Compiling Bayesian Networks by Symbolic Probability Calculation Based on Zero-suppressed BDDs

Shin-ichi Minato
Div. of Computer Science
Hokkaido University
Sapporo 060–0814, Japan

Ken Satoh
National Institute of Informatics
Sokendai
Tokyo 101–8430, Japan

Taisuke Sato
Dept. of Computer Science
Tokyo Institute of Technology
Tokyo 152–8552, Japan

Abstract

Compiling Bayesian networks (BNs) is a hot topic within probabilistic modeling and processing. In this paper, we propose a new method for compiling BNs into Multi-Linear Functions (MLFs) based on Zero-suppressed Binary Decision Diagrams (ZB-DDs), which are a graph-based representation of combinatorial item sets. Our method differs from the original approach of Darwiche et al., which encodes BNs into Conjunctive Normal Forms (CNFs) and then translates CNFs into factored MLFs. Our approach directly translates a BN into a set of factored MLFs using a ZBDD-based symbolic probability calculation. The MLF may have exponential computational complexity, but our ZBDD-based data structure provides a compact factored form of the MLF, and arithmetic operations can be executed in a time almost linear with the ZBDD size. In our method, it is not necessary to generate the MLF for the whole network, as we can extract MLFs for only part of the network related to the query, avoiding unnecessary calculation of redundant MLF terms. We present experimental results for some typical benchmark examples. Although our algorithm is simply based on the mathematical definition of probability calculation, performance is competitive to existing state-of-the-art methods.

1 Introduction

Compiling Bayesian Networks (BNs) is a hot topic within probabilistic modeling and processing. Recently, data structures of decision diagrams[9; 4; 5; 6; 2; 10] were effectively used for accelerating probability computations for BNs. Darwiche et al.[6; 2] have shown an efficient method for compiling BNs into factored forms of Multi-Linear Functions (MLFs), whose evaluation and differentiation solves the exact inference problem. In their method, at first a given BN structure is encoded to a Conjunctive Normal Form (CNF) to be processed in the Boolean domain, and then the CNFs are factored according to Boolean algebra. The compilation procedure generates a kind of decision diagram representing a compact Arithmetic Circuit (AC) with symbolic parameters.

In this paper, we propose a new method of compiling BNs into factored MLFs based on Zero-suppressed Binary Decision Diagrams (ZBDDs)[7], which are the graph-based representation first used for VLSI logic design applications. Our method is based on a similar MLF modeling with symbolic parameters as well as Darwiche's approach. However, our method does not use the CNF representation but directly translates a BN into a set of factored MLFs. Our ZBDD manipulator can generate a new ZBDD as the result of addition/multiplication operations between pairs of ZBDDs. Using such inter-ZBDD operations in a bottom-up manner according to the BN structure, we can produce a set of ZBDDs each of which represents the MLF of each BN node. An important property of our method is that the total product of the ZBDDs for all BN nodes corresponds to the factored MLF, which is basically equivalent to Darwiche's result. Additionally, in our method it is not necessary to calculate the MLF for the whole network, as we can extract MLFs for only the part of the network related to the query, to avoid unnecessary calculation of redundant terms in the MLFs.

In this paper, we show experimental results for some typical benchmark examples. Although our algorithm is simply based on the mathematical definition of probability calculations, performance is competitive to existing state-of-the-art methods.

Our ZBDD-based method can also be compared with recent work by Sanner and McAllester[10], computing BN probabilities using Affine Algebraic DDs (AADDs). Their method generates AADDs as the result of inter-AADD operations for a given BN and an inference query. This is a similar approach to ours, but the semantics of the decision diagrams are quite different. We will discuss this difference in a later section.

We describe the basic concept of BN compilation and existing methods in Section 2. We then describe the ZBDD data structure for representing MLFs in Section 3. In Section 4, we describe the procedure of ZBDD generation and online inference. Experimental results are given in Section 5, followed by the conclusion.

2 Preliminary

Here we briefly review the method for compiling BNs.

2.1 Bayesian networks and MLFs

A *Bayesian network* (BN) is a directed acyclic graph. Each BN node has a network variable X whose domain is a discrete set of values. Each BN node also has a *Conditional Probability Table* (CPT) to describe the conditional probabilities of the value of X given the values of the parent nodes of the parent BN nodes. Figure 1 shows a small example of a BN with its CPTs.

