extend our analysis to discrete random variables with finite support. For an SPN $\mathcal{S}$, let $|\mathcal{S}|$ be the size of the SPN, i.e., the number of nodes plus the number of edges in the graph. For a BN $\mathcal{B}$, the size of $\mathcal{B}$, $|\mathcal{B}|$, is defined by the size of the graph *plus* the size of all the CPDs in $\mathcal{B}$ (the size of a CPD depends on its representation, which will be clear from the context). The main theorems are:

Theorem 1. There exists an algorithm that converts any complete and decomposable SPN $\mathcal{S}$ over Boolean variables $\mathbf{X}_{1:N}$ into a BN $\mathcal{B}$ with CPDs represented by ADDs in time $O(N|\mathcal{S}|)$. Furthermore, $\mathcal{S}$ and $\mathcal{B}$ represent the same distribution and $|\mathcal{B}| = O(N|\mathcal{S}|)$.

As it will be clear later, Thm. 1 immediately leads to the following corollary:

Corollary 1. There exists an algorithm that converts any complete and consistent SPN $\mathcal{S}$ over Boolean variables $\mathbf{X}_{1:N}$ into a BN $\mathcal{B}$ with CPDs represented by ADDs in time $O(N|\mathcal{S}|^2)$. Furthermore, $\mathcal{S}$ and $\mathcal{B}$ represent the same distribution and $|\mathcal{B}| = O(N|\mathcal{S}|^2)$.

Remark 1. The BN $\mathcal{B}$ generated from $\mathcal{S}$ in Theorem 1 and Corollary 1 has a simple bipartite DAG structure, where all the source nodes are hidden variables and the terminal nodes are the Boolean variables $\mathbf{X}_{1:N}$.

Remark 2. Assuming sum nodes alternate with product nodes in SPN $\mathcal{S}$, the depth of $\mathcal{S}$ is proportional to the maximum in-degree of the nodes in $\mathcal{B}$, which, as a result, is proportional to a lower bound of the tree-width of $\mathcal{B}$.

Theorem 2. Given the BN $\mathcal{B}$ with ADD representation of CPDs generated from a complete and decomposable SPN $\mathcal{S}$ over Boolean variables $\mathbf{X}_{1:N}$, the original SPN $\mathcal{S}$ can be recovered by applying the Variable Elimination algorithm to $\mathcal{B}$ in $O(N|\mathcal{S}|)$.

Remark 3. The combination of Theorems 1 and 2 shows that distributions for which SPNs allow a compact representation and efficient inference, BNs with ADDs also allow a compact representation and efficient inference (i.e., no exponential blow up).

To make the upcoming proofs concise, we first define a *normal form* for SPNs and show that every complete and consistent SPN can be transformed into a normal SPN in quadratic time and space without changing the network polynomial. We then derive the proofs with normal SPNs. Note that we only focus on SPNs that are *complete* and *consistent*. Hence, when we refer to an SPN, we assume that it is complete and consistent without explicitly stating this.

4.1. Normal Form

For an SPN $\mathcal{S}$, let $f_{\mathcal{S}}(\cdot)$ be the network polynomial defined at the root of $\mathcal{S}$. Define the *height* of an SPN to be the length of the longest path from the root to a terminal node.

Definition 7. An SPN is said to be normal if

1. It is complete and decomposable.
2. For each sum node in the SPN, the weights of the edges emanating from the sum node are nonnegative and sum to 1.
3. Every terminal node in the SPN is a univariate distribution over a Boolean variable and the size of the scope of a sum node is at least 2 (sum nodes whose scope is of size 1 are reduced into terminal nodes).

Theorem 3. For any complete and consistent SPN $\mathcal{S}$, there exists a normal SPN $\mathcal{S}'$ such that $\Pr_{\mathcal{S}}(\cdot) = \Pr_{\mathcal{S}'}(\cdot)$ and $|\mathcal{S}'| = O(|\mathcal{S}|^2)$.

