On the Relationship between Sum-Product Networks and Bayesian Networks

by

Han Zhao

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Sum-Product Networks (SPNs), which are probabilistic inference machines, have attracted a lot of interests in recent years. They have a wide range of applications, including but not limited to activity modeling, language modeling and speech modeling. Despite their practical applications and popularity, little research has been done in understanding what is the connection and difference between Sum-Product Networks and traditional graphical models, including Bayesian Networks (BNs) and Markov Networks (MNs). In this thesis, I establish some theoretical connections between Sum-Product Networks and Bayesian Networks. First, I prove that every SPN can be converted into a BN in linear time and space in terms of the network size. Second, I show that by applying the Variable Elimination algorithm (VE) to the generated BN, I can recover the original SPN.

In the first direction, I 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. I establish the first connection between the depth of SPNs and the tree-width of the generated BNs, showing that the depth of SPNs is proportional to a lower bound of the tree-width of the BN.

In the other direction, I show that by applying the Variable Elimination algorithm (VE) to the generated BN with ADD representations, I 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, I introduce the notion of *normal* SPN and present a theoretical analysis of the consistency and decomposability properties. I provide constructive algorithms to transform any given SPN into its normal form in time and space quadratic in the size of the SPN. Combining the above two directions gives us a deep understanding about the modeling power of SPNs and their inner working mechanism.

Acknowledgements

First of all, I would like to give my deepest and the most sincere thanks to my supervisor, Professor Pascal Poupart, for his broad knowledge, his endless support and guidance, his patience and encouragement, and his dedication to work days and nights. Needless to say, this thesis would not have been possible without his help and dedication.

I would also like to thank my committee members, Professor Jesse Hoey and Professor Kate Larson, for the time and energy they devoted to reading this thesis and to providing comments and feedback.

Many thanks to my friends and colleagues at Waterloo who gave me invaluable suggestions during my graduate studies. Thanks to all my friends from Tsinghua University who helped me in my life at Waterloo. Finally, I would like to thank my grandfather, Changyi, my parents, Wei and Yuxia and my brother, Rui for their unconditional love and support throughout the years.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Sum-Product Networks (SPNs) have recently been proposed as tractable deep models [19] 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 [22].

The relationship between SPNs and BNs, and more broadly with PGMs, is not clear. Since the introduction of SPNs in the seminal paper of [19], 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 this same distribution. The key behind this belief lies in SPNs' ability to exploit context-specific independence (CSI) [4].

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 [14], 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, I can build an SPN that represents the same joint probability distribution in time and space complexity that may be exponential in the tree-width

[1] Joint distributions over continuous variables are also possible, but I will restrict myself to discrete variables in this thesis.

of the BN. Briefly, this is done by first constructing a junction tree and then translating it into an SPN[2]. However, to the best of my 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.

I prove in this thesis that by adopting Algebraic Decision Diagrams (ADDs) [2] 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, which facilitates the analysis of the structure of an SPN in terms of the structure of the generated BN. Furthermore, I show that by applying the Variable Elimination (VE) algorithm [26] to the generated BN with ADD representation of its CPDs, I can recover the original SPN in linear time and space with respect to the size of the SPN. Hence this thesis clarifies the relationship between SPNs and BNs as follows: 1), Given an SPN, there exists a BN with ADD representation of CPDs, whose size is bounded by a linear factor of the original SPN, that represents the same joint probability distribution. 2), Given a BN with ADD representation of CPDs constructed from an SPN, I can apply VE to recover the original SPN in linear time and space complexity in the size of the original SPN.

1.1 Contributions

My contributions can be summarized as follows. First, I 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, I show that by executing VE on the generated BN, I 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, I 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. Fourth, my construction and analysis provides a new direction for learning the parameter/structure of BNs since the SPNs produced by the algorithms that learn SPNs [8, 10, 16, 21] can be converted into BNs.

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

1.2 Roadmap

The rest of the thesis is structured as follows. In Chapter 2 I will first introduce some background knowledge, including the notion of conditional independence and context-specific independence. I will also give a short introduction to Bayesian networks, Markov networks, algebraic decision diagrams and Sum-Product networks. I present my main results in Chapter 3, which includes constructive algorithms and proofs of transformations between Bayesian networks and Sum-Product networks. I make some remarks and discuss some implications of my main theorems proved in Chapter 4. Chapter 5 concludes the thesis.

Chapter 2

Preliminaries

I start by introducing the notation used in this thesis. I use $1 : N$ to abbreviate the notation $\{1, 2, \ldots, N\}$. I use a capital letter X to denote a random variable and a bold 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 bold lowercase letter $\mathbf{x}_{1:N}$ denotes a joint value taken by the corresponding vector $\mathbf{X}_{1:N}$ of random variables. I 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, I use $x_i^j, j \in 1 : J$ to enumerate all the values taken by X_i. For simplicity, I use $\Pr(x)$ to mean $\Pr(X = x)$ and $\Pr(\mathbf{x})$ to mean $\Pr(\mathbf{X} = \mathbf{x})$. I 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 $\mathcal{A}$. For a DAG $\mathcal{G}$ and a node v in $\mathcal{G}$, I use $\mathcal{G}_v$ to denote the subgraph of $\mathcal{G}$ induced by v and all its descendants. Let $\mathbf{V}$ be a subset of the nodes of $\mathcal{G}$, then $\mathcal{G}|_{\mathbf{V}}$ is a subgraph of $\mathcal{G}$ induced by the node set $\mathbf{V}$. Similarly, I use $\mathbf{X}|_A$ or $\mathbf{x}|_A$ to denote the restriction of a vector to a subset A. I use node and vertex, arc and edge interchangeably when I refer to a graph. Other notation will be introduced when needed.

To ensure that the thesis is self contained, I briefly review some background materials about the notion of conditional independence/context-specific independence, probabilistic graphical models, including both Bayesian Networks and Markov Networks, Algebraic Decision Diagrams and Sum-Product Networks.

2.1 Conditional Independence

In this section I first review some background knowledge about *conditional independence* and *context-specific independence*. Both of these two notions will play a central role in my discussion

afterwards. Let X, Y and Z be three random variables. I say that X is independent of Y, denoted by $X \perp\!\!\!\perp Y$, if and only if

$$\forall x, y \quad \Pr(x, y) = \Pr(x) \Pr(y) \tag{2.1}$$

Similarly, I say that X is conditionally independent of Y given Z, denoted by $X \perp\!\!\!\perp Y|Z$, if and only if

$$\forall x, y, z \quad \Pr(x, y|z) = \Pr(x|z) \Pr(y|z) \tag{2.2}$$

provided that all the conditional distributions $\Pr(X|Z)$, $\Pr(Y|Z)$ and $\Pr(X, Y|Z)$ are well-defined for all the possible values of $Z = z$. I proceed to define the notion of *context-specific independence* (CSI) as follows: X is said to be independent of Y given $Z = z$, denoted by $X \perp\!\!\!\perp Y|Z = z$, if and only if

$$\exists z \, \forall x, y, \quad \Pr(x, y|z) = \Pr(x|z) \Pr(y|z) \tag{2.3}$$

In other words, there exists a value $Z = z$, such that under the context of $Z = z$, the conditional distribution $\Pr(X, Y|Z = z)$ factorizes into $\Pr(X|Z = z)$ and $\Pr(Y|Z = z)$. Note that conditional independence induces context-specific independence. As will be clear later, in probabilistic graphical models, the conditional independence property among the random variables is encoded by the graphical structure, while the context-specific independence can only be exploited by specific data structures.

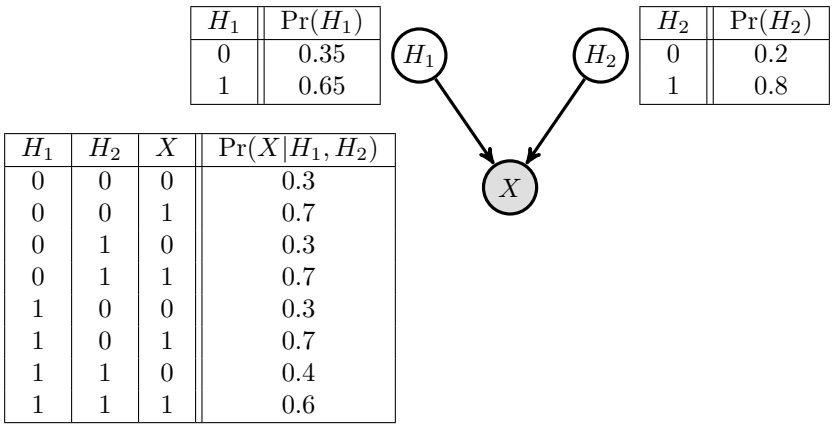
2.2 Probabilistic Graphical Models

Probabilistic graphical models are an elegant framework which combines both graph theory and probability theory to compactly model complex, real-world phenomena. Many commonly used statistical models can be categorized as probabilistic graphical models, including Naive-Bayes, Hidden Markov Models, Kalman Filters, Ising models and so on. In a probabilistic graphical model, each node represents a random variable and the links between them express the probabilistic relationship between the two random variables.

Generally, there are two kinds of probabilistic graphical models, depending on whether the graph is directed or not. The first kind of graphical models is the class of Bayesian networks, also known as directed graphical models or belief networks. The other major class of graphical models is called Markov networks, also known as undirected graphical models or Markov random fields. Directed graphical models are useful to express causal relationships between variables whereas undirected graphical models are often used to express the correlations among random variables. In this thesis I focus more on the the relationship between Sum-Product networks and Bayesian networks. However, my conclusion can also be extended to Markov networks. I will briefly discuss how to convert an SPN into an MN in the discussion section.

2.2.1 Bayesian Networks

Consider a problem whose domain is characterized by a set of random variables $\mathbf{X}_{1:N}$ with finite support. The joint probability distribution over $\mathbf{X}_{1:N}$ can be characterized by a *Bayesian Network*, which is a DAG where nodes represent the random variables and edges represent probabilistic dependencies among the variables. In a BN, I also use the terms "node" and "variable" interchangeably. For each variable in a BN, there is a local conditional probability distribution (CPD) over the variable given its parents in the BN. I give an example of a BN over three variables H_1, H_2 and X in Fig. 2.1.



H_1	$\Pr(H_1)$
0	0.35
1	0.65

H_2	$\Pr(H_2)$
0	0.2
1	0.8

H_1	H_2	X	$\Pr(X\|H_1,H_2)$
0	0	0	0.3
0	0	1	0.7
0	1	0	0.3
0	1	1	0.7
1	0	0	0.3
1	0	1	0.7
1	1	0	0.4
1	1	1	0.6

Figure 2.1: An example of Bayesian network over three variables.

The structure of a BN encodes conditional independencies among the variables in it. Let $X_1, X_2, \ldots, X_N$ be a topological ordering of all the nodes in a BN[1], and let π_{X_i} be the set of parents of node X_i in the BN. Each variable in a BN is conditionally independent of all its non-descendants given its parents. Hence, the joint probability distribution over $\mathbf{X}_{1:N}$ admits the factorization in Eq. 2.4.

$$\Pr(\mathbf{X}_{1:N}) = \prod_{i=1}^{N} \Pr(X_i \mid \mathbf{X}_{1:i-1}) = \prod_{i=1}^{N} \Pr(X_i \mid \pi_{X_i}) \tag{2.4}$$

For the example given in Fig. 2.1, the joint probability distribution factorizes as $\Pr(H_1, H_2, X) = \Pr(H_1) \times \Pr(H_2) \times \Pr(X|H_1, H_2)$. A general procedure called d-separation [15] could be used to

[1] A topological ordering of nodes in a DAG is a linear ordering of its nodes such that each node appears after all its parents in this ordering.

ascertain whether a particular conditional independence statement holds in a Bayesian network. Given the graphical structure and its associated conditional probability distributions, one can use various inference algorithms to do probabilistic reasoning in BNs. Typical algorithms include variable elimination and belief propagation. See [24] for a comprehensive survey.

2.2.2 Markov Networks

The other major class of probabilistic graphical models is known as Markov networks (MNs) or Markov random fields (MRFs). These are undirected graphical models in which nodes still correspond to random variables as in Bayesian networks but the links between them are undirected, hence only indicating the correlations between them, rather than the causality relationships in Bayesian networks.

Again, as in Bayesian networks, the graph structure of Markov networks expresses the qualitative properties of the distribution they encode. Let C denote a clique in the graph, which corresponds to a set of random variables X_C. The joint probability distribution encoded by a Markov network is written as a product of *factors* or *potential functions* $\psi_C(X_C)$ over the maximal cliques of the graph:

$$\Pr(\mathbf{X}_{1:N}) = \frac{1}{Z}\prod_C \psi_C(X_C) \tag{2.5}$$

where all the potential functions are required to be nonnegative and the quantity Z is a normalization constant also known as the *partition function*, which is given by

$$Z = \sum_{\mathbf{X}}\prod_C \psi_C(X_C) \tag{2.6}$$

Because the potential functions are constrained to be nonnegative, it is often useful to express them as exponential functions as

$$\psi_C(X_C) = \exp\{-E(C)\} \tag{2.7}$$

where $E(C)$ is often called the *energy function* for clique C (originates from statistical physics). The joint distribution is defined as the product of local potentials, and the total energy of the Markov network is hence the sum of local energies. An example for a Markov network over 4 random variables is shown in Fig. 2.2. The four potential functions in Fig. 2.2 are $\psi_{X_1,X_2} = \exp\{-(X_1 - X_2)^2\}$, $\psi_{X_1,X_3} = \exp\{-(X_1 - X_3)^2\}$, $\psi_{X_2,X_4} = \exp\{-(X_2 - X_4)^2\}$ and $\psi_{X_3,X_4} = \exp\{-(X_3 - X_4)^2\}$.