A *Multi-Linear Function* (MLF)[5] consists of two types of variables, an *indicator variable* λ_x for each value $X = x$, and a *parameter variable* $\theta_{x|\mathbf{u}}$ for each CPT parameter $\mathrm{Prb}(x|\mathbf{u})$.
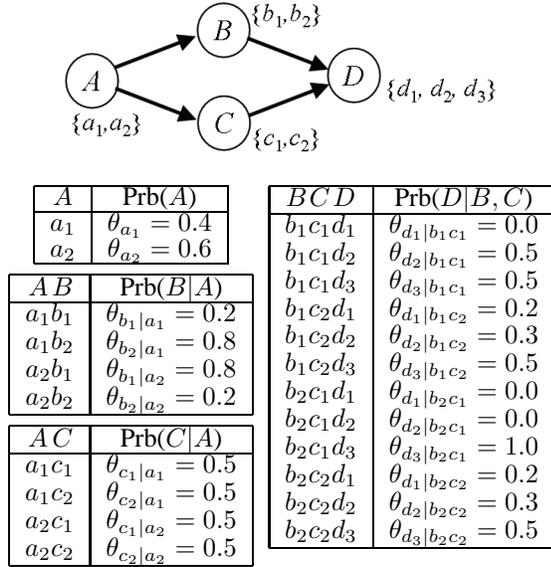
Figure 1: An example Bayesian network.

A	Prb(A)
a_1	$\theta_{a_1} = 0.4$
a_2	$\theta_{a_2} = 0.6$

$A\,B$	Prb($B|A$)
$a_1 b_1$	$\theta_{b_1|a_1} = 0.2$
$a_1 b_2$	$\theta_{b_2|a_1} = 0.8$
$a_2 b_1$	$\theta_{b_1|a_2} = 0.8$
$a_2 b_2$	$\theta_{b_2|a_2} = 0.2$

$A\,C$	Prb($C|A$)
$a_1 c_1$	$\theta_{c_1|a_1} = 0.5$
$a_1 c_2$	$\theta_{c_2|a_1} = 0.5$
$a_2 c_1$	$\theta_{c_1|a_2} = 0.5$
$a_2 c_2$	$\theta_{c_2|a_2} = 0.5$

$B\,C\,D$	Prb($D|B,C$)
$b_1 c_1 d_1$	$\theta_{d_1|b_1 c_1} = 0.0$
$b_1 c_1 d_2$	$\theta_{d_2|b_1 c_1} = 0.5$
$b_1 c_1 d_3$	$\theta_{d_3|b_1 c_1} = 0.5$
$b_1 c_2 d_1$	$\theta_{d_1|b_1 c_2} = 0.2$
$b_1 c_2 d_2$	$\theta_{d_2|b_1 c_2} = 0.3$
$b_1 c_2 d_3$	$\theta_{d_3|b_1 c_2} = 0.5$
$b_2 c_1 d_1$	$\theta_{d_1|b_2 c_1} = 0.0$
$b_2 c_1 d_2$	$\theta_{d_2|b_2 c_1} = 0.0$
$b_2 c_1 d_3$	$\theta_{d_3|b_2 c_1} = 1.0$
$b_2 c_2 d_1$	$\theta_{d_1|b_2 c_2} = 0.2$
$b_2 c_2 d_2$	$\theta_{d_2|b_2 c_2} = 0.3$
$b_2 c_2 d_3$	$\theta_{d_3|b_2 c_2} = 0.5$

The MLF contains a term for each instantiation of the BN variables, and the term is the product of all indicators and parameters that are consistent with the instantiation. For the example in Fig. 1, the MLF has the following forms:

$$\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_1|b_1 c_1}$$
$$+ \quad \lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_2}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_2|b_1 c_1}$$
$$+ \quad \lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_3}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_3|b_1 c_1}$$
$$+ \quad \lambda_{a_1}\lambda_{b_1}\lambda_{c_2}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_2|a_1}\theta_{d_3|b_1 c_2}$$
$$+ \quad \cdots$$
$$+ \quad \lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\lambda_{d_3}\theta_{a_2}\theta_{b_2|a_2}\theta_{c_2|a_2}\theta_{d_3|b_2 c_2}.$$

Once we have generated the MLF for a given BN, the probability of evidence **e** can be calculated by setting indicators that contradict **e** to 0 and other indicators to 1. Namely, we can calculate the probability in a linear time to the size of MLF. Obviously, the MLF has an exponential time and space complexity. The MLF can be factored into an *Arithmetic Circuit* (AC) whose size may not be exponential. If we can generate a compact AC for a given BN, the probability calculation can be greatly accelerated. This means compiling BNs based on MLFs.

