A Differential Approach to Inference in Bayesian Networks

Adnan Darwiche
Computer Science Department
University of California
Los Angeles, Ca 90095
darwiche@cs.ucla.edu

We present a new approach to inference in Bayesian networks which is based on representing the network using a polynomial and then retrieving answers to probabilistic queries by evaluating and differentiating the polynomial. The network polynomial itself is exponential in size, but we show how it can be computed efficiently using an arithmetic circuit that can be evaluated and differentiated in time and space linear in the circuit size. The proposed framework for inference subsumes one of the most influential methods for inference in Bayesian networks, known as the tree–clustering or jointree method, which provides a deeper understanding of this classical method and lifts its desirable characteristics to a much more general setting. We discuss some theoretical and practical implications of this subsumption.

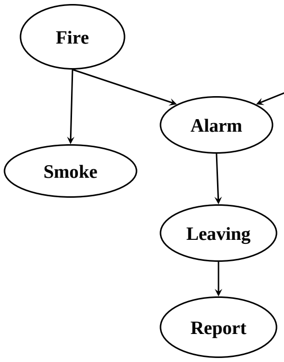
1. INTRODUCTION

A Bayesian network is a compact, graphical model of a probability distribution [Pearl 1988]. It consists of two parts: a directed acyclic graph which represents direct influences among variables, and a set of conditional probability tables that quantify the strengths of these influences. Figure 1 depicts an example Bayesian network relating to a scenario of potential fire in a building. This Bayesian network has six Boolean variables, leading to sixty–four different variable instantiations. The network is interpreted as a complete specification of a probability distribution over these instantiations. And one can easily construct this distribution using the *chain rule* for Bayesian networks which we will discuss later.

Our concern in this paper is with the efficient computation of answers to probabilistic queries posed to Bayesian networks. For example, in Figure 1, we may want to know the probability of fire given that people are reported to be leaving, or the probability of smoke given that the alarm is off. More generally, if a given Bayesian network induces a probability distribution Pr, then we are interested in computing probabilities of events based on the distribution Pr. A brute force approach which constructs the distribution Pr in tabular form and then uses it to answer queries is usually prohibitive as the table size is exponential in the number of variables in the Bayesian network. There has been much research, however, over the last two decades to develop efficient algorithms for inference in Bayesian networks, which are not necessarily exponential in the number of network variables. We review three classes of such algorithms next.

The first class of algorithms is based on the notion of *conditioning,* or case analysis. It is well known that when the value of a network variable is observed, the topology of the network can be simplified by deleting edges that are outgoing from that variable [Pearl 1988; Darwiche 2001]. Even if the variable's value is not observed, one can still exploit this fact by performing a case analysis on the variable. Some conditioning algorithms attempt to reduce the network into a tree structure, which is tractable, leading to what is known as *cutset conditioning* [Pearl 1988]. Other conditioning algorithms attempt to decompose the network into smaller networks that are solved recursively, leading to what is known as *recursive conditioning* [Darwiche 2001]. Conditioning algorithms work by carefully choosing a set of variables on which to perform case analysis and are, therefore, started by some sort of graph theoretic analysis of the given Bayesian network. For example, recursive conditioning starts by building a *dtree* (decomposition tree) which it uses to control case analysis at each level of the recursion [Darwiche 2001].

The second class of algorithms for inference in Bayesian networks is based on the notion of *variable elimination.* The basic idea here is to take a probabilistic model over n variables and reduce it to a model over $n-1$ variables, while maintaining the ability of the model to answer queries of interest [Shachter et al. 1990; Dechter 1996; Zhang and Poole 1996]. The process is repeated until we have a trivial model from which we can look up answers immediately. The complexity of the algorithm is then governed by the amount of work it takes to eliminate a variable, which is known to be very sensitive to the order in which variables are eliminated. Hence, the key step in variable elimination algorithms is the choice of a particular *variable*



	$Fire$	$Pr(Fire)$
	true	.01
	false	.99

	$Tampering$	$Pr(Tampering)$
	true	.02
	false	.98

$Fire$	$Smoke$	$Pr(Smoke\|Fire)$
true	true	.9
true	false	.1
false	true	.01
false	false	.99

$Fire$	$Tampering$	$Alarm$	$Pr(Alarm\|Fire, Tampering)$
true	true	true	.5
true	true	false	.5
true	false	true	.99
true	false	false	.01
false	true	true	.85
false	true	false	.15
false	false	true	.0001
false	false	false	.9999

$Alarm$	$Leaving$	$Pr(Leaving\|Alarm)$
true	true	.88
true	false	.12
false	true	.001
false	false	.999

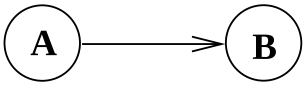
$Leaving$	$Report$	$Pr(Report\|Leaving)$
true	true	.75
true	false	.25
false	true	.01
false	false	.99

Fig. 1. A Bayesian network over six Boolean variables. The network has two parts: a directed acyclic graph over variables of interest, and a conditional probability table (CPT) for each variable in the network. The CPT for a variable provides the distribution of that variable given its parents.

elimination order, which is also based on some graph theoretic analysis of the given Bayesian network.

A third class of algorithms for inference in Bayesian networks is based on the notion of *tree clustering* and capitalizes on the tractibility of inference with respect to tree structures [Shenoy and Shafer 1986; Pearl 1988; Jensen et al. 1990]. This class of algorithms converts the original Bayesian network into a tree structure, known as a *jointree*, and then performs tree–based inference on the resulting jointree. The caveat here is that the jointree is a tree over compound variables, where a compound variable corresponds to a set of variables known as a *cluster*, and the complexity of inference is exponential in the size of such compound variables. Hence, the first step in such algorithms is to build a good jointree, one which minimizes the size of the largest compound variable.

The computational complexity of the three classes of algorithms discussed above can be related through the influential notion of *treewidth* [Bodlaender 1993; Bodlaender 1996], which is a measure of graph connectivity and is defined for both directed and undirected graphs. Suppose we have a Bayesian network with n nodes and bounded treewidth w. Suppose further that our interest is in computing the probability of some instantiation $\mathbf{e}$ of variables $\mathbf{E}$ in the network. One can compute the probability of $\mathbf{e}$ in $O(n \exp(w))$ time and space, using either conditioning, variable elimination or clustering. One of the main benefits of conditioning, however, is that it facilitates the tradeoff between time and space. For example, one can answer the previous query using $O(n)$ space only, but at the expense of $O(n \exp(w \log n))$ time—a complete tradeoff spectrum is also possible [Darwiche 2001]. One of the main benefits of variable elimination is its



A	θ_A
true	$\theta_a = .3$
false	$\theta_{\bar{a}} = .7$

A	B	$\theta_{B\|A}$
true	true	$\theta_{b\|a} = .1$
true	false	$\theta_{\bar{b}\|a} = .9$
false	true	$\theta_{b\|\bar{a}} = .8$
false	false	$\theta_{\bar{b}\|\bar{a}} = .2$

Fig. 2. A Bayesian network.

simplicity, which makes it the method of choice for introductory material on the subject of inference in Bayesian networks. Clustering algorithms, however, enjoy a key feature which makes them quite common in large scale implementations: by only expending $O(n \exp(w))$ time and space, these algorithms not only compute the probability of instantiation $\mathbf{e}$, but also compute other useful probabilistic quantities including the posterior marginals $Pr(x|\mathbf{e})$ for every variable X in the Bayesian network. Hence, tree–clustering algorithms provide the largest amount of probabilistic information about the given Bayesian network, assuming that we are willing to commit $O(n \exp(w))$ time and space only.

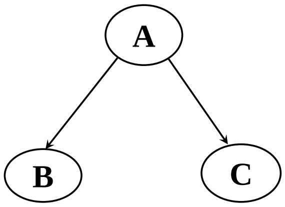
We propose in this paper a new approach to inference in Bayesian networks, which subsumes tree–clustering approaches based on jointrees. According to the proposed approach, the probability distribution of a Bayesian network is represented as a polynomial and probabilistic queries are answered by evaluating and differentiating the polynomial. The polynomial itself is exponential in size, so it cannot be represented explicitly. Instead, it is represented in terms of an arithmetic circuit which can be evaluated and all its partial derivatives computed in time and space linear in its size. Hence, the proposed approach works by first building an arithmetic circuit that computes the network polynomial, and then performs inference by evaluating and differentiating the constructed circuit. As we show later, one can build an arithmetic circuit for a Bayesian network in $O(n \exp(w))$ time and space. Moreover, the probabilistic information that one can retrieve from the partial derivatives of such a circuit include all that can be obtained using jointree methods.

In fact, it was shown recently that every jointree can be interpreted as embedding an arithmetic circuit which computes the network polynomial, and that jointree algorithms are precisely evaluating and differentiating the embedded circuit [Park and Darwiche 2001b]. Therefore, jointree algorithms are a special case of the framework we are proposing here, where specialization is in the specific method used to build the arithmetic circuit. We show, however, that there are other fundamentally different methods for constructing arithmetic circuits. We discuss one particular method in some detail, showing how it can exploit not only the graphical structure of a Bayesian network, but also its local structure as exhibited in the specific values of conditional probabilities. We also point to recent experimental results where the new method could construct efficient arithmetic circuits for networks that are outside the scope of classical jointree methods [Darwiche 2002b]. Hence, the approach we present here not only provides a deeper mathematical understanding of jointree algorithms, but also lifts their basic characteristics to a much more general setting, allowing us to significantly increase the scale of Bayesian networks we can handle efficiently.

This paper is structured as follows. We show in Section 2 how each Bayesian network can be represented as a multi–variate polynomial. We then show in Section 3 how one can obtain answers to a comprehensive list of probabilistic queries by simply evaluating and differentiating the network polynomial. Section 4 is then dedicated to the representation of network polynomials using arithmetic circuits, where we also discuss the evaluation and differentiation of these circuits. We then discuss in Section 5 two methods for generating arithmetic circuits, and finally close in Section 6 with some concluding remarks.

2. BAYESIAN NETWORKS AS MULTI–LINEAR FUNCTIONS

Our goal in this section is to show that the probability distribution induced by a Bayesian network can be represented using a multi–linear function that has very specific properties. We then show in the following sections how this function can be the basis of a comprehensive framework for inference in Bayesian networks.



A	θ_A
true	.5
false	.5

A	B	$\theta_{B\|A}$
true	true	1
true	false	0
false	true	0
false	false	1

A	C	$\theta_{C\|A}$
true	true	.8
true	false	.2
false	true	.2
false	false	.8

Fig. 3. A Bayesian network.