Note that different from the conditional probability distribution associated with each node in the Bayesian networks, the potential function in Markov network does not admit a probabilistic

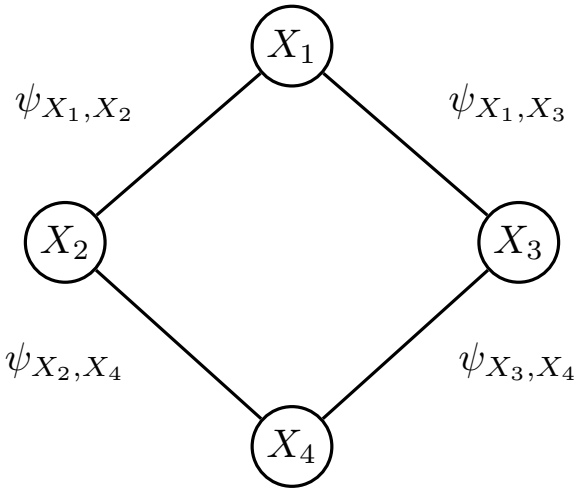


Figure 2.2: A Markov network over 4 random variables, with potential functions over $\{X_1, X_2\}$, $\{X_1, X_3\}$, $\{X_2, X_4\}$ and $\{X_3, X_4\}$.

interpretation. Also, in order to do inference in Markov networks, I need to compute the value of the partition function. This leads to the main difficulty in learning and inference of Markov networks because it normally requires a computation over exponentially many entries.

The test of conditional independence in Markov networks corresponds to the test of graph separation. More concretely, given three sets of random variables (nodes), denoted by $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, $\mathbf{X}$ is conditionally independent of $\mathbf{Y}$ given $\mathbf{Z}$, i.e., $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}$, if there is no path connecting nodes in $\mathbf{X}$ to nodes in $\mathbf{Y}$ that do not contain nodes in $\mathbf{Z}$. Intuitively, this means that if I remove nodes in $\mathbf{Z}$ from the graph, then nodes $\mathbf{X}$ and nodes in $\mathbf{Y}$ are disconnected from each other.

2.2.3 Inference: Variable Elimination Algorithm

Given a probabilistic graphical model over a set of variables $\mathbf{X}_{1:N}$, *inference* refers to the process of computing the posterior probability $\Pr(\mathbf{X}_A|\mathbf{X}_B = \mathbf{x}_B)$, where $A, B \subseteq 1:N$ and $A \cap B = \varnothing$. Roughly, there are three kinds of inference we are interested:

1. Joint inference: $A = 1:N$ and $B = \varnothing$. This corresponds to the case when we want to compute the likelihood of a full instantiation of the variables.

2. Marginal inference: $A \subsetneq 1:N$ and $B = \varnothing$, where we'd like to query the probability for a subset of the random variables.

3. Conditional inference: $B \neq \varnothing$, where we want to query the likelihood of $\mathbf{X}_A$ given that we have observed the event $\mathbf{x}_B$.

All these three kinds of inference queries can be answered in probabilistic graphical models using *inference algorithms*. In this thesis I focus on one specific kind of inference algorithm, known as the *variable elimination* [25] (VE) algorithm, which is simple but yet sufficient for my purpose.

I can use variable elimination algorithm to answer any of the three kinds of queries listed above in a probabilistic graphical model. Let's start from the most general form of inference query: $\Pr(\mathbf{X}_A|\mathbf{X}_B = \mathbf{x}_B)$ where $A, B \subseteq 1:N$ and $A \cap B = \varnothing$. VE takes as input a set of functions, a query set $\mathbf{X}_A$, an evidence set $\mathbf{X}_B = \mathbf{x}_B$, an elimination order over variables and outputs the probability $\Pr(\mathbf{X}_A|\mathbf{X}_B = \mathbf{x}_B)$. Intuitively, VE works in the following way:

1. Select one variable $X \in 1:N \backslash A \cup B$ to be eliminated following the elimination order.

2. Multiply all the functions which contain X.

3. Sum-out X from the product of functions.

VE keeps eliminating variables that do not appear in neither the query set nor the evidence set until there is no such variable. After all the irrelevant variables have been eliminated, one can answer the inference query by first clipping the values of variables in the evidence set and then renormalizing the final function to make it a probability distribution. In the case of a Bayesian network, the set of functions taken as input to VE is the set of conditional probability distributions at each node of the BN. In the case of a Markov network, the set of functions is the set of potential functions defined for each maximal clique in the network.

The order of the variables to be eliminated has a large impact on the running time of VE. However, computing the optimal order of variables which minimizes the computational cost of VE given a fixed network is again, known to be NP-complete (reduction from MAX-CLIQUE problem). Many heuristic algorithms to select a good elimination order exist, but here I focus on introducing the general idea of VE and hence assuming that we're given an elimination order as input to VE. Alg. 1 gives the pseudocode for the variable elimination algorithm:

Algorithm 1 Variable Elimination

Input: A set of functions f_n, a query set $A \subseteq 1:N$, an evidence set $B \subseteq 1:N$ with value $\mathbf{x}_B$, an elimination order π

Output: A posterior probability distribution $\Pr(\mathbf{X}_A|\mathbf{X}_B = \mathbf{x}_B)$

$\Phi \leftarrow \{f_i\}$
for each variable X in π **do**
  // Multiply all the relevant functions
  $\Phi_X \leftarrow \{f_i | X \text{ appears in } f_i\}$
  $f_X \leftarrow \prod_{f \in \Phi_X} f$
  // Summing-out
  $f_{-X} \leftarrow \sum_X f_X$
  $\Phi \leftarrow \Phi \backslash \Phi_X \cup \{f_{-X}\}$
end for
// Value clipping for evidence set B
for each observed variable X in $\mathbf{X}_B$ **do**
  Set the value of X in Φ according to $\mathbf{x}_B$
end for
// Re-normalization
return $\frac{\Phi(\mathbf{X}_A, \mathbf{x}_B)}{\sum_{\mathbf{x}_A} \Phi(\mathbf{x}_A, \mathbf{x}_B)}$

2.2.4 Tree-width

Tree-width is a graph theoretic notion associated with undirected graphs. It is commonly used as a parameter in the parameterized complexity analysis of graph algorithms. The notion of tree-width is especially important in probabilistic graphical models as it has been shown that the computational complexity of exact inference algorithms, including variable elimination, belief propagation, etc, is exponential in the tree-width of the underlying graphical model [24]. To introduce tree-width, I begin by introducing the notion of *tree-decomposition* of an undirected graph as it is closely related to tree-width.

Definition 1 (Tree-decomposition)**.** A *tree-decomposition* of a graph $\mathcal{G} = (\mathcal{G}_V, \mathcal{G}_E)$ is a tree $\mathcal{T}$, with nodes $X_1, X_2, \ldots, X_T$ where each X_i is a subset of $\mathcal{G}_V$, satisfying the following properties:

1. $\bigcup_{i=1}^{T} X_i = \mathcal{G}_V$. That is, each node in $\mathcal{G}$ is contained in at least one node in $\mathcal{T}$.
2. If X_i and X_j both contain a node $v \in \mathcal{G}_V$, then all the nodes X_k of the tree in the unique path between X_i and X_j contain v as well. Equivalently, the tree nodes in $\mathcal{T}$ containing v

form a connected sub-tree of $\mathcal{T}$ (This property is also known as the *running intersection property*).

3. For every edge $(v, w) \in \mathcal{G}_E$, there is a subset X_i that contains both v and w. Or equivalently, every edge in $\mathcal{G}$ is covered by at least one node in $\mathcal{T}$.

The *width* of a tree-decomposition is defined as $\max_i |X_i| - 1$. The *tree-width* of a graph $\mathcal{G}$ is the minimum width among all possible tree-decompositions of $\mathcal{G}$. In the above definition, the size of the largest set X_i is diminished by one to make the tree-width of a tree equal to 1.

There are several equivalent definitions of tree-width. Here I give the one which will be used later in this thesis:

Definition 2 (Tree-width)**.** The tree-width of an undirected graph $\mathcal{G}$ is one less than the size of the maximum (largest) clique in the chordal graph containing $\mathcal{G}$ with the smallest clique number.

As a direct corollary, we know that the tree-width of a complete graph K_n is $n - 1$. Based on the alternative definition of tree-width, I state the following theorem which will be used later to prove the connection between the height of an SPN and the tree-width of a BN:

Theorem 1. For any undirected graph $\mathcal{G}$, we have $\omega(\mathcal{G}) - 1 \leq$ tree-width$(\mathcal{G})$, where $\omega(\mathcal{G})$ is the size of the maximum clique in $\mathcal{G}$.

Thm. 1 says that for any undirected graph $\mathcal{G}$, the size of the largest clique minus 1 is always a lower bound of the tree-width of $\mathcal{G}$. It is also worth mentioning here that it is NP-complete to determine whether a given graph $\mathcal{G}$ has tree-width of at most K [20].

2.3 Algebraic Decision Diagram

One of the key ideas in this thesis is to use Algebraic Decision Diagrams (ADDs) to compactly represent the local conditional probability distributions associated with each node in the Bayesian networks. Intuitively, ADDs can be understood as directed acyclic graphs (DAGs) used to encode discrete functions. There exist other data structures, including decision trees, noisy-OR, etc, which can be used to encode the conditional probability distributions at each node of an BN. One of the advantages of using ADDs lies in the fact that they can exploit the context-specific independence (CSI) among variables while at the same time avoid the replication problem [14] present in decision trees. For readers who are more familiar with Binary Decision Diagrams

(BDDs), an ADD can be viewed as an extension of a BDD where the leaves in ADD are extended to have real values rather than simple binary values as in BDDs.

I 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 3 (Algebraic Decision Diagram [2])**.** 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 $X_n = \texttt{TRUE}$.

The function $f_{\mathcal{A}}(\cdot)$ represented by an ADD $\mathcal{A}$ over $\mathbf{X}_{1:N}$ can be defined recursively. If ADD $\mathcal{A}$ has an empty scope, i.e., it is a leaf node, then $f_{\mathcal{A}}(\mathbf{x}) = c$ for all joint assignment $\mathbf{x}$, where c is the value stored at the leaf node. Otherwise if the ADD $\mathcal{A}$ has non-empty scope, let the variable associated with the root node be X_n, then $f_{\mathcal{A}}(\mathbf{X}_{1:N}) = f_{\mathcal{A}_{X_n=x_n}}(X_1, \ldots, X_{n-1}, X_{n+1}, \ldots, X_N)$, where $\mathcal{A}_{X_n=x_n}$ represents the sub-ADD pointed to by the edge labeled with $X_n = x_n$ from the root of $\mathcal{A}$. The above equations are then recursively applied to all the nodes in $\mathcal{A}$ to build the function $f_{\mathcal{A}}(\mathbf{X}_{1:N})$.

Example 1. Here I show an example of ADD over three boolean variables X_1, X_2 and X_3. The discrete function over X_1, X_2 and X_3 is shown and encoded in Fig. 2.3. Literally, I can also list the function encoded by the ADD in Fig. 2.3 as follows:

Table 2.1: Discrete function encoded in Fig. 2.3

X_1	X_2	X_3	$f(X_1, X_2, X_3)$
True	True	True	0.1
True	True	False	0.9
True	False	True	0.3
True	False	False	0.7
False	True	True	0.3
False	True	False	0.7
False	False	True	0.6
False	False	False	0.4

Note that the ADD in Fig. 2.3 is not the unique ADD representation of the discrete function listed in Table 2.1. Observing that the sub-ADD following the path $X_1 = \texttt{True} \rightarrow X_2 =$

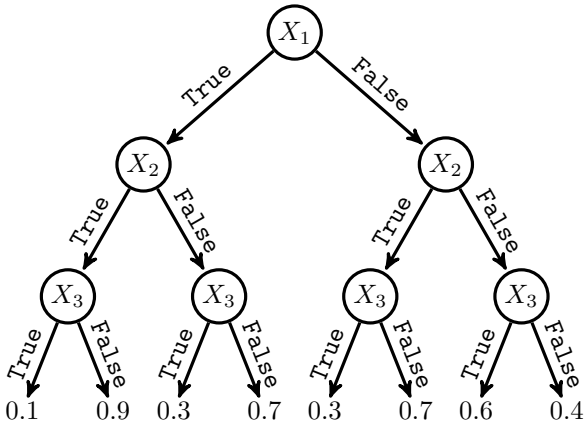


Figure 2.3: An ADD representation of a discrete function over three variables. Each edge emanated from a node X_n is associated with a label either `True` or `False`, which means $X_n =$ `True` or $X_n =$ `False` correspondingly.

`False` is isomorphic to the sub-ADD following the path $X_1 =$ `False` $\rightarrow X_2 =$ `True`, I can construct a more compact ADD representation of the discrete function listed in Table 2.1 in Fig. 2.4.

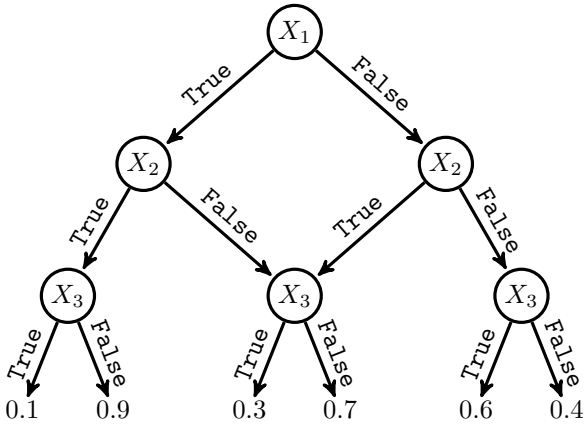


Figure 2.4: A more compact ADD representation of the discrete function listed in Table 2.1.

I 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*[2] between two sets. Henceforth, I will use my extended definition of ADDs throughout the thesis.

For my purpose, I 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 my 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 [4] and effectively avoid the replication problem [14] of the decision tree representation.

I give another example in Fig. 2.5, Fig. 2.6 and Fig. 2.7 where the tabular representation, decision-tree representation and ADD representation of a function of 4 Boolean variables is presented. The tabular representation in Fig. 2.5 cannot exploit CSI and the Decision-Tree representation in Fig. 2.6 cannot reuse isomorphic subgraphs. The ADD representation in Fig. 2.7 can fully exploit CSI by sharing isomorphic subgraphs, which makes it the most compact representation among the three representations. In Fig. 2.6 and Fig. 2.7, the left and right branches of each internal node correspond respectively to `FALSE` and `TRUE`.