2.2 Compiling BNs based on CNFs

Darwiche et al.[6] have found an efficient method for generating compact ACs without processing an exponential sized MLF. In their method, at first a given BN structure is encoded to a *Conjunctive Normal Form* (CNF) to be processed in the Boolean domain. The CNF is factored based on Boolean algebra, and a kind of decision diagram is obtained. The resulting diagram has a special property called *smooth deterministic Decomposable Negational Normal Form* (smooth d-DNNF)[4], so it can be directly converted to the AC for probability calculations.

In addition, the following two techniques are used in variable encoding:

- If a parameter is deterministic ($\theta_{x|\mathbf{u}} = 1$ or 0), we do

$$F1 = a \wedge \overline{b}$$
$$F2 = a \oplus b$$
$$F3 = \overline{b}$$
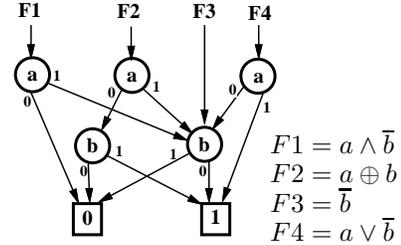$$F4 = a \vee \overline{b}$$

Figure 2: Shared multiple BDDs.

not assign a corresponding parameter variable and just reduce the CNF.

- Different parameter variables related to the same BN node do not coexist in the same term of the MLF. Therefore, if a CPT contains a number of parameters with the same probability, we do not have to distinguish them and may assign only one parameter variable to represent all those CPT parameters. This technique sometimes greatly reduces the size of the CNF.

Darwiche reported that their method succeeded in compiling large-scale benchmark networks, such as "pathfinder" and "Diabetes," with a practical usage of computational time and space. The BN compilation method is a hot topic in probabilistic modeling and inference for practical problems.

3 Zero-suppressed BDDs

In this paper, we present a new method of manipulating MLFs using Zero-suppressed Binary Decision Diagrams (ZBDDs). Here we describe our data structure.

3.1 ZBDDs and combinatorial item sets

A *Reduced Ordered Binary Decision Diagram* (ROBDD)[1] is a compact graph representation of the Boolean function. It is derived by reducing a binary tree graph representing the recursive *Shannon expansion*. ROBDDs provide canonical forms for Boolean functions when the variable order is fixed. (In the following sections, we basically omit "RO" from BDDs.) As shown in Fig. 2, a set of multiple BDDs can be shared with each other under the same fixed variable ordering. In this way, we can handle a number of Boolean functions simultaneously in a monolithic memory space.

A conventional BDD package supports a set of basic logic operations (i.e., AND, OR, XOR) for given a pair of operand BDDs. Those operation algorithms are based on hash table techniques, and the computation time is almost linear with data size unless the data overflows main memory. By using those inter-BDD operations, we can generate BDDs for given Boolean expressions or logic circuits.

BDDs were originally developed for handling Boolean function data, however, they can also be used for implicit representation of *combinatorial item sets*. A combinatorial item set consists of elements each of which is a combination of a number of items. There are 2^n combinations chosen from n items, so we have 2^{2^n} variations of combinatorial item sets. For example, for a domain of five items $a, b, c, d,$ and e, examples of combinatorial item sets are:

$$\{ab, e\}, \{abc, cde, bd, acde, e\}, \{1, cd\}, \emptyset.$$

Here "1" denotes a combination of no items, and $\emptyset$ means an empty set. Combinatorial item sets are one of the basic data structures for various problems in computer science.

a	b	c	F
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1	→ ab
0	0	1	1	→ c
1	0	1	1	→ ac
0	1	1	0
1	1	1	0

As a Boolean function:
$F(a,b,c) = (a\ b\ \sim c) \vee (\sim b\ c)$

As a combinatorial item set:
$S(a,b,c) = \{ab, ac, c\}$
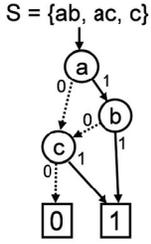
Figure 3: A Boolean function and a combinatorial item set.
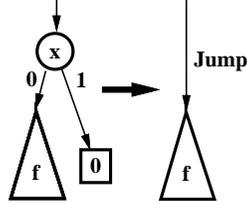

Figure 4: An example of a ZBDD.