2.1 Technical preliminaries

We will start by settling some notational conventions and providing the formal definition of a Bayesian network. Variables are denoted by upper–case letters (A) and their values by lower–case letters (a). Sets of variables are denoted by bold–face upper–case letters ($\mathbf{A}$) and their instantiations are denoted by bold–face lower–case letters ($\mathbf{a}$). For variable A and value a, we often write a instead of $A{=}a$. For a variable A with values true and false, we use a to denote $A{=}$true and $\bar{a}$ to denote $A{=}$false. Finally, let X be a variable and let $\mathbf{U}$ be its parents in a Bayesian network. The set $X\mathbf{U}$ is called the *family* of variable X, and the variable $\theta_{x|\mathbf{u}}$ is called a *network parameter* and is used to represent the conditional probability $Pr(x \mid \mathbf{u})$; see Figure 2.

A *Bayesian network* over variables $\mathbf{X}$ is a directed acyclic graph over $\mathbf{X}$, in addition to conditional probability values $\theta_{x|\mathbf{u}}$ for each variable X in the network and its parents $\mathbf{U}$. The semantics of a Bayesian network are given by the *chain rule,* which says that the probability of instantiation $\mathbf{x}$ of all network variables $\mathbf{X}$ is simply the product of all network parameters $\theta_{x|\mathbf{u}}$, where $x\mathbf{u}$ is consistent with $\mathbf{x}$. More formally,

$$Pr(\mathbf{x}) = \prod_{x\mathbf{u}\sim\mathbf{x}} \theta_{x|\mathbf{u}},$$

where $\sim$ denotes the compatibility relation among instantiations (that is, $x\mathbf{u} \sim \mathbf{x}$ says that instantiations $x\mathbf{u}$ and $\mathbf{x}$ agree on values of their common variables). For example, the probability of instantiation

$$\overline{\mathit{report}},\ \mathit{leaving},\ \mathit{alarm},\ \overline{\mathit{tampering}},\ \mathit{smoke},\ \mathit{fire}$$

in Figure 1 is given by the product

$$\theta_{\overline{\mathit{report}}|\mathit{leaving}}\ \theta_{\mathit{leaving}|\mathit{alarm}}\ \theta_{\mathit{alarm}|\overline{\mathit{tampering}},\mathit{fire}}\ \theta_{\overline{\mathit{tampering}}}\ \theta_{\mathit{smoke}|\mathit{fire}}\ \theta_{\mathit{fire}}.$$

The justification for this particular semantics of Bayesian networks is outside the scope of this paper, but the reader is referred to other sources for an extensive treatment of the subject [Pearl 1988]. Suffice it to say here that the chain rule is all one needs to reconstruct the probability distribution specified by a Bayesian network.

2.2 The network polynomial

We will now define for each Bayesian network a unique multi–linear function over two types of variables:

Evidence indicators: For each network variable X, we have a set of evidence indicators λ_x.

Network parameters: For each network family $X\mathbf{U}$, we have a set of parameters $\theta_{x|\mathbf{u}}$.

The multi–linear function for a Bayesian network over variables $\mathbf{X}$ has an exponential number of terms, one term for each instantiation of the network variables. The term corresponding to instantiation $\mathbf{x}$ is the product of all evidence indicators and network parameters that are compatible with the instantiation. Consider the simple Bayesian network in Figure 2, which has two variables A and B. The multi–linear function for this network is:

$$f = \lambda_a\lambda_b\theta_a\theta_{b|a} + \lambda_a\lambda_{\bar{b}}\theta_a\theta_{\bar{b}|a} + \lambda_{\bar{a}}\lambda_b\theta_{\bar{a}}\theta_{b|\bar{a}} + \lambda_{\bar{a}}\lambda_{\bar{b}}\theta_{\bar{a}}\theta_{\bar{b}|\bar{a}}.$$

v	λ_a	$\lambda_{\bar{a}}$	λ_b	$\lambda_{\bar{b}}$	λ_c	$\lambda_{\bar{c}}$	θ_a	$\theta_{\bar{a}}$	$\theta_{b\|a}$	$\theta_{b\|\bar{a}}$	$\theta_{\bar{b}\|a}$	$\theta_{\bar{b}\|\bar{a}}$	$\theta_{c\|a}$	$\theta_{c\|\bar{a}}$	$\theta_{\bar{c}\|a}$	$\theta_{\bar{c}\|\bar{a}}$
$\partial f/\partial v$	.1	.4	.1	0	.4	.1	.2	0	.1	0	.1	0	0	0	.5	0

Table 1. Partial derivatives of the network polynomial f of Figure 3 at evidence $a\bar{c}$. The value of the polynomial at this evidence is $f(a\bar{c}) = .1$.

For another example, consider the network in Figure 3. The polynomial of this network has eight terms, some of which are shown below:

$$\begin{aligned} f \;=\; & \lambda_a\lambda_b\lambda_c\theta_a\theta_{b|a}\theta_{c|a} + \\ & \lambda_a\lambda_b\lambda_{\bar{c}}\theta_a\theta_{b|a}\theta_{\bar{c}|a} + \\ & \vdots \\ & \lambda_{\bar{a}}\lambda_{\bar{b}}\lambda_{\bar{c}}\theta_{\bar{a}}\theta_{\bar{b}|\bar{a}}\theta_{\bar{c}|\bar{a}}. \end{aligned}$$

In general, for a Bayesian network with n variables, each term in the multi–linear function will contain $2n$ variables: n parameters and n indicators. The multi–linear function of a Bayesian network is a multi–variate polynomial where each variable has degree 1. We will therefore refer to it as the *network polynomial.*

Definition 1. *Let $\mathcal{N}$ be a Bayesian network over variables $\mathbf{X}$, and let $\mathbf{U}$ denote the parents of variable X in the network. The* <u>*polynomial*</u> *of network $\mathcal{N}$ is defined as follows:*

$$f = \sum_{\mathbf{x}} \prod_{x\mathbf{u} \sim \mathbf{x}} \lambda_x \theta_{x|\mathbf{u}}.$$

The outer sum in the above definition ranges over all instantiations $\mathbf{x}$ of the network variables. For each instantiation $\mathbf{x}$, the inner product ranges over all instantiations of families $x\mathbf{u}$ that are compatible with $\mathbf{x}$.

The polynomial f of Bayesian network $\mathcal{N}$ represents the probability distribution Pr of $\mathcal{N}$ in the following sense. For any piece of evidence $\mathbf{e}$—which is an instantiation of some variables $\mathbf{E}$ in the network—we can evaluate the polynomial f so it returns the probability of $\mathbf{e}$, $Pr(\mathbf{e})$.

Definition 2. *The* <u>*value*</u> *of network polynomial f at evidence $\mathbf{e}$, denoted by $f(\mathbf{e})$, is the result of replacing each evidence indicator λ_x in f with 1 if x is consistent with $\mathbf{e}$, and with 0 otherwise.*

Consider the polynomial,

$$f = \lambda_a\lambda_b\theta_a\theta_{b|a} + \lambda_a\lambda_{\bar{b}}\theta_a\theta_{\bar{b}|a} + \lambda_{\bar{a}}\lambda_b\theta_{\bar{a}}\theta_{b|\bar{a}} + \lambda_{\bar{a}}\lambda_{\bar{b}}\theta_{\bar{a}}\theta_{\bar{b}|\bar{a}},$$

for the network in Figure 2. If the evidence $\mathbf{e}$ is $a\bar{b}$, then $f(\mathbf{e})$ is obtained by applying the following substitutions to f: $\lambda_a = 1$, $\lambda_{\bar{a}} = 0$, $\lambda_b = 0$, and $\lambda_{\bar{b}} = 1$, leading to the probability of $\mathbf{e}$, $\theta_a\theta_{\bar{b}|a}$.

Theorem 1. *Let $\mathcal{N}$ be a Bayesian network representing probability distribution Pr and having polynomial f. For any evidence (instantiation of variables) $\mathbf{e}$, we have $f(\mathbf{e}) = Pr(\mathbf{e})$.*

Hence, our ability to represent and evaluate the network polynomial implies our ability to compute probabilities of instantiations. The polynomial has an exponential size, however, and cannot be represented as a set of terms. But we show in Section 4 that one can represent such polynomials efficiently using arithmetic circuits, in a number of interesting cases. We also show in Section 3 that the partial derivatives of the network polynomial contain valuable information, which can be used to answer a comprehensive set of probabilistic queries.

We close this section by noting that [Russell et al. 1995] has observed that $Pr(\mathbf{e})$ is a linear function in each network parameter. More generally, it is shown in [Castillo et al. 1996; Castillo et al. 1997] that $Pr(\mathbf{e})$ can be expressed as a polynomial of network parameters in which each parameter has degree one. In fact, the polynomials discussed in [Castillo et al. 1996; Castillo et al. 1997] correspond to our network polynomials when evidence indicators are fixed to a particular value.

We will next attribute probabilistic semantics to the partial derivatives of network polynomials, and then provide results on the computational complexity of representing them using arithmetic circuits.

3. PROBABILISTIC SEMANTICS OF PARTIAL DERIVATIVES

Our goal in this section is to attribute probabilistic semantics to the partial derivatives of a network polynomial. As explained in Section 4, if the network polynomial is represented by an arithmetic circuit, then all its first partial derivatives can be computed in time linear in the circuit size. This makes the results of this section especially practical.

We will use the following notation in the rest of the paper. Let $\mathbf{e}$ be an instantiation and $\mathbf{X}$ be a set of variables. Then $\mathbf{e} - \mathbf{X}$ denotes the subset of instantiation $\mathbf{e}$ pertaining to variables not appearing in $\mathbf{X}$. For example, if $\mathbf{e} = ab\bar{c}$, then $\mathbf{e} - A = b\bar{c}$ and $\mathbf{e} - AC = b$. We start with the semantics of first partial derivatives.

3.1 Derivatives with respect to evidence indicators

Consider the polynomial of the Bayesian network in Figure 2:

$$f = \lambda_a\lambda_b\theta_a\theta_{b|a} + \lambda_a\lambda_{\bar{b}}\theta_a\theta_{\bar{b}|a} + \lambda_{\bar{a}}\lambda_b\theta_{\bar{a}}\theta_{b|\bar{a}} + \lambda_{\bar{a}}\lambda_{\bar{b}}\theta_{\bar{a}}\theta_{\bar{b}|\bar{a}}.$$

Consider now the derivative of this polynomial with respect to evidence indicator λ_a:

$$\partial f/\partial\lambda_a = \lambda_b\theta_a\theta_{b|a} + \lambda_{\bar{b}}\theta_a\theta_{\bar{b}|a}.$$

The partial derivative $\partial f/\partial\lambda_a$ results from polynomial f by setting indicator λ_a to 1 and indicator $\lambda_{\bar{a}}$ to 0, which means that the derivative $\partial f/\partial\lambda_a$ corresponds to *conditioning* the polynomial f on event a. Note also that the value of $\partial f/\lambda_a$ at evidence $\mathbf{e}$ is independent of the value that variable A may take in $\mathbf{e}$ since $\partial f/\partial\lambda_a$ no longer contains any indicators for variable A. These observations lead to the following theorem.

