Research Task C - The Problem: Parallel Solution for Message Passing on Arithmetic Circuits

André E. dos Santos DOSSANTOS@CS.UREGINA.CA
Department of Computer Science
University of Regina
Regina, Canada

Abstract

Arithmetic Circuits (AC) is a graphical method for reasoning with Bayesian networks (BNs) based on partial differentiation. ACs is a viable framework for applications of BNs to embedded systems, which are characterized for their primitive computational resources. An AC is a directed acyclic graph with four types of variables: evidence indicators, network parameters (probability values), sum nodes and product nodes. There are two phases to compile the AC: upward and inward. Those phases pass messages through the AC, which are essentially numbers. The number of messages passed in both phases is twice the number of edges in the compilation of the polynomial. Once the BN is processed, one can compute in constant-time answers to a large class of probabilistic queries. In this paper we present the algorithm to compile the AC. We show how a parallel solution and describe how it compares to a serial solution.

1. Introduction

Bayesian networks (BNs) (Pearl, 1988) is a probabilistic graphical model for dealing with uncertainty in artificial intelligence. However, BNs are known for requiring a high amount of processing during its running time of execution (Koller and Friedman, 2009). This strongly limits the applications of BNs to embedded systems, which are characterized for their primitive computational resources. Darwiche (2013) proposed a new approach for inference in Bayesian networks based on partial differentiation called *Arithmetic Circuits* (Darwiche, 2013). The differential approach presents two key contributions. First, it enphasizes the role of partial differentiation on probabilistic reasoning, giving a new utility to central tasks of BNs. Second, it helps the migration of BN applications to embedded systems, which are known for constraint in computational resources.

An AC is a directed acyclic graph with four types of variables: evidence indicators, network parameters (probability values), sum nodes and product nodes. The leaves are evidence indicators ands network parameters and the rest of the graph consists of sum and product nodes. There are two phases to compile the AC given and evidence: upward and inward. By the end of the upward phase we can compute the probability of the evidence. By the end of the downward phase we can compute in constant-time answers to a large class of probabilistic queries.

In this paper we present the algorithm to compile an AC. It computes both upward and inward phase. We show how a parallel solution and describe how it compares to a

serial solution. There are limitations to the solutions since those steps are constrained by a topological ordering.

This paper is organized as follows. In Section 2, arithmetic circuits are reviewed. Section 3 presents the parallel solution for message passing on arithmetic circuits. Conclusions are given in Section 4.

2. Background

A *joint probability distribution* (JPD) is function P on the Cartesian product V of the variable domains such that the following two conditions hold:

$$0 \leq P(v) \leq 1.0 \text{ for each configuration } v \in V; \text{ and} \tag{1}$$

$$\sum_{v \in V} P(v) = 1.0 \tag{2}$$

A *Bayesian network* (BN) Pearl (1988) is a directed acyclic graph (DAG) on finite set of discrete variables U together with *conditional probability tables* (CPTs) $P(v_1|Pa(v_1))$, $P(v_2|Pa(v_2))$, ..., $P(v_n|Pa(v_n))$, where $Pa(v_i)$ denotes the parents of the variable v_i in the DAG. The product of the CPTs for the DAG on U is a *joint probability distribution* $P(U)$ (Pearl, 1988). For example, Figure 1 shows a Bayesian network with CPTs $P(a)$ and $P(b|a)$ in Table 1.

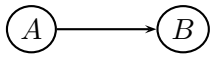

Figure 1: A BN from (Darwiche, 2013).

A *parameterized* CPT receives a evidence indicator λ multiplied to the probability value. For instance, Table 2 shows the parameterized CPTs for the BN of Table 1. Notice that $\bar{x}$ denotes the instantiation $x = 0$.

Table 1: Table (a) provides the prior probability of variable A and Table (b) provides the conditional probability of B given A.

(a)

A	$P(A)$
1	0.3
0	0.7

(b)

A	B	$P(B\|A)$
1	1	0.1
1	0	0.9
0	1	0.8
0	0	0.2

A BN can be represented as a polynomial $\mathcal{F}$ as follows:

$$\mathcal{F}(\lambda, \theta) = \sum_{U} \prod \theta_i \prod \lambda_i \tag{3}$$

As an example, the polynomial of the BN in Figure 1 is

$$\mathcal{F} = \theta_A \theta_{B|A} \lambda_A \lambda_B + \theta_A \theta_{\bar{B}|A} \lambda_A \lambda_{\bar{B}} + \theta_{\bar{A}} \theta_{B|\bar{A}} \lambda_{\bar{A}} \lambda_B + \theta_{\bar{A}} \theta_{\bar{B}|\bar{A}} \lambda_{\bar{A}} \lambda_{\bar{B}} \tag{4}$$

The size of the polynomial is exponential to the size of the network.

Following an elimination ordering of variables, an AC can be drawn for the compiled polynomial. For instance, the compiled polynomial in (4) is

$$\mathcal{F} = \theta_A \lambda_A (\theta_{B|A} \lambda_B + \theta_{\bar{B}|A} \lambda_{\bar{B}}) + \theta_{\bar{A}} \lambda_{\bar{A}} (\theta_{B|\bar{A}} \lambda_B + \theta_{\bar{B}|\bar{A}} \lambda_{\bar{B}}) \tag{5}$$

and depicted in Figure 2.