Figure 5: ZBDD reduction rule.

A combinatorial item set can be mapped into Boolean space of n input variables. For example, Fig. 3 shows the truth table of the Boolean function $F = (a\ b\ \overline{c}) \vee (\overline{b}\ c)$, but also represents the combinatorial item set $S = \{ab, ac, c\}$, which is the set of input combinations for which F is 1. Using BDDs for the corresponding Boolean functions, we can implicitly represent and manipulate combinatorial item sets.

Zero-suppressed BDD (ZBDD)[7] is a variant of BDDs for efficient manipulation of combinatorial item sets. An example of a ZBDD is shown in Fig. 4. ZBDDs are based on the following special reduction rules.

- Delete all nodes whose 1-edge directly points to a 0-terminal node, and jump through to the 0-edge's destination, as shown in Fig. 5.
- Share equivalent nodes as well as ordinary BDDs.

The zero-suppressed deletion rule is asymmetric for the two edges, as we do not delete the nodes whose 0-edge points to a terminal node. It has been proved that ZBDDs also give canonical forms as well as ordinary BDDs under a fixed variable ordering. Here we summarize the properties of ZBDDs.

- Nodes of irrelevant items (never chosen in any combination) are automatically deleted by ZBDD reduction rule.
- Each path from the root node to the 1-terminal node corresponds to each combination in the set. Namely, the number of such paths in the ZBDD equals the number of combinations in the set.
- When many similar ZBDDs are generated, their ZBDD nodes are effectively shared into a monolithic multi-rooted graph, so the memory requirement is much less than storing each ZBDD separately.

Table 1 shows most of the primitive operations of ZBDDs. In these operations, $\emptyset$, **1**, P.top are executed in a constant time, and the others are almost linear to the size of the graph. We can describe various processing on sets of combinations by composing these primitive operations.

Table 1: Primitive ZBDD operations.

"$\emptyset$"	Returns empty set. (0-terminal node)
"**1**"	Returns the set with the null-combination. (1-terminal node)
P.top	Returns the item-ID at the root node of P.
P.factor0(v)	Subset of combinations not including item v.
P.factor1(v)	Gets $P - P$.factor0(v) and then deletes v from each combination.
P.attach(v)	Attaches v to all combinations in P.
$P \cup Q$	Returns union of P and Q.
$P \cap Q$	Returns intersection of P and Q.
$P - Q$	Returns difference sets. (in P but not in Q.)
P.count	Counts the number of combinations.

3.2 MLF representation using ZBDDs

An MLF is a polynomial formula of indicator and parameter variables. It can be regarded as a combinatorial item set, since each term is simply a combination of variables. For example, the MLF at Node B in Fig. 1 can be written as follows:

$$\begin{aligned} \mathrm{MLF}(B) &= \lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1} \\ &+ \lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2} + \lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2}. \end{aligned}$$

Here, we rename the parameter variables so that equal parameters share the same variable.

$$\begin{aligned} \mathrm{MLF}(B) &= \lambda_{a_1}\lambda_{b_1}\theta_{a(0.4)}\theta_{b(0.2)} + \lambda_{a_1}\lambda_{b_2}\theta_{a(0.4)}\theta_{b(0.8)} \\ &+ \lambda_{a_2}\lambda_{b_1}\theta_{a(0.6)}\theta_{b(0.8)} + \lambda_{a_2}\lambda_{b_2}\theta_{a(0.6)}\theta_{b(0.2)}. \end{aligned}$$

An example of the ZBDD for $\mathrm{MLF}(B)$ is shown in Fig. 6. In this example, there are four paths from the root node to the 1-terminal node, each of which corresponds to a term of the MLF. It is an implicit representation of the MLF. At the same time, the structure of ZBDD also represents a compact factored form of MLF. As shown in Fig. 7, each ZBDD decision node can be interpreted as a few AC nodes with the simple mapping rule. This means that a compact AC is quite easily obtained after generating a ZBDD for an MLF.

Our ZBDD representation for an MLF has the important property of being basically equivalent to a smooth d-DNNF, obtained by Darwiche's CNF-based method[6; 2]. In the following sections, we show our new method for generating ACs without CNFs but only using ZBDD operations.