Theorem 2. *Let $\mathcal{N}$ be a Bayesian network representing probability distribution Pr and having polynomial f. For every variable X and evidence $\mathbf{e}$, we have*

$$\frac{\partial f}{\partial\lambda_x}(\mathbf{e}) = Pr(x, \mathbf{e} - X). \tag{1}$$

That is, if we differentiate the polynomial f with respect to indicator λ_x and evaluate the result at evidence $\mathbf{e}$, we obtain the probability of instantiation $x, \mathbf{e} - X$. For an example, consider Table 1, which depicts the partial derivatives of the network polynomial of Figure 3 evaluated at evidence $a\bar{c}$. According to Equation 1, the value of derivative $\partial f/\partial\lambda_{\bar{a}}$ at evidence $a\bar{c}$, .4, gives us the probability of $\bar{a}\bar{c}$.

Therefore, if we evaluate the network polynomial at some evidence $\mathbf{e}$ and compute all its first partial derivatives at this same evidence, then not only do we have the probability of evidence $\mathbf{e}$, but also the probability of every instantiation $\mathbf{e}'$ which differs with $\mathbf{e}$ on the value of one variable. The ability to compute the probabilities of such modifications on instantiation $\mathbf{e}$ efficiently is crucial for approximately solving the problem of *maximum a posteriori hypothesis* (MAP) [Park 2002; Park and Darwiche 2001a]. The MAP problem is that of finding a most probable instantiation $\mathbf{e}$ of some variables $\mathbf{E}$. One class of approximate methods for MAP starts with some instantiation $\mathbf{e}$ and then tries to improve on it using local search, by examining all instantiations that result from changing the value of a single variable in $\mathbf{e}$ (called the neighbors of $\mathbf{e}$). Equation 1 is then very relevant to these approximate algorithms as it provides an efficient method to score the neighbors of $\mathbf{e}$ during local search [Park 2002; Park and Darwiche 2001a].

Another class of queries that is immediately obtainable from partial derivatives is posterior marginals.

Corollary 1. *For every variable X and evidence $\mathbf{e}$, $X \notin \mathbf{E}$:*

$$Pr(x \mid \mathbf{e}) = \frac{1}{f(\mathbf{e})}\frac{\partial f}{\partial\lambda_x}(\mathbf{e}). \tag{2}$$

Therefore, the partial derivatives give us the posterior marginal of every variable. Given Table 1, where evidence $\mathbf{e} = a\bar{c}$, we have

$$Pr(b \mid \mathbf{e}) = \frac{1}{f(\mathbf{e})}\frac{\partial f}{\partial\lambda_b}(\mathbf{e}) = 1,$$

and

$$Pr(\bar{b} \mid \mathbf{e}) = \frac{1}{f(\mathbf{e})}\frac{\partial f}{\partial\lambda_{\bar{b}}}(\mathbf{e}) = 0.$$

The ability to compute such posteriors efficiently is probably the key celebrated property of jointree algorithms [Huang and Darwiche 1996; Shenoy and Shafer 1986; Jensen et al. 1990], as compared to variable elimination algorithms [Shachter et al. 1990; Dechter 1996; Zhang and Poole 1996]. The latter class of algorithms is much simpler except that they can only compute such posteriors by invoking themselves once for each network variable, leading to a complexity of $O(n^2 \exp(w))$, where n is the number of network variables and w is the network treewidth. Jointree algorithms can do this in $O(n \exp(w))$ time, however, but at the expense of a more complicated algorithm. When we discuss complexity in Sections 4 and 5, we will find that the proposed approach in this paper can also perform this computation in $O(n \exp(w))$ time. In fact, we will give a deeper explanation of why jointree algorithms can attain this complexity in Section 5, where we point to recent results showing that they are a special case of the approach presented here (they also differentiate the network polynomial).

One of the main complications in Bayesian network inference relates to the update of probabilities after having retracted evidence. This seems to pose no difficulties in the presented framework. For example, we can immediately compute the posterior marginal of every instantiated variable, after the evidence on that variable has been retracted.

Corollary 2. *For every variable X and evidence $\mathbf{e}$, we have:*

$$Pr(\mathbf{e} - X) \; = \; \sum_x \frac{\partial f}{\partial \lambda_x}(\mathbf{e}); \tag{3}$$

$$Pr(x' \mid \mathbf{e} - X) \; = \; \frac{\frac{\partial f}{\partial \lambda_{x'}}(\mathbf{e})}{\sum_x \frac{\partial f}{\partial \lambda_x}(\mathbf{e})}. \tag{4}$$

Note that $Pr(\mathbf{e} - X)$ can also be obtained be evaluating the polynomial f at evidence $\mathbf{e} - X$, but that would require many evaluations of f if we are to consider every possible variable X. The main point of the above corollary is to obtain all these quantities from the derivatives of f at evidence $\mathbf{e}$. Consider Table 1 for an example of this corollary, where evidence $\mathbf{e} = a\bar{c}$. We have

$$Pr(\mathbf{e} - A) = Pr(\bar{c}) = \frac{\partial f}{\partial \lambda_a}(\mathbf{e}) + \frac{\partial f}{\partial \lambda_{\bar{a}}}(\mathbf{e}) = .5.$$

The above computation is the basis of an investigation of model adequacy [Cowell et al. 1999, Chapter 10] and is typically implemented in the jointree algorithm using the technique of *fast retraction,* which requires a modification to the standard propagation method in jointrees [Cowell et al. 1999, Page 104]. As given by the above theorem, we get this computation for free once we have partial derivatives with respect to network indicators.

3.2 Derivatives with respect to network parameters

We now turn to partial derivatives with respect to network parameters.

Theorem 3. *Let $\mathcal{N}$ be a Bayesian network representing probability distribution Pr and having polynomial f. For every family $X\mathbf{U}$ in the network, and for every evidence $\mathbf{e}$, we have*

$$\theta_{x|\mathbf{u}} \frac{\partial f}{\partial \theta_{x|\mathbf{u}}}(\mathbf{e}) = Pr(x, \mathbf{u}, \mathbf{e}). \tag{5}$$

This theorem has indirectly been shown in [Russell et al. 1995] (since $f(\mathbf{e}) = Pr(\mathbf{e})$) and has major applications to sensitivity analysis and learning. Specifically, the derivative $\partial Pr(\mathbf{e})/\partial \theta_{x|\mathbf{u}}$ is the basis for an efficient approach to sensitivity analysis that identifies minimal parameter changes necessary to satisfy constraints on probabilistic queries [Chan and Darwiche 2002]. Another application of this partial derivative is to the learning of network parameters from data, for which there are two main approaches. The first approach called APN, for Adaptive Probabilistic Networks, is based on reducing the learning problem to that of optimizing a function of many variables [Russell et al. 1995]. Specifically, it attempts to find the values of network parameters which will maximize the probability of data, therefore, requiring that we compute $\partial Pr(\mathbf{d})/\partial \theta_{x|\mathbf{u}}$ for each parameter $\theta_{x|\mathbf{u}}$ and each piece of data $\mathbf{d}$. The second approach for learning parameters is based on the Expectation Maximization (EM) algorithm [Lauritzen 1995], and requires the computation of posterior marginals over network families, which can be easily obtained given Equation 5.

3.3 Second partial derivatives

We now turn to the semantics of second partial derivatives. Since we have two types of variables in a network polynomial (evidence indicators and network parameters), we have three different types of second partial derivatives. The semantics of each derivative is given next.

THEOREM 4. *Let $\mathcal{N}$ be a Bayesian network representing probability distribution Pr and having polynomial f. For every pair of variables X, Y and evidence $\mathbf{e}$, when $x \neq y$:*

$$\frac{\partial^2 f}{\partial \lambda_x \partial \lambda_y}(\mathbf{e}) = Pr(x, y, \mathbf{e} - XY). \tag{6}$$

For every family $X\mathbf{U}$, variable Y, and evidence $\mathbf{e}$:

$$\theta_{x|\mathbf{u}} \frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \lambda_y}(\mathbf{e}) = Pr(x, \mathbf{u}, y, \mathbf{e} - Y). \tag{7}$$

For every pair of families $X\mathbf{U}, Y\mathbf{V}$ and evidence $\mathbf{e}$, when $x\mathbf{u} \neq y\mathbf{v}$:

$$\theta_{x|\mathbf{u}} \theta_{y|\mathbf{v}} \frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \theta_{y|\mathbf{v}}}(\mathbf{e}) = Pr(x, \mathbf{u}, y, \mathbf{v}, \mathbf{e}). \tag{8}$$

Theorems 2–4 show us how to compute answers to classical probabilistic queries by differentiating the polynomial representation of a Bayesian network. Therefore, if we have an efficient way to represent and differentiate the polynomial, then we also have an efficient way to perform probabilistic reasoning. For example, Equation 6 allows us to compute marginals over pairs of variables using second partial derivatives—these marginals are needed for identifying conditional independence and for measuring mutual information between pairs of variables.

Another use of Theorems 2–4 is in computing valuable partial derivatives using classical probabilistic quantities. Therefore, if we need the values of these derivatives but only have access to classical inference algorithms, then we can use the given identities to recover the necessary derivatives. For example, Equation 8 shows us how to compute the second partial derivative of $Pr(\mathbf{e})$ with respect to two network parameters, $\theta_{x|\mathbf{u}}$ and $\theta_{y|\mathbf{v}}$, using the joint probability over their corresponding families, $Pr(x, \mathbf{u}, y, \mathbf{v}, \mathbf{e})$. We have to note, however, that expressing partial derivatives in terms of classical probabilistic quantities requires some conditions: $\theta_{x|\mathbf{u}}$ and $\theta_{y|\mathbf{v}}$ cannot be 0. Therefore, partial derivatives contain more information than their corresponding probabilistic quantities.

Theorems 2–4 can also facilitate the derivation of results relating to sensitivity analysis. Here's one example.