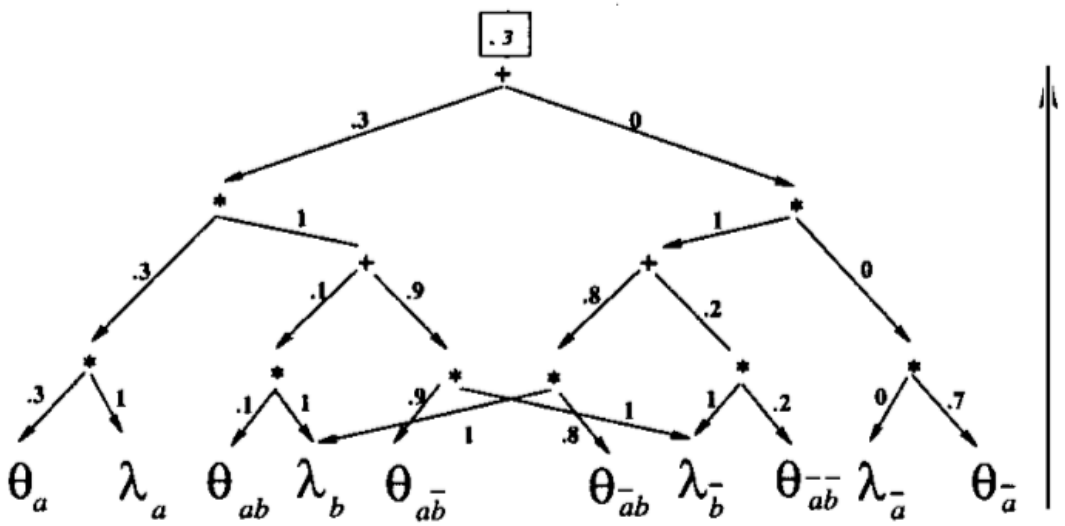

Figure 2: An AC representing Equation (5).

3. Evaluating and Differentiating a Polynomial Representation

The evaluation of the AC and computing the partial derivatives is a two-phase message passing scheme in which each message is simply a number. The first phase, massages are sent from nodes to their parents in the AC following the operations of each node. The phase starts from the leaves up to the root. Note that the value of a of a node can not be computed before all its children values has been computed. First phase is described in lines 1-11 in Algorithm 1. Figure 2 shows this process, where it leads to assigning the value 0.3 to the root, indicating that the probability of evidence E is $P(E) = 0.3$.

Second phase, messages are sent from nodes to children in the same rooted DAG, leading to computation of all partial derivatives. This process is known as *back propagation* Rumelhart et al. (1988), a common method of training artificial neural networks, and its proven to be able to compute the partial derivatives of variables on the leaves. The phase starts from the root down to the leaves. The derivative value of the root, by definition,

is always 1. For the remaining nodes v, if the parent $Pa(v)$ is a summation node, the derivative value of v is equal to its parent. If the parent $Pa(v)$ is a multiplication node, the derivative value of v is equal to the summation of its parent times the others siblings of v. This processes is illustrated in Figure 3. Note that the derivative of a node of a node can not be computed before all its parents derivatives has been computed. Second phase is described in lines 13-27 in Algorithm 35.

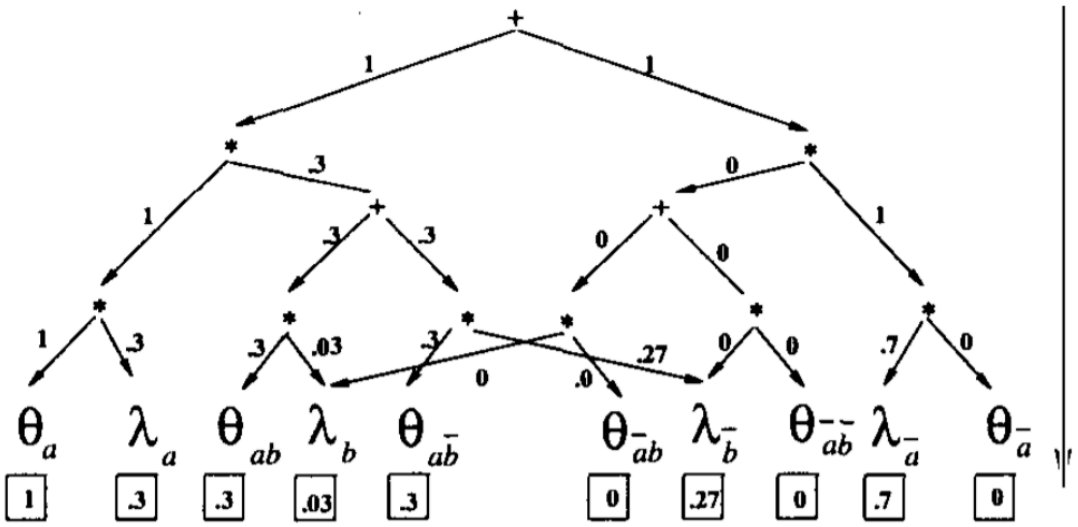

Figure 3: Second phase of the evaluation of AC in Figure 5, under evidence $E = A$.