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

Figure 2.5: Tabular representation.

[2] A Cartesian product is a binary mathematical operation between two sets A and B which returns a set containing all ordered pairs (a, b) where $a \in A$ and $b \in B$.

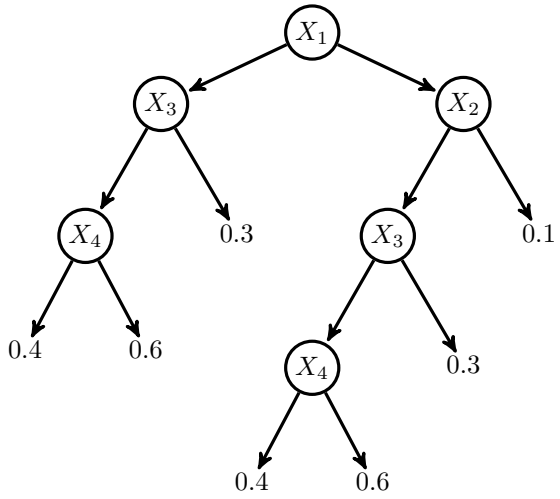


Figure 2.6: Decision-Tree representation.

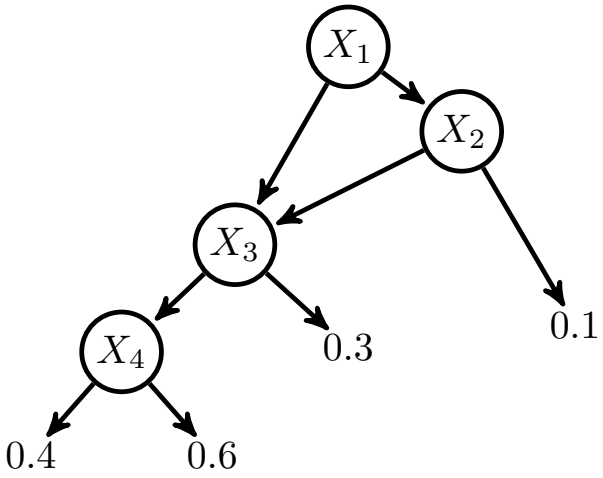


Figure 2.7: ADD representation.

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 [2] for more detailed and thorough discussions about ADDs.

2.4 Sum-Product Network

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

Definition 4 (Network Polynomial [19])**.** 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 [3] 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 5 (Sum-Product Network [19])**.** 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 scope$(v) = \{X\}$, else scope$(v) = \bigcup_{\tilde{v} \in Ch(v)}$ scope$(\tilde{v})$. [19] further define the following properties of an SPN:

Definition 6 (Complete)**.** An SPN is *complete* iff each sum node has children with the same scope.

Definition 7 (Consistent)**.** An SPN is consistent iff no variable appears negated in one child of a product node and non-negated in another.

Definition 8 (Decomposable)**.** An SPN is decomposable iff for every product node v, scope(v_i) $\bigcap$ scope$(v_j) = \varnothing$ where $v_i, v_j \in Ch(v), i \neq j$.

An SPN is said to be *valid* iff it defines a (unnormalized) probability distribution. [19] 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 thesis, I focus only on complete and consistent SPNs as I am 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 $\Pr_{\mathcal{S}}(\mathbf{x}) \triangleq \frac{f_S(\mathbf{x})}{\sum_{\mathbf{x}} f_S(\mathbf{x})}$, where $f_{\mathcal{S}}(\cdot)$ is the network polynomial defined by the root of the SPN $\mathcal{S}$.

Example 2. An example of a complete and consistent SPN is given in Fig. 2.8. The SPN in

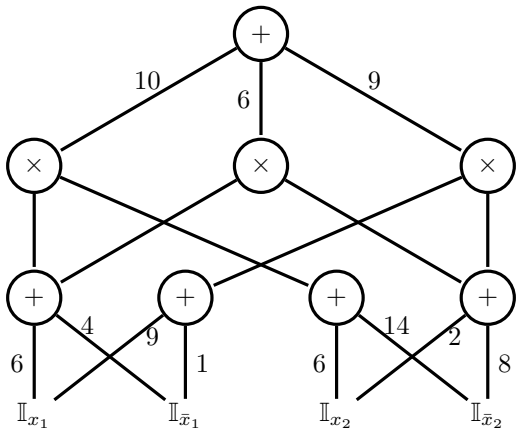


Figure 2.8: A complete and consistent SPN over Boolean variables X_1, X_2.

Fig. 2.8 is also decomposable since every product node has children whose scopes do not intersect. The network polynomial defined by (the root of) this SPN is: $f(X_1, X_2) = 10(6\mathbb{I}_{x_1} + 4\mathbb{I}_{\bar{x}_1})(6\mathbb{I}_{x_2} + 14\mathbb{I}_{\bar{x}_2}) + 6(6\mathbb{I}_{x_1} + 4\mathbb{I}_{\bar{x}_1})(2\mathbb{I}_{x_2} + 8\mathbb{I}_{\bar{x}_2}) + 9(9\mathbb{I}_{x_1} + \mathbb{I}_{\bar{x}_1})(2\mathbb{I}_{x_2} + 8\mathbb{I}_{\bar{x}_2}) = 594\mathbb{I}_{x_1}\mathbb{I}_{x_2} + 1776\mathbb{I}_{x_1}\mathbb{I}_{\bar{x}_2} + 306\mathbb{I}_{\bar{x}_1}\mathbb{I}_{x_2} + 824\mathbb{I}_{\bar{x}_1}\mathbb{I}_{\bar{x}_2}$ and the probability distribution induced by $\mathcal{S}$ is $\Pr_{\mathcal{S}} = \frac{594}{3500}\mathbb{I}_{x_1}\mathbb{I}_{x_2} + \frac{1776}{3500}\mathbb{I}_{x_1}\mathbb{I}_{\bar{x}_2} + \frac{306}{3500}\mathbb{I}_{\bar{x}_1}\mathbb{I}_{x_2} + \frac{824}{3500}\mathbb{I}_{\bar{x}_1}\mathbb{I}_{\bar{x}_2}$.

The computational complexity of inference queries in an SPN $\mathcal{S}$, including joint inference, conditional inference and marginal inference is $O(|\mathcal{S}|)$. I give the joint inference algorithm for SPNs below:

Algorithm 2 Joint Inference in an SPN

Input: SPN $\mathcal{S}$, joint query $\mathbf{x}$, normalization constant $Z_{\mathcal{S}}$
Output: $\Pr_{\mathcal{S}}(\mathbf{x})$

1: **for** each leaf nodes $\mathbb{I}_{x_n}$ **do**
2: **if** $X_n = \mathbf{x}_n$ **then**
3: Set $\mathbb{I}_{x_n} = 1$
4: **else**
5: Set $\mathbb{I}_{x_n} = 0$
6: **end if**
7: **end for**
8: **for** every internal node $V \in \mathcal{S}$ in bottom-up order **do**
9: **if** V is a sum node **then**
10: Set $val(V) = \sum_i w_i \times val(V_i)$
11: **else if** V is a product node **then**
12: Set $val(V) = \prod_i val(V_i)$
13: **end if**
14: **end for**
15: **return** $\frac{1}{Z_{\mathcal{S}}} val(Root(\mathcal{S}))$

In Line 10 of Alg. 2, w_i are the weights associated with the sum node V and $val(V_i)$ is the value stored at the ith child of V. $Z_{\mathcal{S}}$ is the normalization constant in SPN $\mathcal{S}$ which can be computed from the root node of $\mathcal{S}$ by setting all the leaf nodes to 1. It's clear that in Alg. 2 I visit each node and edge in $\mathcal{S}$ once in the bottom-up order to compute the joint probability, so that the computational complexity for Alg. 2 is $O(|\mathcal{S}|)$.

Now I proceed to give the algorithm for marginal inference in SPNs in Alg. 3. The only difference between marginal inference and joint inference in SPNs lies in the fact that in marginal inference I set the value of all leaf nodes that do not appear in the marginal query to be 1 since they should be summed out.

For conditional inference, I can compute the conditional probability $\Pr(\mathbf{x}|\mathbf{y})$ by using Bayes' rule $\Pr(\mathbf{x}|\mathbf{y}) = \frac{\Pr(\mathbf{x},\mathbf{y})}{\Pr(\mathbf{y})}$ where the numerator corresponds to a joint inference query that can be answered with Alg. 2 and the denominator can be computed using Alg. 3 as a marginal query. Again, to compute the conditional probability, I only need to traverse the SPN twice, so the computational complexity is again, $O(|\mathcal{S}|)$.

Example 3. I now give a concrete example to illustrate how to answer a joint query, a marginal query and a conditional query in an SPN using Alg. 2 and Alg. 3. Consider the SPN in Fig. 2.8 for example.

Algorithm 3 Marginal Inference in SPN

Input: SPN $\mathcal{S}$, marginal query $\mathbf{x}$, normalization constant $Z_{\mathcal{S}}$
Output: $\Pr_{\mathcal{S}}(\mathbf{x})$

for each leaf nodes $\mathbb{I}_{x_n}$ **do**
  if X_n is assigned value in $\mathbf{x}$ **then**
    if $X_n = \mathbf{x}_n$ **then**
      Set $\mathbb{I}_{x_n} = 1$
    else
      Set $\mathbb{I}_{x_n} = 0$
    end if
  else
    Set $\mathbb{I}_{x_n} = 1$
  end if
end for
for every internal node $V \in \mathcal{S}$ in bottom-up order **do**
  if V is a sum node **then**
    Set $val(V) = \sum_i w_i \times val(V_i)$
  else if V is a product node **then**
    Set $val(V) = \prod_i val(V_i)$
  end if
end for
return $\frac{1}{Z_{\mathcal{S}}} val(Root(\mathcal{S}))$

First, I need to compute the normalization constant $Z_{\mathcal{S}}$ since it is required for both Alg. 2 and Alg. 3. Setting all the values of leaf nodes in Fig. 2.8 to 1, propagating the values in bottom-up order, I obtain the normalization constant at the root, given by

$$Z_{\mathcal{S}} = 10(6+4) \times (6+14) + 6(6+4) \times (2+8) + 9(9+1) \times (2+8) = 3500$$

Now suppose I am interested in computing the joint query $\Pr(X_1 = \texttt{True}, X_2 = \texttt{False})$. Following Alg. 2, I set the values of the leaf nodes as follows: $\mathbb{I}_{x_1} = 1, \mathbb{I}_{\bar{x}_1} = 0, \mathbb{I}_{x_2} = 0$ and $\mathbb{I}_{\bar{x}_2} = 1$. Again, I propagate the values in the SPN until I obtain the value at the root node, which is 1776. Then I can obtain the joint probability by normalizing the value by $Z_{\mathcal{S}}$, which gives the joint probability $\Pr(X_1 = \texttt{True}, X_2 = \texttt{False}) = \frac{1776}{3500}$.

If I am interested in computing the marginal query $\Pr(X_2 = \texttt{False})$, I can first set the values of the leaf nodes in $\mathcal{S}$ as $\mathbb{I}_{x_1} = 1, \mathbb{I}_{\bar{x}_1} = 1, \mathbb{I}_{x_2} = 0$ and $\mathbb{I}_{\bar{x}_2} = 1$. I set both the values of $\mathbb{I}_{x_1}$ and $\mathbb{I}_{\bar{x}_1}$ to 1 because in the marginal query the variable X_1 needs to be summed out. Following

Alg. 3, I obtain 2600 as the value at the root node. The final step, again, is to normalize the value by the normalization constant. Doing so gives us $\Pr(X_2 = \texttt{False}) = \frac{2600}{3500}$.

Assume I am also interested in computing the value of the conditional query $\Pr(X_1 = \texttt{True}|X_2 = \texttt{False})$. I first apply the Bayes' rule:

$$\Pr(X_1 = \texttt{True}|X_2 = \texttt{False}) = \frac{\Pr(\mathbb{X}_1 = \texttt{True}, X_2 = \texttt{False})}{\Pr(X_2 = \texttt{False})}$$

Since both the numerator and the denominator have been computed before, I can obtain the value of $\Pr(X_1 = \texttt{True}|X_2 = \texttt{False})$ immediately as $\Pr(X_1 = \texttt{True}|X_2 = \texttt{False}) = \frac{1776}{2600}$.

2.4.1 Related Work

Different from traditional probabilistic graphical models, where the computational complexity for exact inference can still be intractable even the graph admits a compact representation, the inference complexity in SPN, including joint inference, marginal inference and conditional inference, is linear in the size of SPN. For example, consider a Markov network that is an $n \times n$ grid with $(n+1)^2$ nodes and $2n(n+1)$ edges. Although the $n \times n$ grid Markov net is a sparse graph, the computational complexity for exact inference in this graph is $O(2^n)$, which is computationally intractable even for a moderate size of n. In general, the computational complexity for exact inference in graphical models, including both Bayesian networks and Markov networks, is exponential in the tree-width of the graph. However, as we introduced in the last section, given a graph, it is NP-complete to compute the tree-width of the graph. Hence practitioners often resort to approximate inference algorithms, such as variational inference methods [27], Markov Chain Monte Carlo based sampling methods [23], loopy belief propagation [13], etc, to handle the inference problem in graphical models. SPNs open the door for tractable exact inference, while at the same time do not sacrifice its representation power [19]. Due to their flexibility and tractability, SPNs have been widely applied in various fields, for example, image completion [19, 8], activity modeling [1], speech modeling [17], language modeling [6] and density estimation [10, 21].

There are two main research problems in SPNs: parameter learning and structure learning. Given an SPN with fixed structure, Domingos et al. proposed both generative learning and discriminative learning algorithms for parameters [19, 9] in SPNs. At a high level, these approaches view SPNs as deep architectures and apply projected gradient descent to optimize the data log-likelihood function for parameter learning. Structure learning algorithms for SPNs have also been designed [8, 10, 16, 21]. In summary, there are two strands of methods for learning the structure of SPNs: top-down approach and bottom-up approach.