THEOREM 5. *Let $\mathcal{N}$ be a Bayesian network representing distribution Pr and having polynomial f. For variable Y, family $X\mathbf{U}$ and evidence $\mathbf{e}$:*

$$\begin{aligned} \frac{\partial Pr(y \mid \mathbf{e})}{\partial \theta_{x|\mathbf{u}}} &= \frac{1}{f(\mathbf{e})^2}\left(f(\mathbf{e}) \frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \lambda_y}(\mathbf{e}) - \frac{\partial f}{\partial \theta_{x|\mathbf{u}}}(\mathbf{e}) \frac{\partial f}{\partial \lambda_y}(\mathbf{e}) \right) \\ &= \frac{Pr(y, x, \mathbf{u} \mid \mathbf{e}) - Pr(y \mid \mathbf{e}) Pr(x, \mathbf{u} \mid \mathbf{e})}{Pr(x \mid \mathbf{u})}, \text{ when } Pr(x|\mathbf{u}) \neq 0. \end{aligned}$$

This theorem provides an elegant answer to the most central question of sensitivity analysis in Bayesian networks, as it shows how we can compute the sensitivity of a conditional probability to a change in some network parameter. The theorem phrases this computation in terms of both partial derivatives and classical probabilistic quantities—the second part, however, can only be used when $Pr(x \mid \mathbf{u}) \neq 0$.[1]

[1] There seems to be two approaches for computing the derivative $\partial Pr(y \mid \mathbf{e})/\partial \theta_{x|\mathbf{u}}$, which has been receiving increased attention recently due to its role in sensitivity analysis and the learning of network parameters [Chan and Darwiche 2002]. We have just presented one approach where we found a closed form for $\partial Pr(y \mid \mathbf{e})/\partial \theta_{x|\mathbf{u}}$, using both partial derivatives and classical probabilistic quantities. The other approach capitalizes on the observation that $Pr(y \mid \mathbf{e})$ has the form $(\alpha\theta_{x|\mathbf{u}} + \beta)/(\gamma\theta_{x|\mathbf{u}} + \delta)$ for some constants α, β, γ and δ [Castillo et al. 1996]. According to this second approach, one tries to compute the values of these constants based on the given Bayesian network and then computes the derivative of $(\alpha\theta_{x|\mathbf{u}} + \beta)/(\gamma\theta_{x|\mathbf{u}} + \delta)$ with respect to $\theta_{x|\mathbf{u}}$. See [Jensen 1999; Kjaerulff and van der Gaag 2000] for an example of this approach, where it is shown how to compute such constants using a limited number of propagations in the context of a jointree algorithm.

4. ARITHMETIC CIRCUITS THAT COMPUTE MULTI–LINEAR FUNCTIONS

We have shown in earlier sections that the probability distribution of a Bayesian network can be represented using a polynomial. We have also shown that a good number of probabilistic queries can be answered immediately once the value and partial derivatives of the polynomial are computed. Therefore, if we have an efficient way to evaluate and differentiate the polynomial, then we have an efficient and comprehensive approach to probabilistic inference in Bayesian networks. The goal of this section is to present a particular representation of the network polynomial which facilitates its evaluation and differentiation.

The network polynomial has an exponential number of terms. Hence, any direct representation of the polynomial will be infeasible in general. Instead, we will *represent* or *compute* the polynomial using an *arithmetic circuit.*

Definition 3. *An arithmetic circuit over variables Σ is a rooted, directed acyclic graph whose leaf nodes are labelled with numeric constants or variables in Σ and whose other nodes are labelled with multiplication and addition operations. The size of an arithmetic circuit is measured by the number of edges that it contains.*

An arithmetic circuit is a graphical representation of a function f over variables Σ; see Figure 5. As we show later, it is sometimes possible to represent a polynomial f of exponential size using an arithmetic circuit of linear size (exponential and linear in the number of polynomial variables). Hence, arithmetic circuits can be very compact representations of polynomials, and we shall adopt them as our representation of network polynomials in this paper. This leaves us with two questions. First, assuming that we have a compact arithmetic circuit which computes the network polynomial, how can we efficiently evaluate and differentiate the circuit? Second, how do we obtain a compact arithmetic circuit that computes a given network polynomial? The first question will be addressed next, while the second question will be delegated to Section 5.

4.1 Differentiating arithmetic circuits

Evaluating an arithmetic circuit is straightforward: we simply traverse the circuit upward, computing the value of a node after having computed the values of its children. Computing the circuit derivatives, however, is a bit more involved. First, we will not distinguish between an arithmetic circuit f and its unique output node. Let v be an arbitrary node in circuit f. We are interested in the partial derivative of f with respect to node v, $\partial f/\partial v$. The key observation is to view the circuit f as a function of each and every circuit node v. If v is the root node (circuit output), then $\frac{\partial f}{\partial v} = 1$. If v is not the root node, and has parents p, then by the chain rule of differential caclulus:

$$\frac{\partial f}{\partial v} = \sum_{p} \frac{\partial f}{\partial p}\frac{\partial p}{\partial v}.$$

Suppose now that v' are the other children of parent p. If parent p is a multiplication node, then

$$\frac{\partial p}{\partial v} = \frac{\partial (v \prod_{v'} v')}{\partial v} = \prod_{v'} v'.$$

Similarly, if parent p is an addition node,

$$\frac{\partial p}{\partial v} = \frac{\partial (v + \sum_{v'} v')}{\partial v} = 1.$$

With these equations, we can recursively compute the partial derivatives of f with respect to any node v. The procedure is described below in terms of two passes, requiring two registers, $vr(v)$ and $dr(v)$, for each circuit node v. In the upward–pass, we evaluate the circuit by setting the values of $vr(v)$ registers, and in the downward–pass, we differentiate the circuit by setting the values of $dr(v)$ registers. From here on, when we say an upward–pass of the circuit, we will mean a traversal of the circuit where the children of a node are visited before the node itself is visited. Similarly, in a downward–pass, the parents of a node will be visited first.

—*Initialization:* $dr(v)$ is initialized to zero except for root v where $dr(v) = 1$.

—*Upward–pass:* At node v, compute the value of v and store it in $vr(v)$.

—*Downward–pass:* At node v and for each parent p, increment $dr(v)$ by

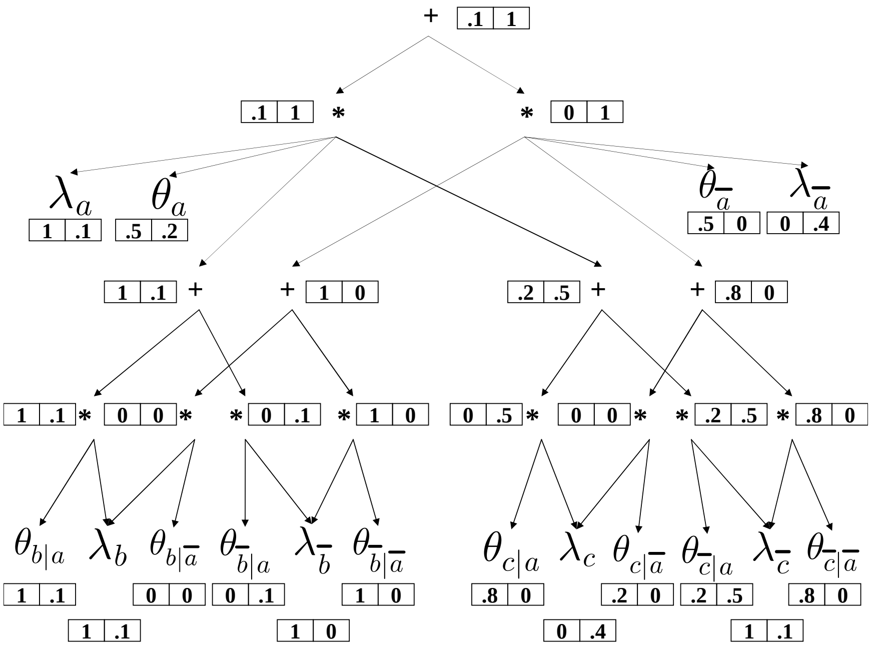

+ .1 1
.1 1 *
* 0 1
λ_a 1 .1
θ_a .5 .2
$\theta_{\overline{a}}$ .5 0
$\lambda_{\overline{a}}$ 0 .4
1 .1 +
+ 1 0
.2 .5 +
+ .8 0
1 .1 *
0 0 *
* 0 .1
* 1 0
0 .5 *
0 0 *
* .2 .5
* .8 0
$\theta_{b|a}$ 1 .1
λ_b 1 .1
$\theta_{b|\overline{a}}$ 0 0
$\theta_{\overline{b}|a}$ 0 .1
$\lambda_{\overline{b}}$ 1 0
$\theta_{\overline{b}|\overline{a}}$ 1 0
$\theta_{c|a}$ .8 0
λ_c 0 .4
$\theta_{c|\overline{a}}$ .2 0
$\theta_{\overline{c}|a}$ .2 .5
$\lambda_{\overline{c}}$ 1 .1
$\theta_{\overline{c}|\overline{a}}$ .8 0

Fig. 4. An arithmetic circuit for the network polynomial of Figure 3, after it has been evaluated and differentiated under evidence $a\bar{c}$. Registers vr are shown on the left, and registers dr are shown on the right.

—$dr(p)$ if p is an addition node;
—$dr(p) \prod_{v'} vr(v')$ if p is a multiplication node, where v' are the other children of p.

Figure 4 contains an arithmetic circuit that has been evaluated and differentiated under evidence $\mathbf{e} = a\bar{c}$ using the above method. This circuit computes the polynomial of the Bayesian network in Figure 3, and will be visited again in Section 5 where we discuss the generation of arithmetic circuits.