3.3 Comparison with AADDs

Sanner and McAllester[10] presented *Affine Algebraic Decision Diagrams* (AADDs), another variant of a decision diagram, for computing BN probabilities. An AADD is a factored form of ADD, which contains indicator variables for splitting conditions, and the results of respective conditional probabilities are written in the leaves of the graph. This is somewhat similar to our approach since they generate AADDs as the result of algebraic operations of AADDs. The greatest difference to our method is that they numerically calculate the probability values with a floating-point data format, not by using symbolic probability parameters as in our MLFs. It is an interesting open problem whether it is more efficient to handle probabilities in symbolic or numerical form. It may well depend on the probability value instances for the specific problem at hand.
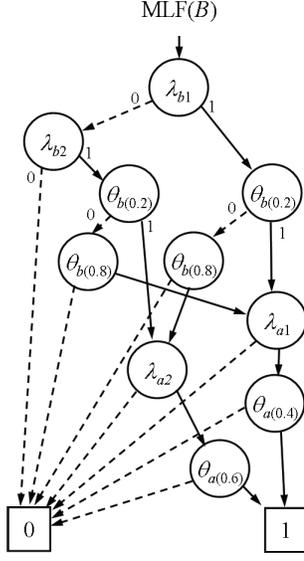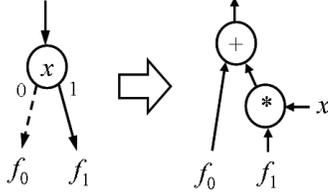
Figure 6: An example of a ZBDD for the MLF(B).

Figure 7: Mapping from a ZBDD node to an AC node.

4 ZBDD-based MLF calculation

4.1 ZBDD construction procedure

BDDs were originally developed for VLSI logic circuit design[1], and the basic technique of BDD construction is shown in Fig. 8. First we make trivial BDDs for the primary inputs F_1 and F_2, and then we apply the inter-BDD logic operations according to the data flow, to generate BDDs for F_3 to F_7. After that, all BDDs are shared in a monolithic multi-rooted graph. This procedure is called *symbolic simulation* for the logic circuit.

Our ZBDD construction procedure for the MLF is similar to the symbolic simulation of logic circuits. The differences are:

- BNs do not only assume binary values. The MLF uses multiple variables at each node for respective values.
- The BN node is not a logic gate. The dependence of each node on its ancestors is specified by a CPT.

As shown in Fig. 9, we first make MLF(A), and then we generate MLF(B) and MLF(C) using MLF(A). Finally, we generate MLF(D) using MLF(B) and MLF(C). After the construction procedure, all MLFs for respective nodes are compactly represented by the shared ZBDDs.

For each BN node X, the MLF is calculated by the following operations using the MLFs at the parent nodes of X.

$$\mathrm{MLF}(X_i) = \lambda_{x_i} \cdot \sum_{\mathbf{u} \in \mathrm{CPT}(X)} \left(\theta_{x(P_{\mathbf{u}})} \cdot \prod_{Y_v \in \mathbf{u}} \mathrm{MLF}(Y_v) \right)$$
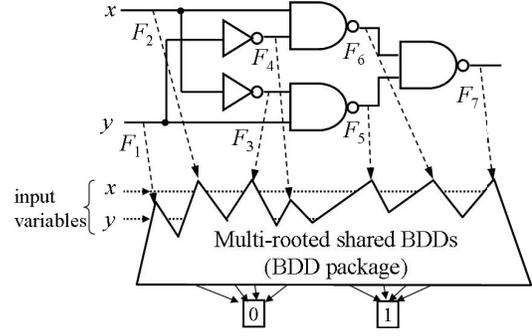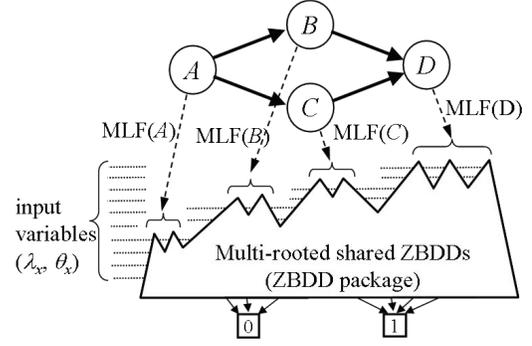

Figure 8: Conventional BDD construction procedure.

Figure 9: ZBDD construction procedure for BN.

Here $\mathrm{MLF}(X_i)$ denotes the value of MLF for the node X when X has the value x_i. Namely, $\mathrm{MLF}(X) = \sum_i \mathrm{MLF}(X_i)$.