An example of a normal SPN constructed from a general SPN is depicted in Fig. 2.

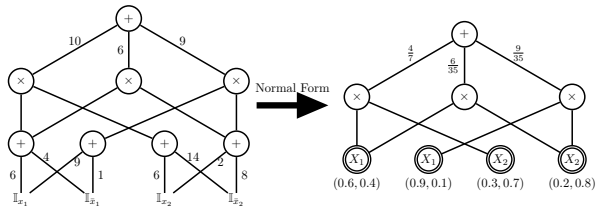


Figure 2. Transform an SPN into a normal form. Terminal nodes which are probability distributions over a single variable are represented by a double-circle.

4.2. SPN to BN

In order to construct a BN from an SPN, we require the SPN to be in a normal form, otherwise we can first transform it into a normal form by Thm. 3 using the algorithms provided in the supplementary material.

Let $\mathcal{S}$ be a normal SPN over $\mathbf{X}_{1:N}$. Before showing how to construct a corresponding BN, we first give some intuitions. One useful view is to associate each sum node in an SPN with a hidden variable. For example, consider a sum node $v \in \mathcal{S}$ with out-degree l. Since $\mathcal{S}$ is normal, we have $\sum_{i=1}^{l} w_i = 1$ and $w_i \geq 0, \forall i \in 1 : l$. This naturally suggests that we can associate a hidden discrete random variable H_v with multinomial distribution $\Pr_v(H_v = i) = w_i, i \in 1 : l$ for each sum node $v \in \mathcal{S}$. Therefore, $\mathcal{S}$ can be thought as defining a joint probability distribution over $\mathbf{X}_{1:N}$ and $\mathbf{H} = \{H_v \mid v \in \mathcal{S}, v \text{ is a sum node}\}$ where $\mathbf{X}_{1:N}$ are the observable variables and $\mathbf{H}$ are the hidden variables. When doing inference with an SPN, we implicitly sum out all the hidden variables $\mathbf{H}$ and compute $\Pr_{\mathcal{S}}(\mathbf{x}) = \sum_{\mathbf{h}} \Pr_{\mathcal{S}}(\mathbf{x}, \mathbf{h})$. Associating each sum node in an SPN with a hidden variable not only gives us a conceptual understanding of the probability distribution defined by an SPN, but also helps to elucidate one of the key properties implied by the structure of an SPN as summarized below:

Proposition 1. Given a normal SPN $\mathcal{S}$, let p be a product node in $\mathcal{S}$ with l children. Let $v_1, \ldots, v_k$ be sum nodes which lie on a path from the root of $\mathcal{S}$ to p. Then

$$\Pr_{\mathcal{S}}(\mathbf{x}|_{\text{scope}(p)} \mid H_{v_1} = v_1^*, \ldots, H_{v_k} = v_k^*) = \prod_{i=1}^{l} \Pr_{\mathcal{S}}(\mathbf{x}|_{\text{scope}(p_i)} \mid H_{v_1} = v_1^*, \ldots, H_{v_k} = v_k^*) \quad (1)$$

where $H_v = v^*$ means the sum node v selects its v^*th branch and $\mathbf{x}|_A$ denotes restricting $\mathbf{x}$ by set A, p_i is the ith child of product node p.

Note that there may exist multiple paths from the root to p in $\mathcal{S}$. Each such path admits the factorization stated in Eq. 1. Eq. 1 explains two key insights implied by the structure of an SPN that will allow us to construct an equivalent BN with ADDs. First, CSI is efficiently encoded by the structure of an SPN using Proposition 1. Second, the DAG structure of an SPN allows multiple assignments of hidden variables to share the same factorization, which effectively avoids the replication problem present in decision trees.