4.2 The complexity of differentiating circuits

The upward–pass in the above scheme clearly takes time linear in the circuit size, where size is defined as the number of edges in the circuit. The downward–pass takes linear time only when each multiplication node has a bounded number of children, otherwise the time to evaluate the term $\prod_{v'} vr(v')$ cannot be bounded by a constant. This can be addressed by observing that the term $\prod_{v'} vr(v')$ equals $vr(p)/vr(v)$ when $vr(v) \neq 0$ and, hence, the time to evaluate it can be bounded by a constant if we use division. Even the case where $vr(v) = 0$ can be handled efficiently, but that requires two additional bits per multiplication node p: $bit1(p)$ indicates whether some child of p has a zero value, and $bit2(p)$ indicates whether exactly one child of node p has a zero value. Moreover, the meaning of register $vr(p)$ is overloaded when the value of p is zero, where it contains the product of all non–zero values attained by children of node p. This leads to the following more refined scheme, which is based on [Sawyer 1984] and assumes that the circuit alternates between addition and multiplication nodes.

```
—Initialization: $dr(v)$ is initialized to zero except for root $v$ where $dr(v) = 1$.
—Upward–pass: At node $v$ with children $c$,
  —if $v$ is an addition node, set $vr(v)$ to $\sum_{bit1(c)=0} vr(c)$
  —if $v$ is a multiplication node,
      set $vr(v)$ to $\prod_{vr(c)\neq 0} vr(c)$;
      set $bit1(v)$ to 1 if $vr(c) = 0$ for some child $c$, and to 0 otherwise;
      set $bit2(v)$ to 1 if $vr(c) = 0$ for exactly one child $c$, and to 0 otherwise.
```

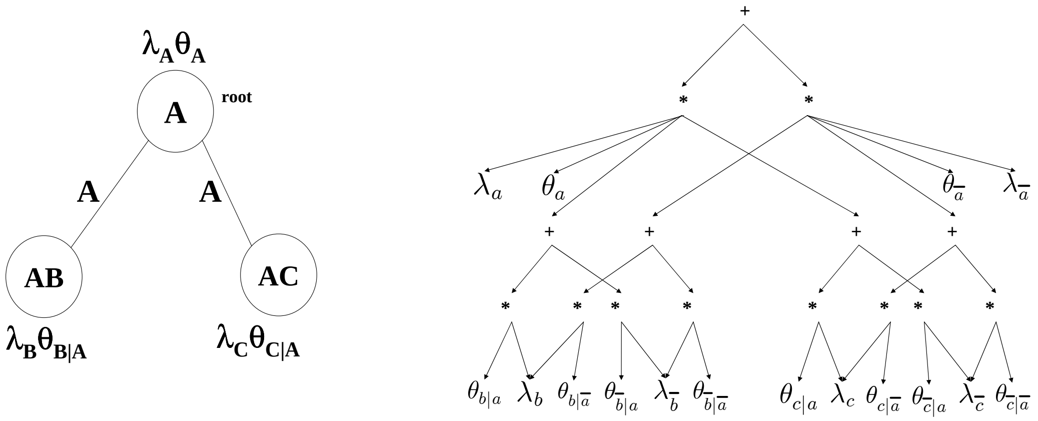


Fig. 5. A jointree for the Bayesian network in Figure 3 and its corresponding arithmetic circuit.

—*Downward–pass:* At node v and for each parent p,
—if p is an addition node, increment $dr(v)$ by $dr(p)$;
—if p is a multiplication node, increment $dr(v)$ by
$dr(p)vr(p)/vr(v)$ if $bit1(p) = 0$;
$dr(p)vr(p)$ if $bit2(p) = 1$ and $vr(v) = 0$.

When the downward–pass of the above method terminates, we are guaranteed that the value of every addition node v is stored in $vr(v)$, and the value of every multiplication node v is stored in $vr(v)$ if $bit1(v) = 0$, and is 0 otherwise. We are also guaranteed that the derivative of f with respect to every node v is stored in $dr(v)$. Finally, the method takes time which is linear in the circuit size.

4.3 Rounding errors

We close this section by pointing out that once a circuit is evaluated and differentiated, it is possible to bound the rounding error in the computed value of the circuit output under a particular model of error propagation. Specifically, let δ be the *local* rounding error generated when computing the value of an addition or multiplication node in the upward–pass. It is reasonable to assume that $|\delta| \leq \epsilon|v|$, where:

- v is the value we would obtain for the node when using infinite–precision to add/multiply its children values;
- ϵ is a constant representing the machine–specific relative error occurring in the floating–point representation of a real number.

We can then bound the rounding error in the computed value of the circuit f by $\epsilon \sum_v v\partial f/\partial v$, where v ranges over all internal nodes in the circuit [Iri 1984]. This bound can be computed easily as the downward–pass is being executed, allowing us to bound the rounding error in the computed probability of evidence as this corresponds to the value of the circuit output.

5. COMPILING ARITHMETIC CIRCUITS

Our goal in this section is to present algorithms for generating arithmetic circuits that compute network polynomials. The goal is to try to generate the smallest circuit possible, and to offer guarantees on the complexity of generated circuits whenever possible. We will discuss two classes of methods for this purpose. The first class exploits the global structure of a Bayesian network (its topology) and comes with a complexity guarantee in terms of the network treewidth. The second class of algorithms can also exploit local structure (the specific values of conditional probabilities), and could be quite effective in situations where the first approach is intractable. But first, we present a new notion of complexity for Bayesian networks which is motivated by *algebraic complexity theory* [von zur Gathen 1988]:

Definition 4. *The circuit complexity of a Bayesian network $\mathcal{N}$ is the size of the smallest arithmetic circuit that computes the network polynomial of $\mathcal{N}$.*

5.1 Circuits that exploit global structure

We will now present a method for generating arithmetic circuits assuming that we have a jointree for the given network [Huang and Darwiche 1996; Pearl 1988; Jensen 1999]. A *jointree* for a Bayesian network $\mathcal{N}$ is a labelled tree $(\mathcal{T}, \mathcal{L})$, where $\mathcal{T}$ is a tree and $\mathcal{L}$ is a function that assigns labels to nodes in $\mathcal{T}$. A jointree must satisfy three properties:

(1) each label $\mathcal{L}(i)$ is a set of variables in the Bayesian network;

(2) each family $X\mathbf{U}$ in the network must appear in some label $\mathcal{L}(i)$;

(3) if a variable appears in the labels of jointree nodes i and j, it must also appear in the label of each node k on the path connecting them.

Nodes in a jointree, and their labels, are called *clusters.* Similarly, edges in a jointree, and their labels, are called *separators,* where the label of edge ij is defined as $\mathcal{L}(i) \cap \mathcal{L}(j)$. Figure 5 depicts a jointree for the Bayesian network of Figure 3, which contains three clusters.

A jointree is the key data structure in a class of influential algorithms for inference in Bayesian networks [Shenoy and Shafer 1986; Jensen et al. 1990]. Before a jointree is used by these algorithms, each CPT $\theta_{X|\mathbf{U}}$ must be assigned to a cluster that contains family $X\mathbf{U}$. Moreover, evidence on a variable X is captured through a table λ_X over variable X which is also assigned to a cluster that contains X. Finally, a cluster in the jointree is chosen and designated as the root, allowing us to direct the tree and define parent/child relationships between neighboring clusters and separators. The jointree in Figure 5 depicts the root cluster, in addition to the assignment of CPTs and evidence tables to various clusters. We will show next that each jointree embeds an arithmetic circuit that computes the network polynomial. Later, we point to recent results showing that classical jointree algorithms actually evaluate and differentiate the embedded circuit and are, therefore, subsumed by the framework discussed here.

Definition 5. *Given a root cluster, a particular assignment of CPT and evidence tables to clusters, the* <u>*arithmetic circuit embedded*</u> *in a jointree is defined as follows. The circuit includes:*

- *one output addition node* f*;*
- *an addition node* $\mathbf{s}$ *for each instantiation of a separator* $\mathbf{S}$*;*
- *a multiplication node* $\mathbf{c}$ *for each instantiation of a cluster* $\mathbf{C}$*;*
- *an input node* λ_x *for each instantiation* x *of variable* X*;*
- *an input node* $\theta_{x|\mathbf{u}}$ *for each instantiation* $x\mathbf{u}$ *of family* $X\mathbf{U}$*.*

The children of the output node f *are the multiplication nodes* $\mathbf{c}$ *generated by the root cluster; the children of an addition node* $\mathbf{s}$ *are all compatible multiplication nodes* $\mathbf{c}$ *generated by the child cluster; the children of a multiplication node* $\mathbf{c}$ *are all compatible addition nodes* $\mathbf{s}$ *generated by child separators, in addition to all compatible inputs nodes* $\theta_{x|\mathbf{u}}$ *and* λ_x *for which CPT* $\theta_{X|\mathbf{U}}$ *and evidence table* λ_X *are assigned to cluster* $\mathbf{C}$*.*

Figure 5 depicts a jointree and its embedded arithmetic circuit. Note the correspondence between addition nodes in the circuit (except the output node) and instantiations of separators in the jointree. Note also the correspondence between multiplication nodes in the circuit and instantiations of clusters in the jointree. Some jointree algorithms maintain a table with each cluster and separator, which are indexed by the instantiations of corresponding cluster or separator [Huang and Darwiche 1996; Jensen et al. 1990]. These algorithms are then representing the addition/multiplication nodes of the embedded circuit explicitly. One useful feature of the circuit embedded in a jointree, however, is that it does not require that we represent its edges explicitly as these can be inferred from the jointree structure. This leads to less space requirements, but increases the time for evaluating and differentiating the circuit given the overhead needed to infer these edges.[2] Another useful feature of the circuit embedded in a jointree is the guarantees one can offer on its size.

Theorem 6. *Let* J *be a jointree for Bayesian network* $\mathcal{N}$ *with* n *clusters, a maximum cluster size* c*, and a maximum separator size* s*. The arithmetic circuit embedded in jointree* J *computes the network polynomial for* $\mathcal{N}$ *and has* $O(n\exp(c))$ *multiplication nodes,* $O(n\exp(s))$ *addition nodes, and* $O(n\exp(c))$ *edges.*

[2] Some optimized implementations of jointree algorithms maintain indices that associate cluster entries with compatible entries in their neighboring separators, in order to reduce jointree propagation time [Huang and Darwiche 1996]. These algorithms are then representing both the nodes and edges of the embedded circuit explicitly.

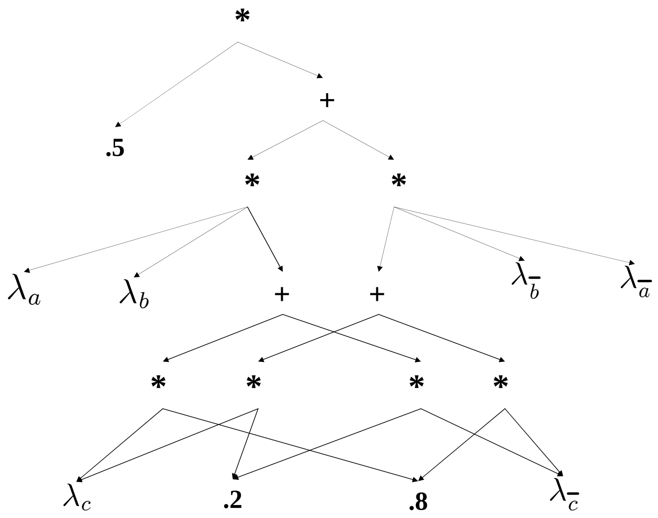


Fig. 6. An arithmetic circuit that exploits local structure. The circuit computes the polynomial of the Bayesian network in Figure 3.