When calculating this expression using ZBDD operations, we have to be mindful of the differences between conventional arithmetic algebra and combinatorial set algebra. Instead of the usual arithmetic sum, we may use *union* operations to perform the sum of MLFs, because MLFs cannot contain the same term more than once. We need to be more careful for the product operation. The product of two MLFs produces all possible combinations of two terms from the respective MLFs. However, no term can contain the same variable more than once, so instead of x^2 for duplicate variables, simply x will appear in the result. In addition, λ_{x_i} and λ_{x_j} (variables representing different values of the same node variable) cannot coexist in the same term as they are mutually exclusive.

4.2 Multi-valued multiplication algorithm

The multiplication algorithm is a key technique in our compiling method. The conventional algorithm for the product of two ZBDDs was developed by Minato[8]. A sketch of this algorithm is shown in Fig. 10. This algorithm is based on a divide-and-conquer approach, with two recursive calls for the subgraphs obtained by assigning 0 and 1 to the top variable. It also uses a hash-based cache technique to avoid duplicated recursive calls. The computation cost is almost linear with the ZBDD size.

Unfortunately, the conventional algorithm does not consider multi-valued variable encoding, so the result of ZBDD may contain redundant terms, such as the coexistence of both λ_{x_i} and λ_{x_j}. While such redundant MLFs are still correct

```
procedure(F · G)
{  if (F.top < G.top) return (G · F) ;
    if (G = ∅) return ∅ ;
    if (G = 1) return F ;
    H ← cache("F · G") ;
    if (H exists) return H ;
    v ← F.top ; /* the highest item in F */
    (F₀, F₁) ← factors of F by v ;
    (G₀, G₁) ← factors of G by v ;
    H ← ((F₁ · G₁) ∪ (F₁ · G₀) ∪ (F₀ · G₁)).attach(v)
       ∪(F₀ · G₀) ;
    cache("F · G") ← H ;
    return H ;
}
```

Figure 10: Conventional multiplication algorithm.

```
procedure(F · G)
{  if (F.top < G.top) return (G · F) ;
    if (G = ∅) return ∅ ;
    if (G = 1) return F ;
    H ← cache("F · G") ;
    if (H exists) return H ;
    v ← F.top ; /* the highest item in F */
    (F₀, F₁) ← factors of F by v ;
    (G₀, G₁) ← factors of G by v ;
    F_Z ← F₀ ; G_Z ← G₀ ;
    while(F_Z.top and v conflict) F_Z ← F_Z.factor0(F_Z.top) ;
    while(G_Z.top and v conflict) G_Z ← G_Z.factor0(G_Z.top) ;
    H ← ((F₁ · G₁) ∪ (F₁ · G_Z) ∪ (F_Z · G₁)).attach(v)
       ∪(F₀ · G₀) ;
    cache("F · G") ← H ;
    return H ;
}
```

Figure 11: Improved multiplication algorithm.

expressions for probability calculation, such redundant terms increase the computation time unnecessarily. For example, we analyzed our MLF construction for a BN in the benchmark set. It is a typical case that we have the two ZBDDs F and G, each of which has about 1,000 decision nodes, and the product $(F \cdot G)$ grows to have as many as 200,000 nodes of ZBDD, but can be reduced to only 400 nodes after eliminating redundant terms. To address this problem, we implemented an improved multiplication algorithm devoted to the multi-valued variable encoding. Figure 11 shows a sketch of the new algorithm. Here we assume that the indicator variables for the same BN node have consecutive positions in the ZBDD variable ordering. This algorithm does not produce any additional redundant terms in the recursive procedure, and we can calculate MLFs without any overhead due to multi-valued encoding.

4.3 Online inference based on ZBDDs

After the compilation procedure, each BN node has its own ZBDD for the MLF. The MLF(X) for node X contains only the variables of X's ancestor nodes, since no other variable is relevant to X's value.

Here we describe the online inference method based on ZBDDs. To obtain the joint probability for evidence $\mathbf{e}$, we first compute the product of MLF(X_v) for all $X_v \in \mathbf{e}$ by the ZBDD multiplication operation. Contradicting terms are automatically eliminated by our multiplication algorithm, so the result of ZBDD contains only the variables related to the joint probability computation. We then set all indicators to 1 and

calculate the AC directly from the ZBDD.