In [16], Peharz et al. proposed a bottom-up approach to construct an SPN from simple models over small variable scopes, and grow the model over larger and larger variable scopes, until a node with complete scope is reached. The limitations in their method lie in two folds: 1). the final SPN has a rigid structure in the sense that every product node will only have two children since only the region pair with the least statistical dependence test will be merged during the structure learning process. 2). it happens during the algorithm that there is no two regions can be merged before the root node in constructed hence the algorithm may fail due to the decomposability constraint of SPN.

Another line of research focuses on top-down learning of SPNs. Dennis and Ventura proposed the first structure learning algorithm for SPNs [8], where the authors applied K-means clustering on variables to generate region graphs and then expand each region graph with a fixed structure to build an SPN. Gens and Domingos built the first connection between structure learning of SPN and hierarchical clustering [10]. In their pioneering work, which we refer to as LearnSPN, they used a naive bayes mixture with EM training to build sum nodes and used a statistical independence test (G-test) to consider pair-wise relationships between variables in order to build product nodes. This process is recursively applied until there is only once instance or one variable in current split of the data set. However, LearnSPN always tries to build a product node first (split on variables), and only when this step fails it tries to build a sum node (split on instances). This will cause the final SPN produced by LearnSPN to have consecutive sum nodes, which is redundant because consecutive sum nodes can be efficiently combined into one sum node without changing the network polynomial. More importantly, consecutive sum nodes essentially build more hard clusters on instances with each cluster owning only a very small fraction of the total data set, hence this will cause the following statistical independence test to be unreliable when building a product node. As a consequence, the final SPN produced by LearnSPN tends to be very large but shallow.

Based on LearnSPN and aiming at modeling both direct and indirect interactions among variables, Rooshenas et al. came up with a new model (ID-SPN) which combines both SPNs and Markov networks [21]. The final model is compiled into an arithmetic circuit for fast inference. ID-SPN explores an interesting direction on combining SPN with classic graphical models, including Bayesian networks and Markov networks, for better performance on density estimation. Interested readers are refered to [12] for more work on this direction.

To understand the theoretical properties of SPNs, Peharz et al. independently showed that the weights of edges from each sum node in an SPN can be locally normalized without changing the probability distribution induced by the SPN [18]. Furthermore, they also independently proved that complete and consistent SPNs cannot model a distribution exponentially more compact than complete and decomposable SPNs. However, to the best of my knowledge, there is no theoretical work on building the connection between SPNs and traditional probabilistic graphical models.

In this thesis, I give two constructive algorithms (and also proofs) on converting from one model to the other without an exponential blow-up in both time and space. This helps to understand better the inner working mechanism of SPNs. Also, by building the corresponding BN from an SPN, one can understand the interactions among the observable random variables in an SPN by inspecting the topological structure of the corresponding BN.

Chapter 3

Main Results

In this chapter, I first state the main results obtained in this thesis and then provide detailed proofs with some discussion of the results. To keep the presentation simple, I assume without loss of generality that all the random variables are Boolean unless explicitly stated. It is straightforward to extend my 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 2. 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. 2 immediately leads to the following corollary:

Corollary 3. 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 2 and Corollary 3 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 4. 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 2 and 4 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).

3.1 Normal Sum-Product Network

To make the upcoming proofs concise, I 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. I then derive the proofs with normal SPNs. Note that in the thesis I only focus on SPNs that are *complete* and *consistent* since I am mainly interested in the (unnormalized) joint probability distribution associated with complete and consistent SPNs. Complete and consistent SPNs are interesting because they form a super class of complete and decomposable SPNs, which are frequently used and discussed in the literature. Hence, when I refer to an SPN, I assume that it is complete and consistent without explicitly stating this.

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 9. 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 an 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 5. 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 $|\mathcal{S}'| = O(|\mathcal{S}|^2)$.

To show this, I first prove the following lemmas.

Lemma 6. For any complete and consistent SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$, there exists a complete and decomposable SPN $\mathcal{S}'$ over $\mathbf{X}_{1:N}$ such that $f_{\mathcal{S}}(\mathbf{x}) = f_{\mathcal{S}'}(\mathbf{x}), \forall \mathbf{x}$ and $|\mathcal{S}'| = O(|\mathcal{S}|^2)$.

Proof. Let $\mathcal{S}$ be a complete and consistent SPN. If it is also decomposable, then simply set $\mathcal{S}' = \mathcal{S}$ and I am done. Otherwise, let $v_1, \ldots, v_M$ be an inverse topological ordering of all the nodes in $\mathcal{S}$, including both terminal nodes and internal nodes, such that for any $v_m, m \in 1:M$, all the ancestors of v_m in the graph appear after v_m in the ordering. Let v_m be the first product node in the ordering that violates decomposability. Let $v_{m_1}, v_{m_2}, \ldots, v_{m_l}$ be the children of v_m where $m_1 < m_2 < \cdots < m_l < m$ (due to the inverse topological ordering). Let $(v_{m_i}, v_{m_j}), i < j, i, j \in 1:l$ be the first ordered pair of nodes such that $\text{scope}(v_{m_i}) \bigcap \text{scope}(v_{m_j}) \neq \varnothing$. Hence, let $X \in \text{scope}(v_{m_i}) \bigcap \text{scope}(v_{m_j})$. Consider $f_{v_{m_i}}$ and $f_{v_{m_j}}$ which are the network polynomials defined by the sub-SPNs rooted at v_{m_i} and v_{m_j}.

Expand network polynomials $f_{v_{m_i}}$ and $f_{v_{m_j}}$ into a *sum-of-product* form by applying the distributive law between products and sums. For example, if $f(X_1, X_2) = (\mathbb{I}_{x_1} + 9\mathbb{I}_{\bar{x}_1})(4\mathbb{I}_{x_2} + 6\mathbb{I}_{\bar{x}_2})$, then the expansion of f is $f(X_1, X_2) = 4\mathbb{I}_{x_1}\mathbb{I}_{x_2} + 6\mathbb{I}_{x_1}\mathbb{I}_{\bar{x}_2} + 36\mathbb{I}_{\bar{x}_1}\mathbb{I}_{x_2} + 54\mathbb{I}_{\bar{x}_1}\mathbb{I}_{\bar{x}_2}$. Since $\mathcal{S}$ is complete, then sub-SPNs rooted at v_{m_i} and v_{m_j} are also complete, which means that each monomial in the expansion of $f_{v_{m_i}}$ must share the same scope. The same applies to $f_{v_{m_j}}$. Since $X \in \text{scope}(v_{m_i}) \bigcap \text{scope}(v_{m_j})$, then every monomial in the expansion of $f_{v_{m_i}}$ and $f_{v_{m_j}}$ must contain an indicator variable over X, either $\mathbb{I}_x$ or $\mathbb{I}_{\bar{x}}$. Furthermore, since $\mathcal{S}$ is consistent, then the sub-SPN rooted at v_m is also consistent. Consider $f_{v_m} = \prod_{k=1}^{l} f_{v_{m_k}} = f_{v_{m_i}} f_{v_{m_j}} \prod_{k \neq i,j} f_{v_{m_k}}$. Because v_m is consistent, I know that each monomial in the expansions of $f_{v_{m_i}}$ and $f_{v_{m_j}}$ must contain the same indicator variable of X, either $\mathbb{I}_x$ or $\mathbb{I}_{\bar{x}}$, otherwise there will be a term $\mathbb{I}_x\mathbb{I}_{\bar{x}}$ in f_{v_m} which violates the consistency assumption. Without loss of generality, assume each monomial in the expansions of $f_{v_{m_i}}$ and $f_{v_{m_j}}$ contains $\mathbb{I}_x$. Then I can re-factorize f_{v_m} in the following way:

$$
\begin{aligned}
f_{v_m} &= \prod_{k=1}^{l} f_{v_{m_k}} = \mathbb{I}_x^2 \frac{f_{v_{m_i}}}{\mathbb{I}_x} \frac{f_{v_{m_j}}}{\mathbb{I}_x} \prod_{k \neq i,j} f_{v_{m_k}} \\
&= \mathbb{I}_x \frac{f_{v_{m_i}}}{\mathbb{I}_x} \frac{f_{v_{m_j}}}{\mathbb{I}_x} \prod_{k \neq i,j} f_{v_{m_k}} = \mathbb{I}_x \tilde{f}_{v_{m_i}} \tilde{f}_{v_{m_j}} \prod_{k \neq i,j} f_{v_{m_k}}
\end{aligned} \tag{3.1}
$$

where I use the fact that indicator variables are idempotent, i.e., $\mathbb{I}_x^2 = \mathbb{I}_x$ and $\tilde{f}_{v_{m_i}}(\tilde{f}_{v_{m_j}})$ is defined as the function by factorizing $\mathbb{I}_x$ out from $f_{v_{m_i}}(f_{v_{m_j}})$. Eq. 3.1 means that in order to make v_m decomposable, I can simply remove all the indicator variables $\mathbb{I}_x$ from sub-SPNs rooted at v_{m_i} and v_{m_j} and later link $\mathbb{I}_x$ to v_m directly. Such a transformation will not change the network polynomial f_{v_m} as shown by Eq. 3.1, but it will remove X from $\text{scope}(v_{m_i}) \bigcap \text{scope}(v_{m_j})$. In principle, I can apply this transformation to all ordered pairs $(v_{m_i}, v_{m_j}), i < j, i, j \in 1:l$ with nonempty intersections of scope. However, this is not algorithmically efficient and more importantly, for local components containing $\mathbb{I}_x$ in f_{v_m} which are reused by other nodes v_n outside of

$\mathcal{S}_{v_m}$, I cannot remove $\mathbb{I}_x$ from them otherwise the network polynomials for each such v_n will be changed due to the removal. In such case, I need to duplicate the local components to ensure that local transformations with respect to f_{v_m} do not affect network polynomials f_{v_n}. I present the transformation in Alg. 4. Alg. 4 transforms a complete and consistent SPN $\mathcal{S}$ into a complete

Algorithm 4 Decomposition Transformation

Input: Complete and consistent SPN $\mathcal{S}$.
Output: Complete and decomposable SPN $\mathcal{S}'$.

Let $v_1, v_2, \ldots, v_M$ be an inverse topological ordering of nodes in $\mathcal{S}$.
for $m = 1$ to M **do**
  if v_m is a non-decomposable product node **then**
    $\Omega(v_m) \leftarrow \bigcup_{i \neq j} \text{scope}(v_{m_i}) \bigcap \text{scope}(v_{m_j})$
    $\mathbf{V} \leftarrow \{v \in \mathcal{S}_{v_m} \mid \text{scope}(v) \bigcap \Omega(v_m) \neq \varnothing\}$
    $\mathcal{S}_{\mathbf{V}} \leftarrow \mathcal{S}_{v_m}|_{\mathbf{V}}$
    $D(v_m) \leftarrow$ descendants of v_m
    for node $v \in \mathcal{S}_{\mathbf{V}} \backslash \{v_m\}$ **do**
      if $Pa(v) \backslash D(v_m) \neq \varnothing$ **then**
        Create $p \leftarrow v \otimes \prod_{X \in \Omega(v_m) \cap \text{scope}(v)} \mathbb{I}_{x^*}$
        Connect p to $\forall f \in Pa(v) \backslash D(v_m)$
        Disconnect v from $\forall f \in Pa(v) \backslash D(v_m)$
      end if
    end for
    for node $v \in \mathcal{S}_{\mathbf{V}}$ in bottom-up order **do**
      Disconnect $\tilde{v} \in Ch(v)$ $\forall \text{scope}(\tilde{v}) \subseteq \Omega(v_m)$
    end for
    Connect $\prod_{X \in \Omega(v_m)} \mathbb{I}_{x^*}$ to v_m directly
  end if
end for
Delete all nodes unreachable from the root of $\mathcal{S}$
Delete all product nodes with out-degree 0
Contract all product nodes with out-degree 1

and decomposable SPN $\mathcal{S}'$. Informally, it works using the following identity:

$$f_{v_m} = \left(\prod_{X \in \Omega(v_m)} \mathbb{I}_{x^*} \right) \prod_{k=1}^{l} \frac{f_{v_{m_k}}}{\prod_{X \in \Omega(v_m) \cap \text{scope}(v_{m_k})} \mathbb{I}^*_x} \tag{3.2}$$

where $\Omega(v_m) \triangleq \bigcup_{i,j \in 1:l, i \neq j} \text{scope}(v_{m_i}) \cap \text{scope}(v_{m_j})$, i.e., $\Omega(v_m)$ is the union of all the shared

variables between pairs of children of v_m and $\mathbb{I}_{x^*}$ is the indicator variable of $X \in \Omega(v_m)$ appearing in $\mathcal{S}_{v_m}$. Based on the analysis above, I know that for each $X \in \Omega(v_m)$ there will be only one kind of indicator variable $\mathbb{I}_{x^*}$ that appears inside $\mathcal{S}_{v_m}$, otherwise v_m is not consistent. In Line 6, $\mathcal{S}_{v_m}|_{\mathbf{V}}$ is defined as the sub-SPN of $\mathcal{S}_{v_m}$ induced by the node set $\mathbf{V}$, i.e., a subgraph of $\mathcal{S}_{v_m}$ where the node set is restricted to $\mathbf{V}$. In Lines 5-6, I first extract the induced sub-SPN $\mathcal{S}_{\mathbf{V}}$ from $\mathcal{S}_{v_m}$ rooted at v_m using the node set in which nodes have nonempty intersections with $\Omega(v_m)$. I disconnect the nodes in $\mathcal{S}_{\mathbf{V}}$ from their children if their children are indicator variables of a subset of $\Omega(v_m)$ (Lines 15-17). At Line 18, I build a new product node by multiplying all the indicator variables in $\Omega(v_m)$ and link it to v_m directly. To keep unchanged the network polynomials of nodes outside $\mathcal{S}_{v_m}$ that use nodes in $\mathcal{S}_{\mathbf{V}}$, I create a duplicate node p for each such node v and link p to all the parents of v outside of $\mathcal{S}_{v_m}$ and at the same time delete the original link (Lines 9-13).