It is well known that if the directed graph underlying a Bayesian network has n nodes and treewidth w, then a jointree for $\mathcal{N}$ exists which has no more than n clusters and a maximum cluster size of $w+1$. Theorem 6 is then telling us that the circuit complexity of such networks is $O(n \exp(w))$.

We note here that the arithmetic circuit embedded in a jointree has a very specific structure: it alternates between addition and multiplication nodes, and each multiplication node has a single parent. This specific structure permits more efficient schemes for circuit evaluation and differentiation than we have proposed earlier (since the partial derivative with respect to a multiplication node and its single parent must be equal). Two such methods are discussed in [Park and Darwiche 2001b], where it is shown that these methods require less space than is required by the methods of Section 4.

Definition 5 provides a method for generating arithmetic circuits based on jointrees, but it also serves as a connection between the approach proposed here and the influential inference approaches based on jointree propagation. According to these approaches, one performs inference by passing messages in two phases: an inward phase where messages are passed towards the root cluster and then an outward phase where messages are passed away from the root cluster. It was shown recently that the inward phase of jointree propagation corresponds to an evaluation of the embedded circuit, and the outward phase corresponds to a differentiation of the circuit [Park and Darwiche 2001b]. Specifically, it was shown that the two main methods for jointree propagation, known as Shenoy–Shafer [Shenoy and Shafer 1986] and Hugin [Jensen et al. 1990] propagation, do correspond precisely to two specific numeric methods for circuit differentiation that have different time/space properties.

These findings have a number of implications. First, they provide a deeper understanding of jointree algorithms, allowing us to extract more information from them than was previously done—see [Park and Darwiche 2001b] for some examples. Second, they suggest that building a jointree is one specific way of accomplishing a more general task, that of building an arithmetic circuit for computing the network polynomial. This leaves us with the question: What other methods can one employ for accomplishing this purpose? We address this question in the following section, where we sketch a new approach for building arithmetic circuits that reduces the problem to one of logical reasoning [Darwiche 2002b].

5.2 Circuits that exploit local structure

The arithmetic circuits embedded in jointrees come with a guarantee on their size. This guarantee, however, is only a function of the network topology and is both an upper and a lower bound. Therefore, if the jointree has a cluster of large size, say 40, then the embedded arithmetic circuit will be intractable.

The key point to observe here is that one can generate arithmetic circuits of manageable size even when the jointree has large clusters, assuming the conditional probabilities of the Bayesian network exhibit some local

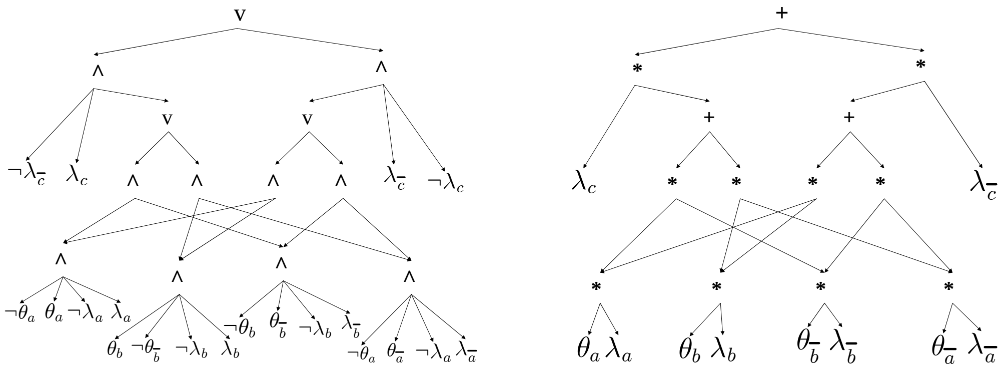

∨
∧
∧
∨
∨
$\neg\lambda_{\bar{c}}$
λ_c
$\lambda_{\bar{c}}$
$\neg\lambda_c$
$\neg\theta_a$ θ_a $\neg\lambda_a$ λ_a
θ_b $\neg\theta_{\bar{b}}$ $\neg\lambda_b$ λ_b
$\neg\theta_b$ $\theta_{\bar{b}}$ $\neg\lambda_b$ $\lambda_{\bar{b}}$
$\neg\theta_a$ $\theta_{\bar{a}}$ $\neg\lambda_a$ $\lambda_{\bar{a}}$
+
*
+
λ_c
$\lambda_{\bar{c}}$
θ_a λ_a
θ_b λ_b
$\theta_{\bar{b}}$ $\lambda_{\bar{b}}$
$\theta_{\bar{a}}$ $\lambda_{\bar{a}}$

Fig. 7. On the left, a negation normal form that satisfies smoothness, determinism, and decomposability. On the right, the corresponding arithmetic circuit after removing leaf nodes labelled with 1. The negation normal form encodes the polynomial of a Bayesian network $A \rightarrow C \leftarrow B$, where each node has two values, and C is an exclusive–or of A and B. That is, $\theta_{c|ab} = 0$ and $\theta_{c|\bar{a}\bar{b}} = 0$ which imply that $\theta_{\bar{c}|ab} = 1$ and $\theta_{\bar{c}|\bar{a}\bar{b}} = 1$.

structure. By local structure, we mean information about the specific values that conditional probabilities attain; for example, whether some probabilities equal 0 or 1, and whether some probabilities in the same table are equal. The Bayesian network of Figure 3 exhibits some local structure in the previous sense. If one exploits this local structure, then one can build the smaller arithmetic circuit in Figure 6, instead of the larger circuit in Figure 5. The difference between the two circuits is that one is valid for any particular values of the network parameter, while the other is valid for the specific values given in Figure 3.

We now turn to a recent approach for generating arithmetic circuits which can exploit local structure, and works by reducing the problem to one of logical reasoning as logic turns out to be useful for specifying information about local structure [Darwiche 2002b]. The approach is based on three conceptual steps. First, the network polynomial is encoded using a propositional theory. Next, the propositional theory is factored by converting it to a special logical form. Finally, an arithmetic circuit is extracted from the factored propositional theory. The first and third steps are representational, but the second step is the one involving computation. We will next explain each step in some more details.

Step 1: Encoding a multi–linear function using a propositional theory. The purpose of this step is to specify the network polynomial using a propositional theory. To illustrate how a multi–linear function can be specified using a propositional theory, consider the following function $f = \alpha\gamma + \alpha\beta\gamma + \gamma$ over real–valued variables α, β, γ. The basic idea is to specify this multi–linear function using a propositional theory that has exactly three models, where each model encodes one of the terms in the function. Specifically, suppose we have the Boolean variables $V_\alpha, V_\beta, V_\gamma$. Then the propositional theory $\Delta_f = (V_\alpha \vee \neg V_\beta) \wedge V_\gamma$ encodes the multi–linear function f since it has three models:

—$\sigma_1 : V_\alpha = \mathsf{true}, V_\beta = \mathsf{false}, V_\gamma = \mathsf{true}$;
—$\sigma_2 : V_\alpha = \mathsf{true}, V_\beta = \mathsf{true}, V_\gamma = \mathsf{true}$;
—$\sigma_3 : V_\alpha = \mathsf{false}, V_\beta = \mathsf{false}, V_\gamma = \mathsf{true}$.

Each one of these models σ_i is interpreted as encoding a term t_i in the multi–linear function f as follows. A real–valued variable appears in term t_i iff model σ_i sets its corresponding Boolean variable to true. Hence, the first model encodes the term $\alpha\gamma$; the second model encodes the term $\alpha\beta\gamma$; and the third model encodes the term γ. The theory Δ_f then encodes the multi–linear function that results from adding up all these terms: $f = \alpha\gamma + \alpha\beta\gamma + \gamma$. This method of specifying network polynomials allows one to easily capture local structure; that is, to declare certain information about values of polynomial variables. For example, if we know that variable α has a zero value, then we can exclude all terms that contain α by conjoining $\neg V_\alpha$ with our encoding. The reader is referred to [Darwiche 2002b] for an efficient method which generates propositional theories that encode network polynomials.

Step 2: Factoring the propositional encoding. If we view the conversion of a network polynomial into an arithmetic circuit as a factoring process, then the purpose of this second step is to accomplish a similar task but at the logical level. Instead of starting with a polynomial (set of terms), we start with a propositional theory (set of models). And instead of building an arithmetic circuit that computes the polynomial, we build a Boolean circuit that computes the propositional theory. To compute a propositional theory in this context is to be able to count its models under any values of propositional variables. One logical form that permits this computation is Negation Normal Form (NNF): a rooted, directed acyclic graph where leaves are labelled with constants/literals, and where internal nodes are labelled with conjunctions /disjunctions; see Figure 7. The NNF must satisfy three properties, which we define next. Let α_n denote the logical sentence represented by the NNF rooted at node n.

—*Decomposability:* For each and–node with children $c_1, \ldots, c_n$, sentences α_{c_i} and α_{c_j} cannot share a variable for $i \neq j$.

—*Determinism:* For each or–node with children $c_1, \ldots, c_n$, sentences α_{c_i} and α_{c_j} must contradict each other for $i \neq j$.

—*Smoothness:* For each or–node with children $c_1, \ldots, c_n$, sentences α_{c_i} and α_{c_j} must mention the same set of variables for $i \neq j$.

The NNF in Figure 7 satisfies the above properties, and encodes the network polynomial of a small Bayesian network. The reader is referred to [Darwiche 2002a] for an algorithm that converts propositional theories from CNF to NNF, while ensuring the above three properties.[3]

Step 3: Extracting an arithmetic circuit. The purpose of this last step is to extract an arithmetic circuit that computes the polynomial encoded by an NNF. If Δ_f is a propositional theory that encodes a network polynomial f, and if Δ_f is an NNF that satisfies the properties of smoothness, determinism, and decomposability, then an arithmetic circuit that computes the polynomial f can be obtained easily as follows: replace and–nodes in Δ_f by multiplications; replace or–nodes by additions; and replace each leaf node labelled with a negated variable by a constant 1. The resulting arithmetic circuit is then guaranteed to compute the polynomial f [Darwiche 2002b]. Figure 7 depicts an NNF and its corresponding arithmetic circuit. Note that the generated arithmetic circuit is no larger than the NNF. Hence, if we attempt to minimize the size of NNF, we are also minimizing the size of generated arithmetic circuit.