For example, to compute Prb(b_1, c_2) for the BN of Fig. 1, the two MLFs are the follows:
$$\text{MLF}(B_1) = \lambda_{a_1}\lambda_{b_1}\theta_{a(0.4)}\theta_{b(0.2)} + \lambda_{a_2}\lambda_{b_1}\theta_{a(0.6)}\theta_{b(0.8)},$$
$$\text{MLF}(C_2) = \lambda_{a_1}\lambda_{c_2}\theta_{a(0.4)}\theta_{c(0.5)} + \lambda_{a_2}\lambda_{c_2}\theta_{a(0.6)}\theta_{c(0.5)},$$
and then
$$\text{MLF}(B_1) \cdot \text{MLF}(C_2) = \lambda_{a_1}\lambda_{b_1}\lambda_{c_2}\theta_{a(0.4)}\theta_{b(0.2)}\theta_{c(0.5)}$$
$$+ \lambda_{a_2}\lambda_{b_1}\lambda_{c_2}\theta_{a(0.6)}\theta_{b(0.8)}\theta_{c(0.5)}.$$
Finally, we can obtain the probability as: $0.4 \times 0.2 \times 0.5 + 0.6 \times 0.8 \times 0.5 = 0.28$.

In our method, each multiplication requires a time almost linear in ZBDD size. However, the ZBDD size need not increase in repeating multiplications for the inference, because many of the terms contradict the evidence and are eliminated. Therefore, the computation cost for the inference will be much smaller than the cost for compilation.

An interesting point is that the above MLF for Prb(b_1, c_2) does not contain the variables for node D since they are irrelevant to the joint probability. In other words, our inference method provides dependency checking for a given query.

As another strategy, we can generate the MLF for the whole network by performing the product of all MLF(X). Such a global MLF is basically equivalent to the result of Darwiche's compilation[2]. After generating the global MLF, we no longer have to perform the product of MLFs. However, the MLF contains the parameters of all the BN nodes, and we should sum the parameters even if they are irrelevant to the query. Having a set of local MLFs will be more efficient than a global set since we can avoid unnecessary calculation of parameters not related to the query.

Finally, we note that our method can save the result of the partial product of MLFs in the shared ZBDD environment, so we do not have to recompute ZBDDs for the same evidence set.

5 Experimental results

For evaluating our method, we implemented a BN compiler based on our own ZBDD package. We used a Pentium-4 PC, 800 MHz, 1.5 GB of main memory, with SuSE Linux 9 and GNU C++ compiler. On this platform, we can manipulate up to 40,000,000 nodes of ZBDDs with up to 65,000 different variables. We applied our method to the practically sized BN examples provided at http://www.cs.huji.ac.il/labs/compbio/Repository.

The experimental results are shown in Table 2. In this table, the first four columns show the network specifications, such as BN name, the number of BN nodes, the indicator variables, and the parameter variables to be used in the MLF. The next three columns present the results of our compilation method. "|ZBDDs|(total)" shows the number of decision nodes in the multi-rooted shared ZBDDs representing the set of MLFs. "|ZBDD|(a node)" is the average size of ZBDD on each BN node. Notice that the total ZBDD size is usually much less than the numerical product of each ZBDD size because their sub-graphs are shared with each other.

After compilation, we evaluated the performance of online inference. In our experiment, we randomly select a pair of BN nodes with random values (x_i, y_j), then generate a ZBDD as the product of the two ZBDDs (MLF(X_i) · MLF(Y_j)). After that we counted the number of decision nodes "|ZBDD|(product)" and the number of the MLF terms. We repeated this process one hundred times and show the

Table 2: Experimental results.

BN name	BN nodes	indicator vars.	parameter vars.	offline compile			inference (ave. for 100 cases)			CNF-based[2](*)	
				|ZBDDs| (total)	|ZBDD| (a node)	time (sec)	|ZBDD| (product)	MLF terms	time (sec)	comp. time	inf. time
alarm	37	105	187	34,299	1,863	0.2	4,139	3.70×10^8	0.04	0.52	0.01
hailfinder	56	223	835	294,605	4,427	3.0	9,799	1.00×10^{17}	0.19	0.86	0.06
mildew	35	616	6,709	15,310,511	2,684,245	8019.4	593,469	6.60×10^{16}	43.43	7,483.80	3.35
pathfinder(pf1)	109	448	1,839	16,808	155	20.1	337	667	0.01	20.36	0.07
pathfinder(pf23)	135	520	2,304	17,557	135	19.6	188	212	0.01	(no data)	(no data)
pigs	441	1,323	1,474	73,543	237	2.9	993	3.27×10^7	0.01	17.84	1.60
water	32	116	3,578	25,629	611	6.1	974	6,295	0.02	4.83	0.21
diabetes	413	4,682	17,622	—	—	(>36k)	—	—	—	2,269.05	16.27
munin1	189	995	4,249	—	—	(>36k)	—	—	—	1,534.97	44.91
munin2	1,003	5,376	22,866	9,936,191	86,267	1,247.8	—	—	(>360)	225.46	6.59
munin3	1,044	5,604	24,116	11,191,778	100,640	777.7	—	—	(>360)	151.72	3.65
munin4	1,041	5,648	24,242	5,724,468	46,989	4,951.1	—	—	(>360)	677.92	7.70