In summary, Lines 15-17 ensure that v_m is decomposable by removing all the shared indicator variables in $\Omega(v_m)$. Line 18 together with Eq. 3.2 guarantee that f_{v_m} is unchanged after the transformation. Lines 9-13 create necessary duplicates to ensure that other network polynomials are not affected. Lines 21-23 simplify the transformed SPN to make it more compact. An example is depicted in Fig. 3.1 to illustrate the transformation process.

I now analyze the size of the SPN constructed by Alg. 4. For a graph $\mathcal{S}$, let $\mathfrak{V}(\mathcal{S})$ be the number of nodes in $\mathcal{S}$ and let $\mathfrak{E}(\mathcal{S})$ be the number of edges in $\mathcal{S}$. Note that in Lines 8-17 I only focus on nodes that appear in the induced SPN $\mathcal{S}_{\mathbf{V}}$, which clearly has $|\mathcal{S}_{\mathbf{V}}| \leq |\mathcal{S}_{v_m}|$. Furthermore, I create a new product node p at Line 10 iff v is reused by other nodes which do not appear in $\mathcal{S}_{v_m}$. This means that the number of nodes created during each iteration between Lines 2 and 20 is bounded by $\mathfrak{V}(\mathcal{S}_{\mathbf{V}}) \leq \mathfrak{V}(\mathcal{S}_{v_m})$. Line 10 also creates 2 new edges to connect p to v and the indicator variables. Lines 11 and 12 first connect edges to p and then delete edges from v, hence these two steps do not yield increases in the number of edges. So the increase in the number of edges is bounded by $2\mathfrak{V}(\mathcal{S}_{\mathbf{V}}) \leq 2\mathfrak{V}(\mathcal{S}_{v_m})$. Combining increases in both nodes and edges, during each outer iteration the increase in size is bounded by $3|\mathcal{S}_{\mathbf{V}}| \leq 3|\mathcal{S}_{v_m}| = O(|\mathcal{S}|)$. There will be at most $M = \mathfrak{V}(\mathcal{S})$ outer iterations hence the total increase in size will be bounded by $O(M|\mathcal{S}|) = O(|\mathcal{S}|^2)$. □

Lemma 7. For any complete and decomposable SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$ that satisfies condition 2 of Def. 9, $\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x}) = 1$.

Proof. I give a proof by induction on the height of $\mathcal{S}$. Let R be the root of $\mathcal{S}$.

- Base case. SPNs of height 0 are indicator variables over some Boolean variable whose network polynomials immediately satisfy Lemma 7.
- Induction step. Assume Lemma 7 holds for any SPN with height $\leq k$. Consider an SPN $\mathcal{S}$ with height $k+1$. I consider the following two cases:

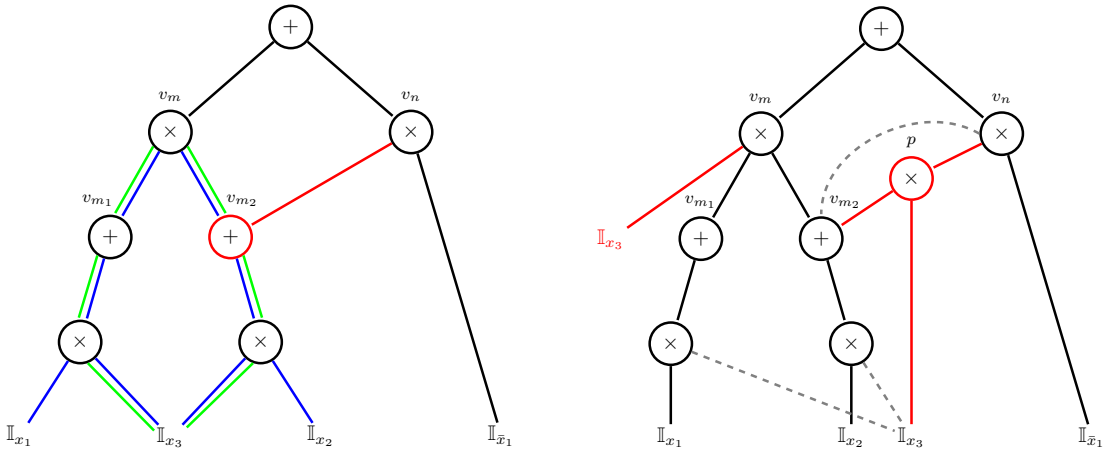


Figure 3.1: Transformation process described in Alg. 4 to construct a complete and decomposable SPN from a complete and consistent SPN. The product node v_m in the left SPN is not decomposable. Induced sub-SPN $\mathcal{S}_{v_m}$ is highlighted in blue and $\mathcal{S}_{\mathbf{V}}$ is highlighted in green. v_{m_2} highlighted in red is reused by v_n which is outside $\mathcal{S}_{v_m}$. To compensate for v_{m_2}, I create a new product node p in the right SPN and connect it to indicator variable $\mathbb{I}_{x_3}$ and v_{m_2}. Dashed gray lines in the right SPN denote deleted edges and nodes while red edges and nodes are added during Alg. 4.

– The root R of $\mathcal{S}$ is a product node. Then in this case the network polynomial $f_{\mathcal{S}}(\cdot)$ for $\mathcal{S}$ is defined as $f_{\mathcal{S}} = \prod_{v \in Ch(R)} f_v$. I have

$$\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x}) = \sum_{\mathbf{x}} \prod_{v \in Ch(R)} f_v(\mathbf{x}|_{\text{scope}(v)}) \tag{3.3}$$

$$= \prod_{v \in Ch(R)} \sum_{\mathbf{x}|_{\text{scope}(v)}} f_v(\mathbf{x}|_{\text{scope}(v)}) \tag{3.4}$$

$$= \prod_{v \in Ch(R)} 1 = 1 \tag{3.5}$$

where $\mathbf{x}|_{\text{scope}(v)}$ means that $\mathbf{x}$ is restricted to the set $\text{scope}(v)$. Eq. 3.4 follows from the decomposability of R and Eq. 3.5 follows from the induction hypothesis.

– The root R of $\mathcal{S}$ is a sum node. The network polynomial is $f_{\mathcal{S}} = \sum_{v \in Ch(R)} w_{R,v} f_v$. I

have

$$\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x}) = \sum_{\mathbf{x}} \sum_{v \in Ch(R)} w_{R,v} f_v(\mathbf{x}) \tag{3.6}$$

$$= \sum_{v \in Ch(R)} w_{R,v} \sum_{\mathbf{x}} f_v(\mathbf{x}) \tag{3.7}$$

$$= \sum_{v \in Ch(R)} w_{R,v} = 1 \tag{3.8}$$

Eq. 3.7 follows from the commutative and associative law of addition and Eq. 3.8 follows by the induction hypothesis.

□

Corollary 8. For any complete and decomposable SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$ that satisfies condition 2 of Def. 9, $\Pr_{\mathcal{S}}(\cdot) = f_{\mathcal{S}}(\cdot)$.

Lemma 9. For any complete and decomposable SPN $\mathcal{S}$, there exists an SPN $\mathcal{S}'$ where the weights of the edges emanating from every sum node are nonnegative and sum to 1, and $\Pr_{\mathcal{S}}(\cdot) = \Pr_{\mathcal{S}'}(\cdot)$, $|\mathcal{S}'| = |\mathcal{S}|$.

Proof. Alg. 5 runs in one pass of $\mathcal{S}$ to construct the required SPN $\mathcal{S}'$. I proceed to prove that the

Algorithm 5 Weight Normalization

Input: SPN $\mathcal{S}$
Output: SPN $\mathcal{S}'$

$\mathcal{S}' \leftarrow \mathcal{S}$
$val(\mathbb{I}_x) \leftarrow 1, \forall \mathbb{I}_x \in \mathcal{S}$
Let $v_1, \ldots, v_M$ be an inverse topological ordering of the nodes in $\mathcal{S}$
for $m = 1$ to M **do**
  if v_m is a sum node **then**
    $val(v_m) \leftarrow \sum_{v \in Ch(v_m)} w_{v_m,v} val(v)$
    $w'_{v_m,v} \leftarrow \frac{w_{v_m,v} val(v)}{val(v_m)}, \quad \forall v \in Ch(v_m)$
  else if v_m is a product node **then**
    $val(v_m) \leftarrow \prod_{v \in Ch(v_m)} val(v)$
  end if
end for

SPN $\mathcal{S}'$ returned by Alg. 5 satisfies $\Pr_{\mathcal{S}'}(\cdot) = \Pr_{\mathcal{S}}(\cdot)$, $|\mathcal{S}'| = |\mathcal{S}|$ and that $\mathcal{S}'$ satisfies condition 2 of Def. 9. It is clear that $|\mathcal{S}'| = |\mathcal{S}|$ because I only modify the weights of $\mathcal{S}$ to construct $\mathcal{S}'$ at Line 7. Based on Lines 6 and 7, it is also straightforward to verify that for each sum node v in $\mathcal{S}'$, the weights of the edges emanating from v are nonnegative and sum to 1. I now show that $\Pr_{\mathcal{S}'}(\cdot) = \Pr_{\mathcal{S}}(\cdot)$. Using Corollary 8, $\Pr_{\mathcal{S}'}(\cdot) = f_{\mathcal{S}'}(\cdot)$. Hence it is sufficient to show that $f_{\mathcal{S}'}(\cdot) = \Pr_{\mathcal{S}}(\cdot)$. Before deriving a proof, it is helpful to note that for each node $v \in \mathcal{S}$, $val(v) = \sum_{\mathbf{x}|_{\text{scope}(v)}} f_v(\mathbf{x}|_{\text{scope}(v)})$. I give a proof by induction on the height of $\mathcal{S}$.

- Base case. SPNs with height 0 are indicator variables which automatically satisfy Lemma 9.
- Induction step. Assume Lemma 9 holds for any SPN of height $\leq k$. Consider an SPN $\mathcal{S}$ of height $k+1$. Let R be the root node of $\mathcal{S}$ with out-degree l. I discuss the following two cases.
  - R is a product node. Let $R_1, \ldots, R_l$ be the children of R and $\mathcal{S}_1, \ldots, \mathcal{S}_l$ be the corresponding sub-SPNs. By induction, Alg. 5 returns $\mathcal{S}'_1, \ldots, \mathcal{S}'_l$ that satisfy Lemma 9. Since R is a product node, I have

$$f_{\mathcal{S}'}(\mathbf{x}) = \prod_{i=1}^{l} f_{\mathcal{S}'_i}(\mathbf{x}|_{\text{scope}(R_i)}) \tag{3.9}$$

$$= \prod_{i=1}^{l} \Pr_{\mathcal{S}_i}(\mathbf{x}|_{\text{scope}(R_i)}) \tag{3.10}$$

$$= \prod_{i=1}^{l} \frac{f_{\mathcal{S}_i}(\mathbf{x}|_{\text{scope}(R_i)})}{\sum_{\mathbf{x}|_{\text{scope}(R_i)}} f_{\mathcal{S}_i}(\mathbf{x}|_{\text{scope}(R_i)})} \tag{3.11}$$

$$= \frac{\prod_{i=1}^{l} f_{\mathcal{S}_i}(\mathbf{x}|_{\text{scope}(R_i)})}{\sum_{\mathbf{x}} \prod_{i=1}^{l} f_{\mathcal{S}_i}(\mathbf{x}|_{\text{scope}(R_i)})} \tag{3.12}$$

$$= \frac{f_{\mathcal{S}}(\mathbf{x})}{\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x})} = \Pr_{\mathcal{S}}(\mathbf{x}) \tag{3.13}$$

Eq. 3.10 follows from the induction hypothesis and Eq. 3.12 follows from the distributive law due to the decomposability of $\mathcal{S}$.

– R is a sum node with weights $w_1, \ldots, w_l \geq 0$. I have

$$f_{\mathcal{S}'}(\mathbf{x}) = \sum_{i=1}^{l} w_i' f_{\mathcal{S}_i'}(\mathbf{x}) \tag{3.14}$$

$$= \sum_{i=1}^{l} \frac{w_i val(R_i)}{\sum_{j=1}^{l} w_j val(R_j)} \Pr_{\mathcal{S}_i}(\mathbf{x}) \tag{3.15}$$

$$= \sum_{i=1}^{l} \frac{w_i val(R_i)}{\sum_{j=1}^{l} w_j val(R_j)} \frac{f_{\mathcal{S}_i}(\mathbf{x})}{\sum_{\mathbf{x}} f_{\mathcal{S}_i}(\mathbf{x})} \tag{3.16}$$

$$= \sum_{i=1}^{l} \frac{w_i val(R_i)}{\sum_{j=1}^{l} w_j val(R_j)} \frac{f_{\mathcal{S}_i}(\mathbf{x})}{val(R_i)} \tag{3.17}$$

$$= \frac{\sum_{i=1}^{l} w_i f_{\mathcal{S}_i}(\mathbf{x})}{\sum_{j=1}^{l} w_j val(R_j)} = \frac{f_{\mathcal{S}}(\mathbf{x})}{\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x})} \tag{3.18}$$

$$= \Pr_{\mathcal{S}}(\mathbf{x}) \tag{3.19}$$

where Eq. 3.15 follows from the induction hypothesis, Eq. 3.17 and 3.18 follow from the fact that $val(v) = \sum_{\mathbf{x}|_{\text{scope}(v)}} f_v(\mathbf{x}|_{\text{scope}(v)}), \forall v \in \mathcal{S}$.

This completes the proof since $\Pr_{\mathcal{S}'}(\cdot) = f_{\mathcal{S}'}(\cdot) = \Pr_{\mathcal{S}}(\cdot)$. □

Given a complete and decomposable SPN $\mathcal{S}$, I now construct and show that the last condition in Def. 9 can be satisfied in time and space $O(|\mathcal{S}|)$.