We refer the reader to [Darwiche 2002b] for further details on this approach, and for experimental results showing how Bayesian networks whose jointrees have clusters with more than 60 variables could be handled very efficiently, leading to arithmetic circuits of relatively very small size. These networks correspond to applications involving well–known sequential and combinational digital circuits. They are deterministic in the sense that all their conditional probabilities are 0/1 except for the probabilities on root nodes. Such networks are completely outside the scope of the method discussed in Section 5.1 since the sizes of corresponding jointrees are prohibitive. Note that a simpler approach to handle determinism would have been to build a circuit as discussed in Section 5.1, identify circuit nodes whose values are stuck at zero, and then prune such nodes to build a smaller circuit. This approach will work and is capable of having the same effect as the approach we just discussed, as long as the full circuit (the one before pruning) is manageable.[4] This approach is not feasible, however, for many of the networks that are discussed in [Darwiche 2002b].

6. CONCLUSION

We have presented a comprehensive approach for inference in Bayesian networks which is based on evaluating and differentiating arithmetic circuits. Specifically, we have shown how the probability distribution of a Bayesian network can be represented using a polynomial, and how a large number of probabilistic queries can be retrieved immediately from the value and partial derivatives of such a polynomial. We have also shown

[3] Note that an Ordered Binary Decision Diagram (OBDD) can be understood as a Boolean circuit, in which case it can be shown to be an NNF that satisfies the properties of decomposability and determinism [Bryant 1986; Darwiche and Marquis 2002]. Moreover, the property of smoothness can always be ensured in polynomial time. Hence, if one has an algorithm for converting CNF into OBDD, then one immediately has an algorithm for converting CNF into smooth, deterministic, decomposable NNF [Darwiche 2002a]. An OBDD, however, satisfies additional properties, leading to larger NNFs than necessary.

[4] This approach corresponds to the technique of zero–compression in jointree algorithms [Jensen and Andersen 1990], which performs inference on a jointree to identify and remove cluster and separator entries that are stuck at zero. After such pruning, however, one must explicitly link cluster entries (multiplication nodes) and separator entries (addition nodes), leading to an explicit representation of the embedded circuit. In such a case, the jointree as a data structure loses much of its appeal since it does not provide much value beyond an explicit representation of the circuit.

how to represent polynomials efficiently using arithmetic circuits, and how to evaluate and differentiate them in time and space which is linear in their size. Finally, we have presented two classes of methods for building arithmetic circuits and discussed their properties.

The approach we have presented here subsumes the jointree approach for inference in Bayesian networks, which has been shown recently to correspond to circuit evaluation and differentiation as discussed in this paper. Our proposed framework provides a deeper understanding of the jointree approach and lifts its basic characteristics to a more general framework, in which the complexity of inference is sensitive to both the local and global structure of Bayesian networks. This also leads to a more refined notion of computational complexity for Bayesian network inference, circuit complexity, which is based on both local and global network structure.

Acknowledgment

This work has been partially supported by NSF grant IIS-9988543 and MURI grant N00014-00-1-0617.

REFERENCES

Bodlaender, H. L. 1993. A tourist guide through treewidth. *ACTA CYBERNETICA 11*, 1-2, 1–22.

Bodlaender, H. L. 1996. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing 25*, 6, 1305–1317.

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

Castillo, E., Gutiérrez, J. M., and Hadi, A. S. 1996. Goal oriented symbolic propagation in Bayesian networks. In *Proc. AAAI National Conference* (1996), pp. 1263–1268.

Castillo, E., Gutiérrez, J. M., and Hadi, A. S. 1997. Sensitivity analysis in discrete Bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics 27*, 412–423.

Chan, H. and Darwiche, A. 2002. When do numbers really matter? *Journal of Artificial Intelligence Research 17*, 265–287.

Cowell, R., Dawid, A., Lauritzen, S., and Spiegelhalter, D. 1999. *Probabilistic Networks and Expert Systems.* Springer.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence 126*, 1-2, 5–41.

Darwiche, A. 2002a. A compiler for deterministic, decomposable negation normal form. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)* (Menlo Park, California, 2002), pp. 627–634. AAAI Press.

Darwiche, A. 2002b. A logical approach to factoring belief networks. In *Proceedings of KR* (2002), pp. 409–420.

Darwiche, A. and Marquis, P. 2002. A knowlege compilation map. *Journal of Artificial Intelligence Research 17*, 229–264.

Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)* (1996), pp. 211–219.

Huang, C. and Darwiche, A. 1996. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning 15*, 3, 225–263.

Iri, M. 1984. Simultaneous computation of functions, partial derivatives and estimates of rounding error. *Japan J. Appl. Math. 1*, 223–252.

Jensen, F. and Andersen, S. K. 1990. Approximations in Bayesian belief universes for knowledge based systems. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI)* (Cambridge, MA, July 1990), pp. 162–169.

Jensen, F. V. 1999. Gradient descent training of Bayesian networks. In *Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)* (1999), pp. 5–9.

Jensen, F. V., Lauritzen, S., and Olesen, K. 1990. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly 4*, 269–282.

Kjaerulff, U. and van der Gaag, L. C. 2000. Making sensitivity analysis computationally efficient. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)* (2000).

Lauritzen, S. L. 1995. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis 19*, 191–201.

Park, J. 2002. MAP complexity results and approximation methods. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)* (San Francisco, California, 2002), pp. 388–396. Morgan Kaufmann Publishers, Inc.

Park, J. and Darwiche, A. 2001a. Approximating MAP using stochastic local search. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)* (San Francisco, California, 2001), pp. 403–410. Morgan Kaufmann Publishers, Inc.

Park, J. and Darwiche, A. 2001b. A differential semantics for jointree algorithms. Technical Report D–118, Computer Science Department, UCLA, Los Angeles, Ca 90095. To appear in NIPS'02.

PEARL, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., San Mateo, California.

RUSSELL, S., BINDER, J., KOLLER, D., AND KANAZAWA, K. 1995. Local learning in probabilistic networks with hidden variables. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)* (1995), pp. 1146–1152.

SAWYER, J. W. 1984. First partial differentiation by computer with an application to categorical data analysis. *The American Statistician 38*, 4, 300–308.

SHACHTER, R., D'AMBROSIO, B., AND DEL FAVERO, B. 1990. Symbolic Probabilistic Inference in Belief Networks. In *Proc. Conf. on Uncertainty in AI* (1990), pp. 126–131.

SHENOY, P. P. AND SHAFER, G. 1986. Propagating belief functions with local computations. *IEEE Expert 1*, 3, 43–52.

VON ZUR GATHEN, J. 1988. Algebraic complexity theory. *Ann. Rev. Comp. Sci. 3*, 317–347.

ZHANG, N. L. AND POOLE, D. 1996. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research 5*, 301–328.

APPENDIX

A. PROOFS OF THEOREMS

In the following proofs, $\sim$ will denote the compatibility relationship among variable instantiations. Hence, $\mathbf{x} \sim \mathbf{y}$ means that instantiations $\mathbf{x}$ and $\mathbf{y}$ are compatible: they agree on every common variable. Also, we will assume that the Bayesian network variables are $\mathbf{Z}$ and that Z is an arbitrary variable in the network with parents $\mathbf{W}$. Hence, the network polynomial will be written as:

$$f = \sum_{\mathbf{z}} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \lambda_z.$$

Proof of Theorem 1

Given

$$f = \sum_{\mathbf{z}} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \lambda_z,$$

Definition 2 gives us

$$\begin{aligned} f(\mathbf{e}) &= \sum_{\mathbf{z}} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \begin{cases} 1, & \text{if } z \sim \mathbf{e}; \\ 0, & \text{otherwise.} \end{cases} \\ &= \sum_{\mathbf{z} \sim \mathbf{e}} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \\ &= Pr(\mathbf{e}). \; \Box \end{aligned}$$

Proof of Theorem 2

By definition of partial derivative of a multi–linear function, we have:

$$\frac{\partial f}{\partial \lambda_x} = \sum_{\mathbf{z} \sim x} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \prod_{z \sim \mathbf{z},\, z \neq x} \lambda_z.$$

Definition 2 then gives us:

$$\begin{aligned} \frac{\partial f}{\partial \lambda_x}(\mathbf{e}) &= \sum_{\mathbf{z} \sim x} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \prod_{z \sim \mathbf{z},\, z \neq x} \begin{cases} 1, & \text{if } z \sim \mathbf{e}; \\ 0, & \text{otherwise.} \end{cases} \\ &= \sum_{\mathbf{z} \sim x,\, z \sim \mathbf{e} - X} \prod_{z\mathbf{w} \sim \mathbf{z}} \theta_{z|\mathbf{w}} \\ &= Pr(x, \mathbf{e} - X). \; \Box \end{aligned}$$

Proof of Theorem 3

By definition of partial derivative of a multi–linear function, we have:

$$\frac{\partial f}{\partial \theta_{x|\mathbf{u}}} = \sum_{\mathbf{z} \sim x\mathbf{u}} \prod_{z\mathbf{w} \sim \mathbf{z},\, z \neq x} \theta_{z|\mathbf{w}} \prod_{z \sim \mathbf{z}} \lambda_z.$$

Definition 2 then gives us:

$$\begin{aligned}\frac{\partial f}{\partial \theta_{x|\mathbf{u}}}(\mathbf{e}) &= \sum_{\mathbf{z}\sim x\mathbf{u}} \prod_{z\mathbf{w}\sim\mathbf{z},\, z\neq x} \theta_{z|\mathbf{w}} \prod_{z\sim\mathbf{z}} \begin{cases}1, & \text{if } z\sim\mathbf{e};\\ 0, & \text{otherwise.}\end{cases}\\ &= \sum_{\mathbf{z}\sim x\mathbf{u},\, \mathbf{z}\sim\mathbf{e}} \prod_{z\mathbf{w}\sim\mathbf{z},\, z\neq x} \theta_{z|\mathbf{w}}.\end{aligned}$$

Multiplying both sides by $\theta_{x|\mathbf{u}}$, we get:

$$\begin{aligned}\frac{\partial f}{\partial \theta_{x|\mathbf{u}}}(\mathbf{e})\theta_{x|\mathbf{u}} &= \sum_{\mathbf{z}\sim x\mathbf{u},\, \mathbf{z}\sim\mathbf{e}} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}}.\\ &= Pr(x,\mathbf{u},\mathbf{e}). \ \square\end{aligned}$$

Proof of Theorem 4

Proving Equation 6. Since $x \neq y$, we have:

$$\frac{\partial^2 f}{\partial \lambda_x \partial \lambda_y} = \sum_{\mathbf{z}\sim xy} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}} \prod_{z\sim\mathbf{z},\, z\neq x, z\neq y} \lambda_z.$$