Based on the observations above and with the help of the normal form for SPNs, we now proceed to prove the first main result in this paper: Thm. 1. First, we present the algorithm to construct the structure of a BN $\mathcal{B}$ from $\mathcal{S}$ in Alg. 1. In a nutshell, Alg. 1 creates an observable variable

Algorithm 1 Build BN Structure

Input: normal SPN $\mathcal{S}$
Output: BN $\mathcal{B} = (\mathcal{B}_V, \mathcal{B}_E)$

```
1:  R ← root of S
2:  if R is a terminal node over variable X then
3:      Create an observable variable X
4:      B_V ← B_V ∪ {X}
5:  else
6:      for each child R_i of R do
7:          if BN has not been built for S_{R_i} then
8:              Recursively build BN Structure for S_{R_i}
9:          end if
10:     end for
11:     if R is a sum node then
12:         Create a hidden variable H_R associated with R
13:         B_V ← B_V ∪ {H_R}
14:         for each observable variable X ∈ S_R do
15:             B_E ← B_E ∪ {(H_R, X)}
16:         end for
17:     end if
18: end if
```

X in $\mathcal{B}$ for each terminal node over X in $\mathcal{S}$ (Lines 2-4). For each internal sum node v in $\mathcal{S}$, Alg. 1 creates a hidden variable H_v associated with v and builds directed edges from H_v to all observable variables X appearing in the sub-SPN rooted at v (Lines 11-17). *The BN $\mathcal{B}$ created by Alg. 1 has a directed bipartite structure with a layer of hidden variables pointing to a layer of observable variables*. A hidden variable H points to an observable variable X in $\mathcal{B}$ iff X appears in the sub-SPN rooted at H in $\mathcal{S}$.

We now present Alg. 2 and 3 to build ADDs for each observable variable X and hidden variable H in $\mathcal{B}$. For each hidden variable H, Alg. 3 builds $\mathcal{A}_H$ as a decision stump[3] obtained by finding H and its associated weights in $\mathcal{S}$. Consider ADDs built by Alg. 2 for observable variables Xs. Let X be the current observable variable we are considering. Basically, Alg. 2 is a recursive algorithm applied to each node in $\mathcal{S}$ whose scope intersects with $\{X\}$. There

[3] A decision stump is a decision tree with one variable.

Algorithm 2 Build CPD using ADD, observable variable

Input: normal SPN $\mathcal{S}$, variable X
Output: ADD $\mathcal{A}_X$
1: **if** ADD has already been created for $\mathcal{S}$ and X **then**
2: $\mathcal{A}_X \leftarrow$ retrieve ADD from cache
3: **else**
4: $R \leftarrow$ root of $\mathcal{S}$
5: **if** R is a terminal node **then**
6: $\mathcal{A}_X \leftarrow$ decision stump rooted at R
7: **else if** R is a sum node **then**
8: Create a node H_R into $\mathcal{A}_X$
9: **for** each $R_i \in Ch(R)$ **do**
10: Link BuildADD$(\mathcal{S}_{R_i}, X)$ as ith child of H_R
11: **end for**
12: **else if** R is a product node **then**
13: Find child $\mathcal{S}_{R_i}$ such that $X \in \text{scope}(R_i)$
14: $\mathcal{A}_X \leftarrow$ BuildADD$(\mathcal{S}_{R_i}, X)$
15: **end if**
16: store $\mathcal{A}_X$ in cache
17: **end if**

Algorithm 3 Build CPD using ADD, hidden variable

Input: normal SPN $\mathcal{S}$, variable H
Output: ADD $\mathcal{A}_H$
1: Find the sum node H in $\mathcal{S}$
2: $\mathcal{A}_H \leftarrow$ decision stump rooted at H in $\mathcal{S}$

are three cases. If the current node is a terminal node, then it must be a probability distribution over X. In this case we simply return the decision stump at the current node. If the current node is a sum node, then due to the completeness of $\mathcal{S}$, we know that all the children of R share the same scope with R. We first create a node H_R corresponding to the hidden variable associated with R into $\mathcal{A}_X$ (Line 8) and recursively apply Alg. 2 to all the children of R and link them to H_R respectively. If the current node is a product node, then due to the decomposability of $\mathcal{S}$, we know that there will be a unique child of R whose scope intersects with $\{X\}$. We recursively apply Alg. 2 to this child and return the resulting ADD (Lines 12-15).