Lemma 10. Given a complete and decomposable SPN $\mathcal{S}$, there exists an SPN $\mathcal{S}'$ satisfying condition 3 in Def. 9 such that $\Pr_{\mathcal{S}'}(\cdot) = \Pr_{\mathcal{S}}(\cdot)$ and $|\mathcal{S}'| = O(|\mathcal{S}|)$.

Proof. I give a proof by construction. First, if $\mathcal{S}$ is not weight normalized, apply Alg. 5 to normalize the weights (i.e., the weights of the edges emanating from each sum node sum to 1).

Now check each sum node v in $\mathcal{S}$ in a bottom-up order. If $|\text{scope}(v)| = 1$, by Corollary 8 I know the network polynomial f_v is a probability distribution over its scope, say, $\{X\}$. Reduce v into a terminal node which is a distribution over X induced by its network polynomial and disconnect v from all its children. The last step is to remove all the unreachable nodes from $\mathcal{S}$ to obtain $\mathcal{S}'$. Note that in this step I will only decrease the size of $\mathcal{S}$, hence $|\mathcal{S}'| = O(|\mathcal{S}|)$. □

Proof of Thm. 5. The combination of Lemma 6, 9 and 10 completes the proof of Thm. 5. □

An example of a normal SPN constructed from the SPN in Fig. 2.8 is depicted in Fig. 3.2.

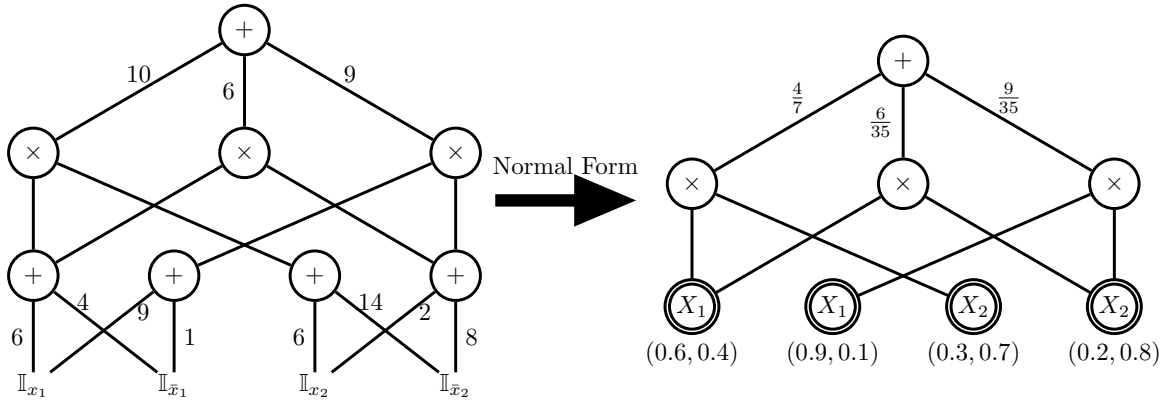


Figure 3.2: Transform an SPN into a normal form. Terminal nodes which are probability distributions over a single variable are represented by a double-circle.

3.2 Sum-Product Network to Bayesian Network

In order to construct a BN from an SPN, I require the SPN to be in a normal form, otherwise I can first transform it into a normal form using Alg. 4 and 5.

Let $\mathcal{S}$ be a normal SPN over $\mathbf{X}_{1:N}$. Before showing how to construct a corresponding BN, I 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, I have $\sum_{i=1}^{l} w_i = 1$ and $w_i \geq 0, \forall i \in 1 : l$. This naturally suggests that I 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, I 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 11. 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

$$\Pr_{\mathcal{S}}(\mathbf{x}|_{\text{scope}(p)} \mid H_{v_1} = v_1^*, \ldots, H_{v_k} = v_k^*) = \prod_{i=1}^{l} \Pr_{\mathcal{S}}(\mathbf{x}|_{\text{scope}(p_i)} \mid H_{v_1} = v_1^*, \ldots, H_{v_k} = v_k^*) \tag{3.20}$$

where $H_v = v^*$ means the sum node v selects its v^*th branch.

Proof. Consider the sub-SPN $\mathcal{S}_p$ rooted at p. $\mathcal{S}_p$ can be obtained by restricting $H_{v_1} = v_1^*, \ldots, H_{v_k} = v_k^*$, i.e., going from the root of $\mathcal{S}$ along the path $H_{v_1} = v_1^*, \ldots, H_{v_k} = v_k^*$. Since p is a decomposable product node, $\mathcal{S}_p$ admits the above factorization by the definition of a product node and Corollary 8. □

Note that there may exist multiple paths from the root to p in $\mathcal{S}$. Each such path admits the factorization stated in Eq. 3.20. Eq. 3.20 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 3.20. 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, I now proceed to prove the first main result in this thesis: Thm. 2. First, I present the algorithm to construct the structure of a BN $\mathcal{B}$ from $\mathcal{S}$ in Alg. 6. In a nutshell, Alg. 6 creates an observable variable 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. 6 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. 6 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 $\mathcal{S}$.

I now present Alg. 7 and 8 to build ADDs for each observable variable X and hidden variable H in $\mathcal{B}$. For each hidden variable H, Alg. 8 builds $\mathcal{A}_H$ as a decision stump[1] obtained by finding H and its associated weights in $\mathcal{S}$. Consider ADDs built by Alg. 7 for observable variables Xs. Let X be the current observable variable I am considering. Basically, Alg. 7 is a recursive algorithm applied to each node in $\mathcal{S}$ whose scope intersects with $\{X\}$. There are three cases. If current node is a terminal node, then it must be a probability distribution over X. In this case I simply return the decision stump at the current node. If the current node is a sum node, then due