(*) using a computer twice as fast as ours.

average time and space. The inference time shown here is the total time of ZBDD multiplication and traversing every decision node once in the ZBDD for calculating probability. From the experimental results, we can observe that the size of "|ZBDD|(product)" is dramatically smaller than "|ZBDDs|(total)." This indicates that we can avoid calculating many redundant terms of MLFs by using a product of local MLFs instead of the global MLF.

In the last two columns, we referred to the results of the CNF-based method[2]. For some examples, our results are competitive or better than theirs. Notice that we cannot directly compare our results to theirs because (1) the experimental setting of online inference would be different, (2) we may use a different variable ordering since it is not described in [2], and (3) for the larger examples, they applied another technique called *decomposition tree*(dtree)[3], to reduce the original network of CPTs.

Currently, our simple variable ordering strategy is that a variable appearing at an earlier stage of the calculation will get a lower position (nearer to the leaf) in ZBDDs. The ZBDD size is sometimes very sensitive to the variable ordering. For example, we have observed that the ZBDDs for "munin2" can be easily reduced by half using ad-hoc exchange of variable ordering. We expect that a good variable ordering will bring a significant improvement to the current results.

Our method is too time consuming for larger examples, such as "diabetes" and "munins". This could be because we have not applied the dtree-based CPT network reduction. This technique is independent of our ZBDD-based data structure, so we hope it will be effective as well as for the CNF-based method.

6 Conclusion

We have presented a new method of compiling BNs, not using CNF encoding but by directly calculating MLFs using ZBDDs. In our method, it is not necessary to generate MLFs for whole networks. We can extract MLFs for only those BN nodes related to the query, avoiding unnecessary calculation of redundant terms of MLFs. Our algorithm is quite simple and is based on the mathematical definition of probability calculation, but it still efficiently calculates exponential-sized MLFs using a compact ZBDD representation. Our method will be improved when combined with other state-of-the-art techniques developed in BN processing.

In this paper, we have shown that the BN compilation process is similar to the symbolic simulation of VLSI logic circuits. There have been many heuristic techniques on BDDs in the VLSI logic design area, and some of them would be useful for probabilistic inference on BNs or Markov Decision Processes.

References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., C-35, 8 (1986), 677–691.

[2] M. Chavira and A. Darwiche, "Compiling Bayesian Networks with Local Structure," *In Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pp. 1306–1312, Aug. 2005.

[3] A. Darwiche, "Recursive conditioning," *Artificial Intelligence*, vol. 126, No. 1–2, pp. 5–41, 2001.

[4] A. Darwiche, "A logical approach to factoring belief networks," *In Proc. KR*, pp. 409–420, 2002.

[5] A. Darwiche, "A differential approach to inference in Bayesian networks," *JACM*, Vol. 50, No. 3, pp. 280–305, 2003.

[6] A. Darwiche, "New advances in compiling CNF to decomposable negational normal form," *In Proc. European Conference on Artificial Intelligence (ECAI-2004)*, pp. 328–332, 2004.

[7] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," *In Proc. 30th Design Automation Conf. (DAC-93)*, pp. 272–277, Jun. 1993.

[8] S. Minato, "Zero-Suppressed BDDs and Their Applications," *International Journal on Software Tools for Technology Transfer*, Vol. 3, No. 2, pp. 156–170, Springer, May 2001.

[9] T. Nielsen, P. Wuillemin, F. Jensen, and U. Kjaerulff, "Using ROBDDs for inference in Bayesian networks with troubleshooting as an example," *In Proc. the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 426–435, 2000.

[10] S. Sanner and D. McAllester, "Affine Algebraic Decision Diagrams (AADDs) and their Application to Structured Probabilistic Inference," *In Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, Aug. 2005.