Equivalently, Alg. 2 can be understood in the following way: *we extract the sub-SPN induced by* $\{X\}$ *and contract*[4] *all the product nodes in it to obtain* $\mathcal{A}_X$. Note that the *contraction* of product nodes will not add more edges into $\mathcal{A}_X$ since the out-degree of each product node in the induced sub-SPN must be 1 due to the decomposability of the product node. We illustrate the application of Alg. 1, 2 and 3 on the normal SPN in Fig. 2, which results in the BN $\mathcal{B}$ with CPDs represented by ADDs shown in Fig. 3.

Theorem 4. For any normal SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$, the BN $\mathcal{B}$ constructed by Alg. 1, 2 and 3 encodes the same probability distribution, i.e., $\Pr_{\mathcal{S}}(\mathbf{x}) = \Pr_{\mathcal{B}}(\mathbf{x}), \forall \mathbf{x}$.

[4]In graph theory, the contraction of a node v in a DAG is the operation that connects each parent of v to each child of v and then delete v from the graph.

The proof is by induction on the height of $\mathcal{S}$ (see the supplementary material for more details).

Theorem 5. $|\mathcal{B}| = O(N|\mathcal{S}|)$, where BN $\mathcal{B}$ is constructed by Alg. 1, 2 and 3 from normal SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$.

Theorem 6. For any normal SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$, Alg. 1, 2 and 3 construct an equivalent BN in time $O(N|\mathcal{S}|)$.

4.3. BN to SPN

Algorithm 4 Multiplication of two symbolic ADDs, $\otimes$

Input: Symbolic ADD $\mathcal{A}_{X_1}, \mathcal{A}_{X_2}$
Output: Symbolic ADD $\mathcal{A}_{X_1,X_2} = \mathcal{A}_{X_1} \otimes \mathcal{A}_{X_2}$
1: $R_1 \leftarrow$ root of $\mathcal{A}_{X_1}$, $R_2 \leftarrow$ root of $\mathcal{A}_{X_2}$
2: **if** R_1 and R_2 are both variable nodes **then**
3: **if** $R_1 = R_2$ **then**
4: Create a node $R = R_1$ into $\mathcal{A}_{X_1,X_2}$
5: **for** each $r \in dom(R)$ **do**
6: $\mathcal{A}^r_{X_1} \leftarrow Ch(R_1)|_r$
7: $\mathcal{A}^r_{X_2} \leftarrow Ch(R_2)|_r$
8: $\mathcal{A}^r_{X_1,X_2} \leftarrow \mathcal{A}^r_{X_1} \otimes \mathcal{A}^r_{X_2}$
9: Link $\mathcal{A}^r_{X_1,X_2}$ to the rth child of R in $\mathcal{A}_{X_1,X_2}$
10: **end for**
11: **else**
12: $\mathcal{A}_{X_1,X_2} \leftarrow$ create a symbolic node $\otimes$
13: Link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as two children of $\otimes$
14: **end if**
15: **else if** R_1 is a variable node and R_2 is $\otimes$ **then**
16: **if** R_1 appears as a child of R_2 **then**
17: $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2}$
18: $\mathcal{A}^{R_1}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1} \otimes \mathcal{A}^{R_1}_{X_2}$
19: **else**
20: Link $\mathcal{A}_{X_1}$ as a new child of R_2
21: $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2}$
22: **end if**
23: **else if** R_1 is $\otimes$ and R_2 is a variable node **then**
24: **if** R_2 appears as a child of R_1 **then**
25: $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1}$
26: $\mathcal{A}^{R_2}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2} \otimes \mathcal{A}^{R_2}_{X_1}$
27: **else**
28: Link $\mathcal{A}_{X_2}$ as a new child of R_1
29: $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1}$
30: **end if**
31: **else**
32: $\mathcal{A}_{X_1,X_2} \leftarrow$ create a symbolic node $\otimes$
33: Link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as two children of $\otimes$
34: **end if**
35: Merge connected product nodes in $\mathcal{A}_{X_1,X_2}$