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

Algorithm 6 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
```

to the completeness of $\mathcal{S}$, I know that all the children of R share the same scope with R. I first create a node H_R corresponding to the hidden variable associated with R into $\mathcal{A}_X$ (Line 8) and recursively apply Alg. 7 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 $\mathcal{S}$, I know that there will be a unique child of R whose scope intersects with $\{X\}$. I recursively apply Alg. 7 to this child and return the resulting ADD (Lines 12-15).

Equivalently, Alg. 7 can be understood in the following way: *I extract the sub-SPN induced by* $\{X\}$ *and contract*[2] *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. I illustrate the application of Alg. 6, 7 and 8 on the normal SPN in Fig. 3.2, which results in the BN $\mathcal{B}$ with CPDs represented by ADDs shown in Fig. 3.3.

[2]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.

Algorithm 7 Build CPD using ADD, observable variable

Input: normal SPN $\mathcal{S}$, variable X
Output: ADD $\mathcal{A}_X$

if ADD has already been created for $\mathcal{S}$ and X **then**
  $\mathcal{A}_X \leftarrow$ retrieve ADD from cache
else
  $R \leftarrow$ root of $\mathcal{S}$
  if R is a terminal node **then**
    $\mathcal{A}_X \leftarrow$ decision stump rooted at R
  else if R is a sum node **then**
    Create a node H_R into $\mathcal{A}_X$
    for each $R_i \in Ch(R)$ **do**
      Link BuildADD$(\mathcal{S}_{R_i}, X)$ as ith child of H_R
    end for
  else if R is a product node **then**
    Find child $\mathcal{S}_{R_i}$ such that $X \in$ scope(R_i)
    $\mathcal{A}_X \leftarrow$ BuildADD$(\mathcal{S}_{R_i}, X)$
  end if
  store $\mathcal{A}_X$ in cache
end if

I now show that $\Pr_{\mathcal{S}}(\mathbf{x}) = \Pr_{\mathcal{B}}(\mathbf{x}) \quad \forall \mathbf{x}$.

Lemma 12. Given a normal SPN $\mathcal{S}$, the ADDs constructed by Alg. 7 and 8 encode local CPDs at each node in $\mathcal{B}$.

Proof. It is easy to verify that for each hidden variable H in $\mathcal{B}$, $\mathcal{A}_H$ represents a local CPD since $\mathcal{A}_H$ is a decision stump with normalized weights.

For any observable variable X in $\mathcal{B}$, let $Pa(X)$ be the set of parents of X. By Alg. 6, every node in $Pa(X)$ is a hidden variable. Furthermore, $\forall H$, $H \in Pa(X)$ iff there exists one terminal

Algorithm 8 Build CPD using ADD, hidden variable

Input: normal SPN $\mathcal{S}$, variable H
Output: ADD $\mathcal{A}_H$

Find the sum node H in $\mathcal{S}$
$\mathcal{A}_H \leftarrow$ decision stump rooted at H in $\mathcal{S}$

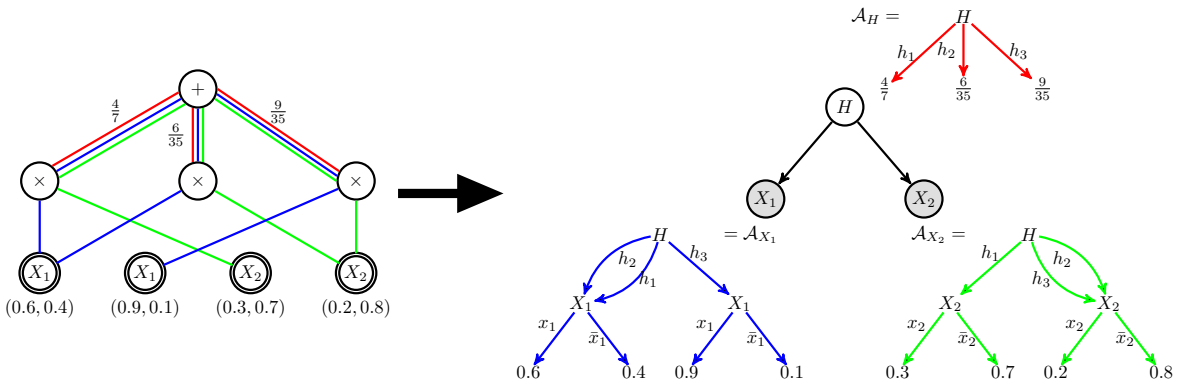


Figure 3.3: Construct a BN with CPDs represented by ADDs from an SPN. On the left, the induced sub-SPNs used to create $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ by Alg. 7 are indicated in blue and green respectively. The decision stump used to create $\mathcal{A}_H$ by Alg. 8 is indicated in red.

node over X in $\mathcal{S}$ that appears in the sub-SPN rooted at H. Hence given any joint assignment $\mathbf{h}$ of $Pa(X)$, there will be a path in $\mathcal{A}_X$ from the root to a terminal node that is consistent with the joint assignment of the parents. Also, the leaves in $\mathcal{A}_X$ contain normalized weights corresponding to the probabilities of X (see Def. 9) induced by the creation of decision stumps over X in Lines 5-6 of Alg. 7. □

Theorem 13. For any normal SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$, the BN $\mathcal{B}$ constructed by Alg. 6, 7 and 8 encodes the same probability distribution, i.e., $\Pr_{\mathcal{S}}(\mathbf{x}) = \Pr_{\mathcal{B}}(\mathbf{x}), \forall \mathbf{x}$.

Proof. Again, I give a proof by induction on the height of $\mathcal{S}$.

- Base case. The height of SPN $\mathcal{S}$ is 0. In this case, $\mathcal{S}$ will be a single terminal node over X and $\mathcal{B}$ will be a single observable node with decision stump $\mathcal{A}_X$ constructed from the terminal node by Lines 5-6 in Alg. 7. It is clear that $\Pr_{\mathcal{S}}(x) = \Pr_{\mathcal{B}}(x), \forall x$.
- Induction step. Assume $\Pr_{\mathcal{B}}(\mathbf{x}) = \Pr_{\mathcal{S}}(\mathbf{x}), \forall \mathbf{x}$ for any $\mathcal{S}$ with height $\leq k$, where $\mathcal{B}$ is the corresponding BN constructed by Alg. 6, 7 and 8 from $\mathcal{S}$. Consider an SPN $\mathcal{S}$ with height $k+1$. Let R be the root of $\mathcal{S}$ and $R_i, i \in 1 : l$ be the children of R in $\mathcal{S}$. I consider the following two cases:
  - R is a product node. Let $\text{scope}(R_t) = \mathbf{X}_t, t \in 1 : l$. Claim: there is no edge between $\mathcal{S}_i$ and $\mathcal{S}_j, i \neq j$, where $\mathcal{S}_i(\mathcal{S}_j)$ is the sub-SPN rooted at $R_i(R_j)$. If there is

an edge, say, from v_j to v_i where $v_j \in \mathcal{S}_j$ and $v_i \in \mathcal{S}_i$, then $\text{scope}(v_i) \subseteq \text{scope}(v_j) \subseteq \text{scope}(R_j)$. On the other hand, $\text{scope}(v_i) \subseteq \text{scope}(R_i)$. So I have $\varnothing \neq \text{scope}(v_i) \subseteq \text{scope}(R_i) \bigcap \text{scope}(R_j)$, which contradicts the decomposability of the product node R. Hence the constructed BN $\mathcal{B}$ will be a forest of l disconnected components, and each component $\mathcal{B}_t$ will correspond to the sub-SPN $\mathcal{S}_t$ rooted at $R_t, \forall t \in 1 : l$, with height $\leq k$. By the induction hypothesis I have $\Pr_{\mathcal{B}_t}(\mathbf{x}_t) = \Pr_{\mathcal{S}_t}(\mathbf{x}_t), \forall t \in 1 : l$. Consider the whole BN $\mathcal{B}$, I have:

$$\Pr_{\mathcal{B}}(\mathbf{x}) = \prod_t \Pr_{\mathcal{B}_t}(\mathbf{x}_t) = \prod_t \Pr_{\mathcal{S}_t}(\mathbf{x}_t) = \Pr_{\mathcal{S}}(\mathbf{x}) \tag{3.21}$$

where the first equation is due to the d-separation rule in BNs by noting that each component $\mathcal{B}_t$ is disconnected from all other components. The second equation follows from the induction hypothesis. The last equation follows from the definition of a product node.

– R is a sum node. In this case, due to the completeness of $\mathcal{S}$, all the children of R share the same scope as R. By the construction process presented in Alg. 6, 7 and 8, there is a hidden variable H corresponding to R that takes l different values in $\mathcal{B}$. Let $w_{1:l}$ be the weights of the edges emanating from R in $\mathcal{S}$. For the tth branch of R, I use $\mathbf{H}_t$ to denote the set of hidden variables in $\mathcal{B}$ that also appear in $\mathcal{B}_t$, and let $\mathbf{H}_{-t} = \mathbf{H} \backslash \mathbf{H}_t$, where $\mathbf{H}$ is the set of all hidden variables in $\mathcal{B}$ except H. First, I show the following identity:

$$\Pr_{\mathcal{B}}(\mathbf{x}|H = h_t) = \sum_{\mathbf{h}_t} \sum_{\mathbf{h}_{-t}} \Pr_{\mathcal{B}}(\mathbf{x}, \mathbf{h}_t, \mathbf{h}_{-t}|H = h_t) \tag{3.22}$$

$$= \sum_{\mathbf{h}_t} \sum_{\mathbf{h}_{-t}} \Pr_{\mathcal{B}}(\mathbf{x}, \mathbf{h}_t|H = h_t, \mathbf{h}_{-t}) \Pr_{\mathcal{B}}(\mathbf{h}_{-t}|H = h_t) \tag{3.23}$$

$$= \sum_{\mathbf{h}_t} \sum_{\mathbf{h}_{-t}} \Pr_{\mathcal{B}}(\mathbf{x}, \mathbf{h}_t|H = h_t) \Pr_{\mathcal{B}}(\mathbf{h}_{-t}|H = h_t) \tag{3.24}$$

$$= \sum_{\mathbf{h}_t} \Pr_{\mathcal{B}}(\mathbf{x}, \mathbf{h}_t|H = h_t) \sum_{\mathbf{h}_{-t}} \Pr_{\mathcal{B}}(\mathbf{h}_{-t}|H = h_t) \tag{3.25}$$

$$= \sum_{\mathbf{h}_t} \Pr_{\mathcal{B}}(\mathbf{x}, \mathbf{h}_t|H = h_t) \tag{3.26}$$

$$= \sum_{\mathbf{h}_t} \Pr_{\mathcal{B}_t}(\mathbf{x}, \mathbf{h}_t) = \Pr_{\mathcal{B}_t}(\mathbf{x}) \tag{3.27}$$

Using this identity, I have

$$\Pr_{\mathcal{B}}(\mathbf{x}) = \sum_{t=1}^{l} \Pr_{\mathcal{B}}(h_t) \Pr_{\mathcal{B}}(\mathbf{x}|H = h_t) \tag{3.28}$$

$$= \sum_{t=1}^{l} w_t \Pr_{\mathcal{B}_t}(\mathbf{x}) \tag{3.29}$$

$$= \sum_{t=1}^{l} w_t \Pr_{\mathcal{S}_t}(\mathbf{x}) \tag{3.30}$$

$$= \Pr_{\mathcal{S}}(\mathbf{x}) \tag{3.31}$$

Eq. 3.24 follows from the fact that $\mathbf{X}$ and $\mathbf{H}_t$ are independent of $\mathbf{H}_{-t}$ given $H = h_t$, i.e., I take advantage of the CSI described by ADDs of $\mathbf{X}$. Eq. 3.25 follows from the fact that $\mathbf{H}_{-t}$ appears only in the second term. Combined with the fact that $H = h_t$ is given as evidence in $\mathcal{B}$, this gives us the induced subgraph $\mathcal{B}_t$ referred to in Eq. 3.27. Eq. 3.29 follows from Eq. 3.27 and Eq. 3.30 follows from the induction hypothesis.

Combing the base case and the induction step completes the proof for Thm. 13. □

I now bound the size of $\mathcal{B}$:

Theorem 14. $|\mathcal{B}| = O(N|\mathcal{S}|)$, where BN $\mathcal{B}$ is constructed by Alg. 6, 7 and 8 from normal SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$.

Proof. For each observable variable X in $\mathcal{B}$, $\mathcal{A}_X$ is constructed by first extracting from $\mathcal{S}$ the induced sub-SPN $\mathcal{S}_X$ that contains all nodes whose scope includes X and then contracting all the product nodes in $\mathcal{S}_X$ to obtain $\mathcal{A}_X$. By the decomposability of product nodes, each product node in $\mathcal{S}_X$ has out-degree 1 otherwise the original SPN $\mathcal{S}$ violates the decomposability property. Since contracting product nodes does not increase the number of edges in $\mathcal{S}_X$, I have $|\mathcal{A}_X| \leq |\mathcal{S}_X| \leq |\mathcal{S}|$.

For each hidden variable H in $\mathcal{B}$, $\mathcal{A}_H$ is a decision stump constructed from the internal sum node corresponding to H in $\mathcal{S}$. Hence, I have $\sum_H \mathcal{A}_H \leq |\mathcal{S}|$.

Now consider the size of the graph $\mathcal{B}$. Note that only terminal nodes and sum nodes will have corresponding variables in $\mathcal{B}$. It is clear that the number of nodes in $\mathcal{B}$ is bounded by the number of nodes in $\mathcal{S}$. Furthermore, a hidden variable H points to an observable variable X in $\mathcal{B}$ iff X appears in the sub-SPN rooted at H in $\mathcal{S}$, i.e., there is a path from the sum node corresponding to

H to one of the terminal nodes in X. For a sum node H (which corresponds to a hidden variable $H \in \mathcal{B}$) with scope size s, each edge emanated from H in $\mathcal{S}$ will correspond to directed edges in $\mathcal{B}$ at most s times, since there are exactly s observable variables which are children of H in $\mathcal{B}$. It is clear that $s \leq N$, so each edge emanated from a sum node in $\mathcal{S}$ will be counted at most N times in $\mathcal{B}$. Edges from product nodes will not occur in the graph of $\mathcal{B}$, instead, they have been counted in the ADD representations of the local CPDs in $\mathcal{B}$. So again, the size of the graph $\mathcal{B}$ is bounded by $\sum_H \text{scope}(H) \times \deg(H) \leq \sum_H N \deg(H) \leq 2N|\mathcal{S}|$.

There are N observable variables in $\mathcal{B}$. So the total size of $\mathcal{B}$, including the size of the graph and the size of all the ADDs, is bounded by $N|\mathcal{S}| + |\mathcal{S}| + 2N|\mathcal{S}| = O(N|\mathcal{S}|)$. □

I give the time complexity of Alg. 6, 7 and 8.

Theorem 15. For any normal SPN $\mathcal{S}$ over $\mathbf{X}_{1:N}$, Alg. 6, 7 and 8 construct an equivalent BN in time $O(N|\mathcal{S}|)$.

Proof. First consider Alg. 6. Alg. 6 recursively visits each node and its children in $\mathcal{S}$ if they have not been visited (Lines 6-10). For each node v in $\mathcal{S}$, Lines 7-9 cost at most $2 \cdot \text{out-degree}(v)$. If v is a sum node, then Lines 11-17 create a hidden variable and then connect the hidden variable to all observable variables that appear in the sub-SPN rooted at v, which is clearly bounded by the number of all observable variables, N. So the total cost of Alg. 6 is bounded by $\sum_v 2 \cdot \text{out-degree}(v) + \sum_{v \text{ is a sum node}} N \leq 2\mathfrak{V}(\mathcal{S}) + 2\mathfrak{E}(\mathcal{S}) + N\mathfrak{V}(\mathcal{S}) \leq 2|\mathcal{S}| + N|\mathcal{S}| = O(N|\mathcal{S}|)$. Note that I assume that inserting an element into a set can be done in $O(1)$ by using hashing.

The analysis for Alg. 7 and 8 follows the same analysis as in the proof for Thm. 14. The time complexity for Alg. 7 and Alg. 8 is then bounded by $N|\mathcal{S}| + |\mathcal{S}| = O(N|\mathcal{S}|)$. □

Proof of Thm. 2. The combination of Thm. 13, 14 and 15 proves Thm. 2. □

Proof of Corollary. 3. Given a complete and consistent SPN $\mathcal{S}$, I can first transform it into a normal SPN $\mathcal{S}'$ with $|\mathcal{S}'| = O(|\mathcal{S}|^2)$ by Thm. 5 if it is not normal. After this the analysis follows from Thm. 2. □

3.3 Bayesian Network to Sum-Product Network

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. As a result, the generated SPN loses its power to compactly represent some BNs with high tree-width, yet, with CSI in its local CPDs.

Alternatively, one can also compile a BN with ADDs into an arithmetic circuit (AC) [5] and then convert an AC into an SPN [21]. However, in [5]'s compilation approach, the variables appearing along a path from the root to a leaf in each ADD must be consistent with a pre-defined global variable ordering. The global variable ordering, may, to some extent restrict the compactness of ADDs as the most compact representation for different ADDs normally have different topological orderings. Interested readers are referred to [5] for more details on this topic.

In this section, I focus on BNs with ADDs that are constructed using Alg. 7 and 8 from normal SPNs. I show that when applying VE to those BNs with ADDs I 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*. More specifically, since all the ADDs constructed using Alg. 7 are induced sub-SPNs with contraction of product nodes from the original SPN $\mathcal{S}$, the topological ordering of all the nodes in $\mathcal{S}$ can be used as the pre-defined variable ordering for all the ADDs.

Algorithm 9 Multiplication of two symbolic ADDs, $\otimes$

Input: Symbolic ADD $\mathcal{A}_{X_1}$, $\mathcal{A}_{X_2}$
Output: Symbolic ADD $\mathcal{A}_{X_1,X_2} = \mathcal{A}_{X_1} \otimes \mathcal{A}_{X_2}$

$R_1 \leftarrow$ root of $\mathcal{A}_{X_1}$, $R_2 \leftarrow$ root of $\mathcal{A}_{X_2}$
if R_1 and R_2 are both variable nodes **then**
  if $R_1 = R_2$ **then**
    Create a node $R = R_1$ into $\mathcal{A}_{X_1,X_2}$
    for each $r \in dom(R)$ **do**
      $\mathcal{A}^r_{X_1} \leftarrow Ch(R_1)|_r$
      $\mathcal{A}^r_{X_2} \leftarrow Ch(R_2)|_r$
      $\mathcal{A}^r_{X_1,X_2} \leftarrow \mathcal{A}^r_{X_1} \otimes \mathcal{A}^r_{X_2}$
      Link $\mathcal{A}^r_{X_1,X_2}$ to the rth child of R in $\mathcal{A}_{X_1,X_2}$
    end for
  else
    $\mathcal{A}_{X_1,X_2} \leftarrow$ create a symbolic node $\otimes$
    Link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as two children of $\otimes$
  end if
else if R_1 is a variable node and R_2 is $\otimes$ **then**
  if R_1 appears as a child of R_2 **then**
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2}$
    $\mathcal{A}^{R_1}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1} \otimes \mathcal{A}^{R_1}_{X_2}$
  else
    Link $\mathcal{A}_{X_1}$ as a new child of R_2
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2}$
  end if
else if R_1 is $\otimes$ and R_2 is a variable node **then**
  if R_1 appears as a child of R_2 **then**
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1}$
    $\mathcal{A}^{R_2}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2} \otimes \mathcal{A}^{R_2}_{X_1}$
  else
    Link $\mathcal{A}_{X_2}$ as a new child of R_1
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1}$
  end if
else
  $\mathcal{A}_{X_1,X_2} \leftarrow$ create a symbolic node $\otimes$
  Link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as two children of $\otimes$
end if
Merge connected product nodes in $\mathcal{A}_{X_1,X_2}$

Algorithm 10 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 obtained from $\mathcal{A}_H$
3: Replace H by a symbolic $\oplus$ node
4: **end if**

In order to apply VE to a BN with ADDs, I 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 my purpose, I 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 [5] in their compilation approach. For example, given symbolic ADDs $\mathcal{A}_{X_1}$ over X_1 and $\mathcal{A}_{X_2}$ over X_2, Alg. 9 returns a 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, I 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.

Given two symbolic ADDs $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$, Alg. 9 recursively visits nodes in $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ simultaneously. In general, there are 3 cases: 1) the roots of $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ are both variable nodes (Lines 2-14); 2) one of the two roots is a variable node and the other is a product node (Lines 15-30); 3) both roots are product nodes or at least one of them is a sum node (Lines 31-34). I discuss these 3 cases.