Definition 2 then gives:

$$\begin{aligned}\frac{\partial^2 f}{\partial \lambda_x \partial \lambda_y}(\mathbf{e}) &= \sum_{\mathbf{z}\sim xy} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}} \prod_{z\sim\mathbf{z},\, z\neq x, z\neq y} \begin{cases}1, & \text{if } z\sim\mathbf{e};\\ 0, & \text{otherwise.}\end{cases}\\ &= \sum_{\mathbf{z}\sim xy} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}} \begin{cases}1, & \text{if } z\sim\mathbf{e}-XY;\\ 0, & \text{otherwise.}\end{cases}\\ &= \sum_{\mathbf{z}\sim xy,\, \mathbf{z}\sim\mathbf{e}-XY} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}}\\ &= Pr(x,y,\mathbf{e}-XY).\end{aligned}$$

Proving Equation 7. We have:

$$\frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \lambda_y} = \sum_{\mathbf{z}\sim x\mathbf{u}y} \prod_{z\mathbf{w}\sim\mathbf{z},\, z\mathbf{w}\neq x\mathbf{u}} \theta_{z|\mathbf{w}} \prod_{z\sim\mathbf{z},\, z\neq y} \lambda_z.$$

Definition 2 then gives:

$$\frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \lambda_y}(\mathbf{e}) = \sum_{\mathbf{z}\sim x\mathbf{u}y, \mathbf{z}\sim\mathbf{e}-Y} \prod_{z\mathbf{w}\sim\mathbf{z},\, z\mathbf{w}\neq x\mathbf{u}} \theta_{z|\mathbf{w}}.$$

Multiplying both sides by $\theta_{x|\mathbf{u}}$,

$$\theta_{x|\mathbf{u}} \frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \lambda_y}(\mathbf{e}) = \sum_{\mathbf{z}\sim x\mathbf{u}y, \mathbf{z}\sim\mathbf{e}-Y} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}}.$$

Hence,

$$\theta_{x|\mathbf{u}} \frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \lambda_y}(\mathbf{e}) = Pr(x,\mathbf{u},y,\mathbf{e}-Y).$$

Proving Equation 8. Since $x\mathbf{u} \neq y\mathbf{v}$, we have:

$$\frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \theta_{y|\mathbf{v}}} = \sum_{\mathbf{z}\sim x\mathbf{u}y\mathbf{v}} \prod_{z\mathbf{w}\sim\mathbf{z},\, z\mathbf{w}\neq x\mathbf{u},\, z\mathbf{w}\neq y\mathbf{v}} \theta_{z|\mathbf{w}} \prod_{z\sim\mathbf{z}} \lambda_z.$$

Definition 2 then gives:

$$\frac{\partial^2 f}{\partial \theta_{x|\mathbf{u}} \partial \theta_{y|\mathbf{v}}}(\mathbf{e}) = \sum_{\mathbf{z}\sim x\mathbf{u}y\mathbf{v},\, \mathbf{z}\sim\mathbf{e}} \prod_{z\mathbf{w}\sim\mathbf{z},\, z\mathbf{w}\neq x\mathbf{u},\, z\mathbf{w}\neq y\mathbf{v}} \theta_{z|\mathbf{w}}.$$

If $x\mathbf{u}$ and $y\mathbf{v}$ are not compatible, then $\frac{\partial^2 f}{\partial\theta_{x|\mathbf{u}}\partial\theta_{y|\mathbf{v}}}(\mathbf{e}) = 0$, and the equation holds. Suppose now that they are compatible, and multiply both sides by $\theta_{x|\mathbf{u}}\theta_{y|\mathbf{v}}$:

$$\theta_{x|\mathbf{u}}\theta_{y|\mathbf{v}}\frac{\partial^2 f}{\partial\theta_{x|\mathbf{u}}\partial\theta_{y|\mathbf{v}}}(\mathbf{e}) = \sum_{\mathbf{z}\sim x\mathbf{u}y\mathbf{v},\, \mathbf{z}\sim\mathbf{e}} \prod_{z\mathbf{w}\sim\mathbf{z}} \theta_{z|\mathbf{w}}.$$

Finally,

$$\theta_{x|\mathbf{u}}\theta_{y|\mathbf{v}}\frac{\partial^2 f}{\partial\theta_{x|\mathbf{u}}\partial\theta_{y|\mathbf{v}}}(\mathbf{e}) = Pr(x, \mathbf{u}, y, \mathbf{v}, \mathbf{e}).\ \square$$

Proof of Theorem 5

If $Y \in \mathbf{E}$, we have two cases: either $\mathbf{e}$ implies y or $\mathbf{e}$ contradicts y. It is easy to verify the theorem in each of the previous cases. Suppose now that $Y \notin \mathbf{E}$. We have:

$$\begin{aligned}\frac{\partial Pr(y \mid \mathbf{e})}{\partial\theta_{x|\mathbf{u}}} &= \frac{\partial}{\partial\theta_{x|\mathbf{u}}}\frac{Pr(y,\mathbf{e})}{Pr(\mathbf{e})}\\ &= \frac{\partial}{\partial\theta_{x|\mathbf{u}}}\frac{f(y,\mathbf{e})}{f(\mathbf{e})}\\ &= \frac{1}{f(\mathbf{e})^2}\left[f(\mathbf{e})\frac{\partial f(y,\mathbf{e})}{\partial\theta_{x|\mathbf{u}}} - f(y,\mathbf{e})\frac{\partial f(\mathbf{e})}{\partial\theta_{x|\mathbf{u}}}\right].\end{aligned}$$

Since

$$\frac{\partial f(y,\mathbf{e})}{\partial\theta_{x|\mathbf{u}}} = \frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(y,\mathbf{e}),$$

and

$$\frac{\partial f(\mathbf{e})}{\partial\theta_{x|\mathbf{u}}} = \frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(\mathbf{e}),$$

we get:

$$\frac{\partial Pr(y \mid \mathbf{e})}{\partial\theta_{x|\mathbf{u}}} = \frac{1}{f(\mathbf{e})^2}\left[f(\mathbf{e})\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(y,\mathbf{e}) - f(y,\mathbf{e})\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(\mathbf{e})\right].$$

We can now replace all terms by classical probabilistic quantities using Theorems 1–3:

$$\begin{aligned}\frac{\partial Pr(y \mid \mathbf{e})}{\partial\theta_{x|\mathbf{u}}} &= \frac{1}{Pr(\mathbf{e})^2}\left[\frac{Pr(\mathbf{e})Pr(y,\mathbf{e},x,\mathbf{u})}{Pr(x \mid \mathbf{u})} - \frac{Pr(y,\mathbf{e})Pr(\mathbf{e},x,\mathbf{u})}{Pr(x \mid \mathbf{u})}\right]\\ &= \frac{Pr(y,\mathbf{e},x,\mathbf{u})}{Pr(x \mid \mathbf{u})Pr(\mathbf{e})} - \frac{Pr(y,\mathbf{e})Pr(\mathbf{e},x,\mathbf{u})}{Pr(x \mid \mathbf{u})Pr(\mathbf{e})^2}\\ &= \frac{Pr(y,x,\mathbf{u} \mid \mathbf{e}) - Pr(y \mid \mathbf{e})Pr(x,\mathbf{u} \mid \mathbf{e})}{Pr(x \mid \mathbf{u})}.\end{aligned}$$

Or we can replace some of the terms by their corresponding derivatives using Theorems 2–4:

$$\begin{aligned}\frac{\partial Pr(y \mid \mathbf{e})}{\partial\theta_{x|\mathbf{u}}} &= \frac{1}{f(\mathbf{e})^2}\left[f(\mathbf{e})\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(y,\mathbf{e}) - f(y,\mathbf{e})\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(\mathbf{e})\right]\\ &= \frac{1}{f(\mathbf{e})^2}\left[f(\mathbf{e})\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(y,\mathbf{e}-Y) - f(y,\mathbf{e}-Y)\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(\mathbf{e})\right]\\ &= \frac{1}{f(\mathbf{e})^2}\left[f(\mathbf{e})\frac{\partial^2 f}{\partial\theta_{x|\mathbf{u}}\partial\lambda_y}(\mathbf{e}) - \frac{\partial f}{\partial\lambda_y}(\mathbf{e})\frac{\partial f}{\partial\theta_{x|\mathbf{u}}}(\mathbf{e})\right].\ \square\end{aligned}$$

Proof of Theorem 6

That the embedded arithmetic circuit computes the network polynomial is shown in [Park and Darwiche 2001b].

By Definition 5, there is a one–to–one correspondence between multiplication nodes and cluster instantiations; hence, the number of multiplication nodes is $O(n\exp(c))$. Similarly, and except for the root node, there is a one–to–one correspondence between addition nodes and separator instantiations; hence, the number of addition nodes is $O(n\exp(s))$ since the number of jointree edges is $n-1$.

As for the number of edges, note that the circuit alternates between addition and multiplication nodes, where inputs nodes are always children of multiplication nodes. Hence, we will count edges by simply counting the total number of neighbors (parents and children) that each multiplication node has. By Definition 5, each multiplication node will have a single parent. Moreover, the number of children that a multiplication node $\mathbf{c}$ will have depends on the cluster $\mathbf{C}$ that generates it. Specifically, the node will have one child $\mathbf{s}$ for each child separator $\mathbf{S}$, will have one child λ_x for each evidence table λ_X assigned to cluster $\mathbf{C}$, and one child $\theta_{x|\mathbf{u}}$ for each CPT $\theta_{X|\mathbf{U}}$ assigned to the same cluster. Now let r be the root cluster; i be any cluster; c_i be the cluster size; n_i be the number of its neighbors; e_i and p_i be the numbers of evidence tables and CPTs assigned to the cluster, respectively. The total number of neighbors for multiplication nodes is then bounded by:

$$\exp(c_r)(n_r + 1 + e_r + p_r) + \sum_{i \neq r} \exp(c_i)(n_i + e_i + p_i).$$

Note: a multiplication node generated by the root cluster will have one addition parent and n_r addition children, while a multiplication node generated by a non–root cluster will have one addition parent and $n_i - 1$ addition children. Since, $c_i \leq c$ for all i, we can bound the number of edges by:

$$\exp(c) + \exp(c) \sum_i (n_i + e_i + p_i).$$

Note also that the number of edges in a tree is one less than the number of nodes, leading to $\sum_i n_i = 2(n-1)$. Moreover, we have $\sum_i e_i = n$ and $\sum_i p_i = n$ since we only have n evidence tables and n CPTs. Hence, the total number of edges can be bounded by $(4n-1)\exp(c)$, which is $O(n\exp(c))$. $\square$