It is known that a BN with CPDs represented by tables can be converted into an SPN by first converting the BN into a junction tree and then translating the junction tree into an SPN. The size of the generated SPN, however, will be exponential in the tree-width of the original BN since the tabular representation of CPDs is ignorant of CSI. Hence, the generated SPN loses its power to compactly represent some BNs with high tree-width, yet, with CSI in its CPDs.

In this section, we focus on BNs with ADDs that are constructed using Alg. 2 and 3 from normal SPNs. We show

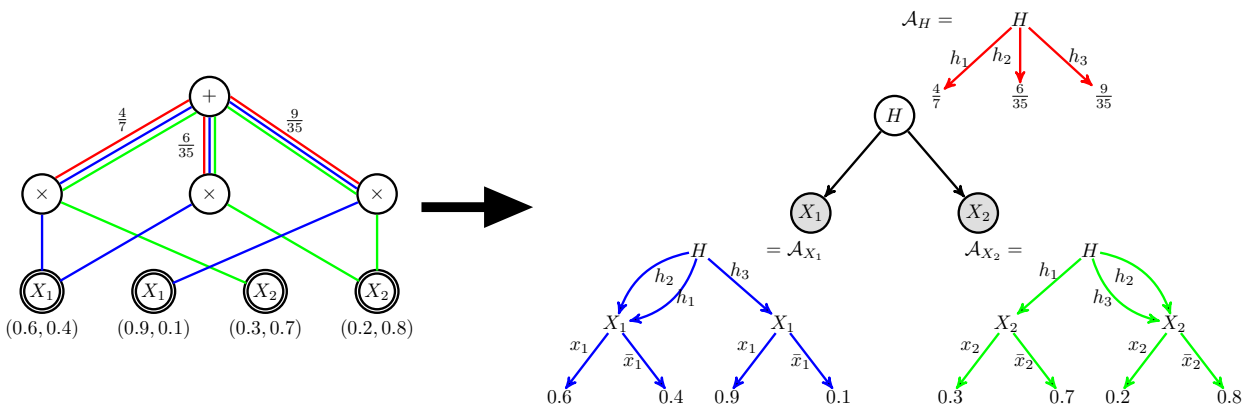


Figure 3. Construct a BN with CPDs represented by ADDs from an SPN. On the left, the induced sub-SPNs used to create $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ by Alg. 2 are indicated in blue and green respectively. The decision stump used to create $\mathcal{A}_H$ by Alg. 3 is indicated in red.

that when applying VE to those BNs with ADDs we can recover the original normal SPNs. *The key insight is that the structure of the original normal SPN naturally defines a global variable ordering that is consistent with the topological ordering of every ADD constructed.*