After the two-phase steps we have the capacity to answer, in constant time, a very large class of probabilistic queries, relating to classical inference, parameter estimation, model validation and sensitivity analysis. Some of those queries are (i) in the root node the probability of the evidence, and (ii) the posterior marginal of any network variable.

3.1 Serial Solution

The serial solution computes the values and derivatives of each node sequentially in both cases. The only restrictions are those imposed by the topological ordering. For instance, the second phase where the derivatives are propagated top-bottom, can be implemented with *breadth-first search* algorithm.

3.2 Parallel Solution

The phases of evaluating and differentiating an AC can improved by the parallelization of the nodes value calculation. Phase one, massages are sent from nodes to their parents in the AC following the operations of each node, as described in lines 1-11 in Algorithm 1. Since the phase starts from the leaves up to the root, the layers of operations (sum and product) can be computed in parallel. For instance, in the second last layer of Figure 2, the value of

nodes $(\theta_a \star \lambda_a)$, $(\theta_{ab} \star \lambda_b)$, $(\theta_{a\bar{b}} \star \lambda_{\bar{b}})$, $(\lambda_b \star \theta_{\bar{a}b})$, $(\lambda_{\bar{b}} \star \theta_{\bar{a}\bar{b}})$, and $(\lambda_{\bar{a}} \star \theta_{\bar{a}})$ can be computed in parallel.

In the second phase, messages are sent from nodes to children in the same rooted DAG, leading to computation of all partial derivatives, as described in lines 12-27 in Algorithm 1. Since the phase starts from the root down to the leaves, the derivatives of the layers of children can be computed in parallel. For instance, in the last layers of Figure 3, the derivatives of nodes θ_a, λ_a, θ_{ab}, λ_b, $\theta_{a\bar{b}}$, $\theta_{\bar{a}b}$, $\lambda_{\bar{b}}$, $\theta_{\bar{a}\bar{b}}$, $\lambda_{\bar{a}}$, and $\theta_{\bar{a}}$ can be computed in parallel.

4. Conclusion

Reasoning with ACs presents several advantages. It emphasizes the role of partial differentiation on probabilistic reasoning, giving a new utility to central tasks of BNs. Also, it helps the migration of BN applications to embedded systems, which are known for constraint in computational resources.

There are two phases to compile the AC: upward and inward. Once the BN is processed, one can compute in constant-time answers to a large class of probabilistic queries. In this paper we presented the algorithm to compile the AC, drawn from Darwiche (2009). It computes both upward and inward phase. We have shown how a parallel solution can be implemented and described how it compares to a serial solution. The limitations to the solutions are essentially imposed by topological ordering only.

References

Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

Adnan Darwiche. A differential approach to inference in bayesian networks. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI2000)*, 2013.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Algorithm 35 `CircP2`($\mathcal{AC}$, vr(), dr()). Assumes the values of leaf circuit nodes v have been initialized in vr(v) and the circuit alternates between addition and multiplication nodes, with leaves having multiplication parents.

input:

$\mathcal{AC}$: arithmetic circuit
vr(): array of value registers (one register for each circuit node)
dr(): array of derivative registers (one register for each circuit node)

output: computes the value of circuit output v in vr(v) and computes derivatives of leaf nodes v in dr(v)

main:

1: **for** each non-leaf node v with children c (visit children before parents) **do**
2: **if** v is an addition node **then**
3: vr(v)$\leftarrow \sum_{c:\ \text{bit}(c)=0}$ vr(c) {if bit(c) $= 1$, value of c is 0}
4: **else**
5: **if** v has a single child c' with vr(c') $= 0$ **then**
6: bit(v)$\leftarrow$1; vr(v)$\leftarrow \prod_{c \neq c'}$ vr(c)
7: **else**
8: bit(v)$\leftarrow$0; vr(v)$\leftarrow \prod_{c}$ vr(c)
9: **end if**
10: **end if**
11: **end for**
12: dr(v)$\leftarrow$0 for all non-root nodes v; dr(v)$\leftarrow$1 for root node v
13: **for** each non-root node v (visit parents before children) **do**
14: **for** each parent p of node v **do**
15: **if** p is an addition node **then**
16: dr(v)$\leftarrow$dr(v) + dr(p)
17: **else**
18: **if** vr(p) $\neq 0$ **then** {p has at most one child with zero value}
19: **if** bit(p) $= 0$ **then** {p has no zero children}
20: dr(v)$\leftarrow$dr(v) + dr(p)vr(p)/vr(v)
21: **else if** vr(v) $= 0$ **then** {v is the single zero child}
22: dr(v)$\leftarrow$dr(v) + dr(p)vr(p)
23: **end if**
24: **end if**
25: **end if**
26: **end for**
27: **end for**