If both roots of $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ are variable nodes, there are two subcases to be considered. First, if they are nodes labeled with the same variable (Lines 3-10), then the computation related to the common variable is shared and the multiplication is recursively applied to all the children, otherwise I simply create a symbolic product node $\otimes$ and link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as its two children (Lines 11-14). Once I find $R_1 \in \mathcal{A}_{X_1}$ and $R_2 \in \mathcal{A}_{X_2}$ such that $R_1 \neq R_2$, there will be no common node that is shared by the sub-ADDs rooted at R_1 and R_2. To see this, note that Alg. 9 recursively calls itself as long as the roots of $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ are labeled with the same variable. Let R be the last variable shared by the roots of $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ in Alg. 9. Then R_1 and R_2 must be the children of R in the original SPN $\mathcal{S}$. Since R_1 does not appear in $\mathcal{A}_{X_2}$, then $X_2 \notin \text{scope}(R_1)$, otherwise R_1 will occur in $\mathcal{A}_{X_2}$ and R_1 will be a new shared variable below R, which is a contradiction to the fact that R is the last shared variable. Since R_1 is the root of the sub-ADD of $\mathcal{A}_{X_1}$ rooted at

R, hence no variable whose scope contains X_2 will occur as a descendant of R_1, otherwise the scope of R_1 will also contain X_2, which is again a contradiction. On the other hand, each node appearing in $\mathcal{A}_{X_2}$ corresponds to a variable whose scope intersects with $\{X_2\}$ in the original SPN, hence no node in $\mathcal{A}_{X_2}$ will appear in $\mathcal{A}_{X_1}$. The same analysis also applies to R_2. Hence no node will be shared between $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$.

If one of the two roots, say, R_1, is a variable node and the other root, say, R_2, is a product node, then I consider two subcases. If R_1 appears as a child of R_2 then I recursively multiply R_1 with the child of R_2 that is labeled with the same variable as R_1 (Lines 16-18). If R_1 does not appear as a child of R_2, then I link the ADD rooted at R_1 to be a new child of the product node R_2 (Lines 19-22). Again, let R be the last shared node between $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ during the multiplication process. Then both R_1 and R_2 are children of R, which corresponds to a sum node in the original SPN $\mathcal{S}$. Furthermore, both R_1 and R_2 lie in the same branch of R in $\mathcal{S}$. In this case, since $\text{scope}(R_1) \subseteq \text{scope}(R)$, $\text{scope}(R_1)$ must be a strict subset of $\text{scope}(R)$ otherwise I would have $\text{scope}(R_1) = \text{scope}(R)$ and R_1 will also appear in $\mathcal{A}_{X_2}$, which contradicts the fact that R is the last shared node between $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$. Hence here I only need to discuss the two cases where either their scope disjoint (Line 16-18) or the scope of one root is a strict subset of another (Line 19-22).

If the two roots are both product nodes or at least one of them is a sum node, then I simply create a new product node and link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ to be children of the product node. The above analysis also applies here since sum nodes in symbolic ADD are created by summing out processed variable nodes and I eliminate all the hidden variables using the inverse topological ordering.

The last step in Alg. 9 (Line 35) simplifies the symbolic ADD by merging all the connected product nodes without changing the function it encodes. This can be done in the following way: suppose $\otimes_1$ and $\otimes_2$ are two connected product nodes in symbolic ADD $\mathcal{A}$ where $\otimes_1$ is the parent of $\otimes_2$, then I can remove the link between $\otimes_1$ and $\otimes_2$ and connect $\otimes_1$ to every child of $\otimes_2$. It is easy to verify that such an operation will remove links between connected product nodes while keeping the encoded function unchanged.

To sum-out one hidden variable H, Alg. 10 simply replaces H in $\mathcal{A}$ by a symbolic sum node $\oplus$ and labels each edge of $\oplus$ with weights obtained from $\mathcal{A}_H$.

I now present the Variable Elimination (VE) algorithm in Alg. 11 used to recover the original SPN $\mathcal{S}$, taking Alg. 9 and Alg. 10 as two operations $\otimes$ and $\oplus$ respectively.

In each iteration of Alg. 11, I select one hidden variable H in ordering π, multiply all the ADDs $\mathcal{A}_X$ in which H appears using Alg. 9 and then sum-out H using Alg. 10. 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 $\mathcal{B}$.

Algorithm 11 Variable Elimination for BN with ADDs

Input: BN $\mathcal{B}$ with ADDs for all observable variables and hidden variables
Output: Original SPN $\mathcal{S}$

1: $\pi \leftarrow$ the inverse topological ordering of all the hidden variables present in the ADDs
2: $\Phi \leftarrow \{\mathcal{A}_X \mid X \text{ is an observable variable}\}$
3: **for** each hidden variable H in π **do**
4: $\quad P \leftarrow \{\mathcal{A}_X \mid H \text{ appears in } \mathcal{A}_X\}$
5: $\quad \Phi \leftarrow \Phi \backslash P \cup \{\oplus_H \otimes_{\mathcal{A} \in P} \mathcal{A}\}$
6: **end for**
7: **return** Φ

Note that the SPN returned by Alg. 11 may not be literally equal to the original SPN since during the multiplication of two symbolic ADDs I effectively remove redundant nodes by merging connected product nodes. Hence, the SPN returned by Alg. 11 could have a smaller size while representing the same probability distribution. An example is given in Fig. 3.4 to illustrate the recovery process. The BN in Fig. 3.4 is the one constructed in Fig. 3.3.

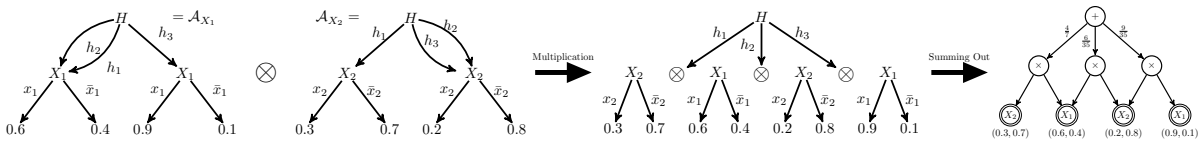


Figure 3.4: Multiply $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ that contain H using Alg. 9 and then sum out H by applying Alg. 10. The final SPN is isomorphic with the SPN in Fig. 3.3.

Note that Alg. 9 and 10 apply only to ADDs constructed from normal SPNs by Alg. 7 and 8 because such ADDs naturally inherit the topological ordering of sum nodes (hidden variables) in the original SPN $\mathcal{S}$. Otherwise I 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 pre-defined ordering. Note also that Alg. 9 and 10 should be implemented with caching of repeated operations in order to ensure that directed acyclic graphs are preserved. Alg. 11 suggests that an SPN can also be viewed as a *history record* or *caching* of the sums and products computed during inference when applied to the resulting BN with ADDs.

I now bound the run time of Alg. 11.

Theorem 16. Alg. 11 builds SPN $\mathcal{S}$ from BN $\mathcal{B}$ with ADDs in $O(N|\mathcal{S}|)$.

Proof. First, it is easy to verify that Alg. 9 takes at most $|\mathcal{A}_{X_1}| + |\mathcal{A}_{X_2}|$ operations to compute the multiplication of $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$. More importantly, the size of the generated $\mathcal{A}_{X_1,X_2}$ is also bounded by $|\mathcal{S}|$. This is because all the common nodes and edges in $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ are shared (not duplicated) in $\mathcal{A}_{X_1,X_2}$. Also, all the other nodes and edges which are not shared between $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ will be in two branches of a product node in $\mathcal{S}$, otherwise they will be shared by $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as they have the same scope which contain both X_1 and X_2. This means that $\mathcal{A}_{X_1,X_2}$ can be viewed as a sub-SPN of $\mathcal{S}$ induced by the node set $\{X_1, X_2\}$ with some product nodes contracted out. So I have $|\mathcal{A}_{X_1,X_2}| \leq |\mathcal{S}|$.

Now consider the for loop (Lines 3-6) in Alg. 11. The loop ends once I've summed out all the hidden variables and there is only one ADD left. Note that there may be only one ADD in Φ during some intermediate steps, in which case I do not have to do any multiplication. In such steps, I only need to perform the sum out procedure without multiplying ADDs. Since there are N ADDs at the beginning of the loop and after the loop I only have one ADD, then there is exactly $N-1$ multiplications during the for loop, which costs at most $(N-1)|\mathcal{S}|$ operations. Furthermore, in each iteration there is exactly one hidden variable being summed out. So the total cost for summing out all the hidden variables in Lines 3-6 is bounded by $|\mathcal{S}|$.

Overall, the operations in Alg. 11 are bounded by $(N-1)|\mathcal{S}| + |\mathcal{S}| = O(N|\mathcal{S}|)$. □

Proof of Thm. 4. Thm. 16 and the analysis above prove Thm. 4. □

Chapter 4

Discussion

Thm. 2 together with Thm. 4 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 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. 3.20 can also be encoded explicitly with ADDs in a BN. I 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 quickly. 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. Indeed, my constructive algorithm gives us a way to relate the depth of an SPN and the tree-width of the corresponding BN. Without loss of generality, let's assume that product layers alternate with sum layers in the SPN I am considering. Let the height of the SPN, i.e., the longest path from the root to a terminal node, be K. By my assumption, there will be at least $\lfloor K/2 \rfloor$ sum nodes in the longest path. Accordingly, in the BN constructed by Alg. 6, 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. I 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 constructed BN and the more complex are the probability distributions that can be encoded*. This observation is consistent with the conclusion drawn in [7] 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. I reinforce this observation and make it more precise by pointing out that the depth of an SPN has a direct impact on the tree-width of the corresponding BN, which decides the set of probability distributions that can be represented compactly.

In the literature, 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 help to 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.

Although I mainly focus on the relationship between SPNs and BNs, it is straightforward to extend the conclusions obtained in this thesis to the broader family of probabilistic graphical models, including both BNs and MNs. The key observation here lies in the fact that I can always convert a given BN into an MN using *moralization* [11]. Again, for the potential function associated with each maximal clique in MN, I can achieve it by multiplying the corresponding ADDs using Alg. 9. Here I give the following algorithm used to create both the structure and the potential functions of an MN given an SPN:

Algorithm 12 SPN to MN

Input: normal SPN $\mathcal{S}$
Output: MN $\mathcal{M}$

Apply Alg. 6, Alg. 7 and Alg. 8 to create an BN $\mathcal{B}$ from $\mathcal{S}$
$\mathcal{M} \leftarrow \mathcal{B}$
// Moralization
for each observable variable $X_n \in \mathcal{M}$ **do**
  $H_{X_n} \leftarrow \{H | H \in \mathcal{M}_V, (H, X_n) \in \mathcal{M}_E\}$
  for $H_i, H_j \in H_{X_n}, i \neq j$ **do**
    Create an undirected edge between H_i and H_j
  end for
end for
Change all the directed edges in $\mathcal{M}$ to undirected edges
// Build potential function
for each observable variable $X_n \in \mathcal{M}$ **do**
  $\mathcal{A}_{X_n} \leftarrow \{\mathcal{A}_H | H \in \mathcal{M}_V, (H, X_n) \in \mathcal{M}_E\} \bigcup \{\mathcal{A}_{X_n}\}$
  $\psi_{X_n} \leftarrow \otimes \mathcal{A}_{X_n}$
end for
return $\mathcal{M}$ with potentials represented in ADD $\psi_{X_n}, \forall n$

In summary, I create an MN from an SPN by first converting the SPN into a BN, and then I convert the BN into an MN. Here again, the simple bipartite graphical structure of the corresponding BN helps me to count and construct all the maximal cliques and their corresponding potential functions by enumerating all the observable variables since there is a one-to-one correspondence between maximal cliques and observable variables in the directed bipartite BN.

I now bound the size of the constructed MN, including both the size of all the potential functions, $\psi_{X_n}, \forall n$, and the size of the graph, $|\mathcal{M}|$. Let's first consider the size of all the potential functions. Since there is a one-to-one correspondence between observable variables and maximal cliques in the graph, I count all the potential functions by enumerating all the observable variables. Let X_n be an observable variable and $\mathcal{A}_{X_n}$ be its ADD. Lines 13-14 in Alg. 12 collect both the ADD of X_n and its neighbors and then multiply them into a single ADD using Alg. 9. As analyzed before, the size of the resulting ADD is bounded by the sum of sizes of the individual ADDs, which is given by $|\mathcal{A}_{X_n}| + \sum_H \mathcal{A}_H \leq |\mathcal{S}| + |\mathcal{S}| = 2|\mathcal{S}|$. Since there are N observable variables, the total size of the potential functions encoded in ADDs is bounded by $N \times 2|\mathcal{S}| = O(N|\mathcal{S}|)$. To bound the size of the graph $\mathcal{M}$, I only need to consider the number of undirected edges added during the moralization step since we know the size of $\mathcal{B}$ is bounded by $O(N|\mathcal{S}|)$. Let V_S be the total number of sum nodes in the original SPN. In the worst case, there will be exactly $V_S + 1$ nodes in each maximal clique in $\mathcal{M}$, hence the edges added during the moralization is V_S^2 for each maximal clique, leading to a bound on the total number of edges added of $O(NV_S^2)$. In all, the size of the graph $\mathcal{M}$ is $O(N|\mathcal{S}| + NV_S^2)$. So the total size of $\mathcal{M}$, including both the size of the graph and the size of all potential functions, is bounded by $N|\mathcal{S}| + N|\mathcal{S}| + NV_S^2 = O(N|\mathcal{S}| + NV_S^2)$. Again, I construct an MN whose size is bounded by a polynomial function of the size of the original SPN, which indicates that SPNs are no more powerful than MNs.

So far I have presented constructive algorithms to convert an SPN into a BN and/or an MN whose size is bounded by a polynomial function of the original SPN. The other direction is clear: given a BN or MN, we can first convert it into a junction tree and then translate the junction tree into an SPN. The size of the resulting SPN is exponential in the tree-width of the original BN/MN, which again, indicates that inference is equally efficient in SPNs and probabilistic graphical models (PGMs).

Chapter 5

Conclusion

In this thesis, I establish a precise connection between BNs and SPNs by providing constructive algorithms to transform between these two models. To simplify the proof, I introduce the notion of normal SPN and describe the relationship between consistency and decomposability in SPNs. I analyze the impact of the depth of SPNs onto the tree-width of the corresponding BNs. My 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. In the resulting BNs, correlations are not expressed by links directly between observed variables, but rather through hidden variables that are ancestors of correlated observed variables. The structure of the resulting BNs can be used to study probabilistic dependencies and causal relationships between the variables of the original SPNs. It would also be interesting to explore the opposite direction since there is already a large literature on parameter and structure learning for BNs. One could learn a BN from data and then exploit CSI to convert it into an SPN.

References

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

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

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

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

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

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

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

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

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

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

[11] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[12] Daniel Lowd and Amirmohammad Rooshenas. Learning markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 406–414, 2013.

[13] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[14] Giulia Pagallo. Learning DNF by decision trees. In *IJCAI*, volume 89, pages 639–644, 1989.

[15] Judea Pearl. *Causality: models, reasoning and inference*, volume 29. Cambridge Univ Press, 2000.

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

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

[18] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, Pedro Domingos, and BEEPRI BioTechMed-Graz. On theoretical properties of sum-product networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 744–752, 2015.

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

[20] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

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

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

[23] Sameer Singh, Michael Wick, and Andrew McCallum. Monte carlo mcmc: efficient inference by approximate sampling. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1104–1113. Association for Computational Linguistics, 2012.

[24] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

[25] Nevin L Zhang and David Poole. A simple approach to bayesian network computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 1994.

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

[27] Jun Zhu, Jianfei Chen, and Wenbo Hu. Big learning with bayesian methods. *arXiv preprint arXiv:1411.6370*, 2014.