In order to apply VE to a BN with ADDs, we need to show how to apply two common operations used in VE, i.e., multiplication of two factors and summing-out a hidden variable, on ADDs. For our purpose, we use a *symbolic ADD* as an intermediate representation during the inference process of VE by allowing symbolic operations, such as $+, -, \times, /$ to appear as internal nodes in ADDs. In this sense, an ADD can be viewed as a special type of symbolic ADD where all the internal nodes are variables. The same trick was applied by (Chavira & Darwiche, 2007) in their compilation approach. For example, given symbolic ADDs $\mathcal{A}_{X_1}$ over X_1 and $\mathcal{A}_{X_2}$ over X_2, the multiplication of those ADDs returns a new symbolic ADD $\mathcal{A}_{X_1,X_2}$ over X_1, X_2 such that $\mathcal{A}_{X_1,X_2}(x_1, x_2) \triangleq (\mathcal{A}_{X_1} \otimes \mathcal{A}_{X_2})(x_1, x_2) = \mathcal{A}_{X_1}(x_1) \times \mathcal{A}_{X_2}(x_2)$. To simplify the presentation, we choose the inverse topological ordering of the hidden variables in the original SPN $\mathcal{S}$ as the elimination order used in VE. This helps to avoid the situations where a multiplication is applied to a sum node in symbolic ADDs. Other elimination orders could be used, but a more detailed discussion of sum nodes is needed. The algorithm for multiplying two symbolic ADDs is listed in Alg. 4. It is important to point out here that the time complexity of multiplication between two symbolic ADDs is $O(|\mathcal{S}|)$, i.e., bounded by the size of the original SPN $\mathcal{S}$ since all the ADDs constructed by Alg. 2 are induced sub-SPNs with contraction of product nodes from the original SPN $\mathcal{S}$. Please refer to the supplementary material for a more detailed analysis of the linear complexity of Alg. 4.

To sum-out one hidden variable H, we simply replace H in $\mathcal{A}$ by a symbolic sum node $\oplus$ and label each edge of $\oplus$ with weights obtained from $\mathcal{A}_H$. The algorithm to implement the summing-out operation in a symbolic ADD is presented in Alg. 5. Note that Alg. 4 and 5 apply only to ADDs constructed from normal SPNs by Alg. 2 and 3 because such ADDs naturally inherit the topological ordering of sum nodes (hidden variables) in the original SPN $\mathcal{S}$. Otherwise we need to pre-define a global variable ordering of all the sum nodes and then arrange each ADD such that its topological ordering is consistent with the predefined ordering. Note also that Alg. 4 and 5 should be implemented with caching of repeated operations in order to ensure that directed acyclic graphs are preserved.

We now present the Variable Elimination (VE) algorithm in Alg. 6 used to recover the original SPN $\mathcal{S}$, taking multiplication and summing-out as two operations $\otimes$ and $\oplus$ respectively. In each iteration of Alg. 6, we select one hidden variable H in ordering π, multiply all the symbolic ADDs $\mathcal{A}_X$ in which H appears and then sum-out H from the new ADD. The algorithm keeps going until all the hidden variables have been summed out and there is only one symbolic ADD left in Φ. The final symbolic ADD gives us the SPN $\mathcal{S}$ which can be used to build BN $\mathcal{B}$. Note that the SPN returned by Alg. 6 may not be literally equal to the original SPN since during the multiplication of two symbolic ADDs we effectively remove redundant nodes by merging connected product nodes. Hence, the SPN returned by Alg. 6 could have a smaller size while representing the same probability distribution. An example is given in Fig. 4 to illustrate the recovery process. The BN in Fig. 4 is the one constructed in Fig. 3.

Theorem 7. Alg. 6 builds SPN $\mathcal{S}$ from BN $\mathcal{B}$ with ADDs in $O(N|\mathcal{S}|)$.

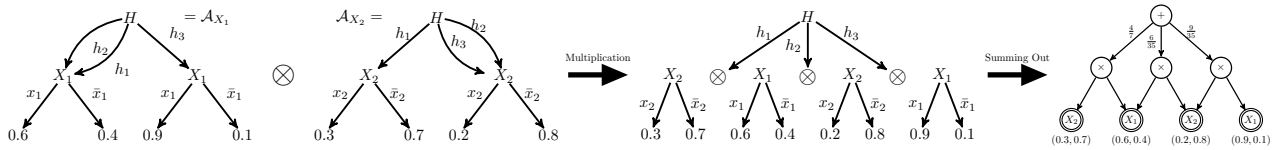


Figure 4. Multiply $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ that contain H and then sum out H. The final SPN is isomorphic with the SPN in Fig. 3.

Algorithm 5 Summing-out a hidden variable H from $\mathcal{A}$ using $\mathcal{A}_H$, $\oplus$

Input: Symbolic ADDs $\mathcal{A}$ and $\mathcal{A}_H$
Output: Symbolic ADD with H summed out
1: **if** H appears in $\mathcal{A}$ **then**
2: Label each edge emanating from H with weights from $\mathcal{A}_H$
3: Replace H by a symbolic $\oplus$ node
4: **end if**

Algorithm 6 Variable Elimination for a BN with ADDs

Input: BN $\mathcal{B}$ with ADDs for all observable and hidden variables
Output: Original SPN $\mathcal{S}$
1: $\pi \leftarrow$ inverse topological order of hidden variables in ADDs
2: $\Phi \leftarrow \{\mathcal{A}_X \mid X$ is an observable variable$\}$
3: **for** each hidden variable H in π **do**
4: $P \leftarrow \{\mathcal{A}_X \mid H$ appears in $\mathcal{A}_X\}$
5: $\Phi \leftarrow \Phi \backslash P \cup \{\oplus_H \otimes_{\mathcal{A} \in P} \mathcal{A}\}$
6: **end for**
7: **return** Φ

5. Discussion and Conclusion

Thm. 1 together with Thm. 2 establish a relationship between BNs and SPNs: SPNs are no more powerful than BNs with ADD representation. Informally, a model is considered to be more powerful than another one if there exists a distribution that can be encoded in polynomial size in some input parameter N, while the other model requires exponential size in N to represent the same distribution. The key is to recognize that the CSI encoded by the structure of an SPN as stated in Proposition 1 can also be encoded explicitly with ADDs in a BN. We can also view an SPN as an inference machine that efficiently records the history of the inference process when applied to a BN. Based on this perspective, an SPN is actually storing the calculations to be performed (sums and products), which allows online inference queries to be answered quicky. The same idea also exists in other fields, including propositional logic (d-DNNF) and knowledge compilation (AC).

The constructed BN has a simple bipartite structure, no matter how deep the original SPN is. However, we can relate the depth of an SPN to a lower bound on the tree-width of the corresponding BN obtained by our algorithm. Without loss of generality, let's assume that product layers alternate with sum layers in the SPN we are considering. Let the height of the SPN, i.e., the longest path from the root to a terminal node, be K. By our assumption, there will be at least $\lfloor K/2 \rfloor$ sum nodes in the longest path. Accordingly, in the BN constructed by Alg. 1, the observable variable corresponding to the terminal node in the longest path will have in-degree at least $\lfloor K/2 \rfloor$. Hence, after moralizing the BN into an undirected graph, the clique-size of the moral graph is bounded below by $\lfloor K/2 \rfloor + 1$. Note that for any undirected graph the clique-size minus 1 is always a lower bound of the tree-width. We then reach the conclusion that the tree-width of the constructed BN has a lower bound of $\lfloor K/2 \rfloor$. In other words, *the deeper the SPN, the larger the tree-width of the BN constructed by our algorithm and the more complex are the probability distributions that can be encoded*. This observation is consistent with the conclusion drawn in (Delalleau & Bengio, 2011) where the authors prove that there exist families of distributions that can be represented much more efficiently with a deep SPN than with a shallow one, i.e. with substantially fewer hidden internal sum nodes. Note that we only give a proof that there exists an algorithm that can convert an SPN into a BN without any exponential blow-up. There may exist other techniques to convert an SPN into a BN with a more compact representation and also a smaller tree-width.

High tree-width is usually used to indicate a high inference complexity, but this is not always true as there may exist lots of CSI between variables, which can reduce inference complexity. CSI is precisely what enables SPNs and BNs with ADDs to compactly represent and tractably perform inference in distributions with high tree-width. In contrast, in a Restricted Boltzmann Machine, which is an undirected bipartite Markov network, CSI may not be present or not exploited, which is why practitioners have to resort to approximate algorithms, such as contrastive divergence (Carreira-Perpinan & Hinton, 2005). Similarly, approximate inference is required in bipartite diagnostic BNs such as the Quick Medical Reference network (Shwe et al., 1991) since causal independence is insufficient to reduce the complexity, while CSI is not present or not exploited.

Although we mainly focus on the relationship between SPNs and BNs, our analysis can be straightforwardly applied to the relationship between SPNs and MNs. The insight lies in the fact that we can always moralize a BN into an MN and an ADD can also be used to encode potential functions defined by cliques in MNs. Our work also provides a new direction for future research about SPNs and BNs. Structure and parameter learning algorithms for SPNs can now be used to indirectly learn BNs with ADDs.

References

Amer, Mohamed R and Todorovic, Sinisa. Sum-product networks for modeling activities with stochastic structure. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1314–1321. IEEE, 2012.

Bahar, R Iris, Frohm, Erica A, Gaona, Charles M, Hachtel, Gary D, Macii, Enrico, Pardo, Abelardo, and Somenzi, Fabio. Algebraic decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206, 1997.

Boole, George. *The mathematical analysis of logic*. Philosophical Library, 1847.

Boutilier, Craig, Friedman, Nir, Goldszmidt, Moises, and Koller, Daphne. Context-specific independence in bayesian networks. In *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, pp. 115–123. Morgan Kaufmann Publishers Inc., 1996.

Carreira-Perpinan, Miguel A and Hinton, Geoffrey E. On contrastive divergence learning. In *Proceedings of the tenth international workshop on artificial intelligence and statistics*, pp. 33–40. Citeseer, 2005.

Chavira, Mark and Darwiche, Adnan. Compiling bayesian networks using variable elimination. In *IJCAI*, pp. 2443–2449, 2007.

Chavira, Mark, Darwiche, Adnan, and Jaeger, Manfred. Compiling relational bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1):4–20, 2006.

Cheng, Wei-Chen, Kok, Stanley, Pham, Hoai Vu, Chieu, Hai Leong, and Chai, Kian Ming A. Language modeling with sum-product networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Darwiche, Adnan. A differential approach to inference in bayesian networks. In *UAI*, pp. 123–132, 2000.

Darwiche, Adnan. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.

Darwiche, Adnan and Marquis, Pierre. A perspective on knowledge compilation. In *IJCAI*, volume 1, pp. 175–182. Citeseer, 2001.

Delalleau, Olivier and Bengio, Yoshua. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pp. 666–674, 2011.

Dennis, Aaron and Ventura, Dan. Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, pp. 2042–2050, 2012.

Gens, Robert and Domingos, Pedro. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pp. 3248–3256, 2012.

Gens, Robert and Domingos, Pedro. Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 873–880, 2013.

Huang, Jinbo, Chavira, Mark, and Darwiche, Adnan. Solving map exactly by searching on compiled arithmetic circuits. In *AAAI*, volume 6, pp. 3–7, 2006.

Pagallo, Giulia. Learning DNF by decision trees. In *IJCAI*, volume 89, pp. 639–644, 1989.

Peharz, Robert, Geiger, Bernhard C, and Pernkopf, Franz. Greedy part-wise learning of sum-product networks. In *Machine Learning and Knowledge Discovery in Databases*, pp. 612–627. Springer, 2013.

Peharz, Robert, Kapeller, Georg, Mowlaee, Pejman, and Pernkopf, Franz. Modeling speech with sum-product networks: Application to bandwidth extension. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 3699–3703. IEEE, 2014.

Poon, Hoifung and Domingos, Pedro. Sum-product networks: A new deep architecture. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pp. 2551–2558, 2011.

Rooshenas, Amirmohammad and Lowd, Daniel. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of The 31st International Conference on Machine Learning*, pp. 710–718, 2014.

Roth, Dan. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996.

Shwe, Michael A, Middleton, B, Heckerman, DE, Henrion, M, Horvitz, EJ, Lehmann, HP, and Cooper, GF. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base. *Methods of information in Medicine*, 30(4):241–255, 1991.

Zhang, Nevin Lianwen and Poole, David. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.