

## Propositionalizing the EM algorithm by BDDs

Masakazu Ishihata  
Yoshitaka Kameya  
Taisuke Sato  
Shin-ichi Minato

TR08-0004 June

DEPARTMENT OF COMPUTER SCIENCE  
TOKYO INSTITUTE OF TECHNOLOGY  
Ôokayama 2-12-1 Meguro Tokyo 152-8552, Japan  
<http://www.cs.titech.ac.jp/>

## Abstract

We propose two EM algorithms that work on binary decision diagrams (BDDs) and Zero-suppressed BDDs (ZBDDs) respectively. They open a way to applying BDDs and ZBDDs to statistical inference in general and machine learning in particular. We also present the complexity analysis of noisy-OR models based on BDDs and probabilistic context free grammars based on ZBDDs.

## 1 Introduction

*Binary decision diagrams* (BDDs) have been popular as a basic tool for compactly representing boolean functions [1, 2], and *Zero-suppressed BDDs* (ZBDDs), BDDs based on a different principle, are known as suitable to manipulate sparse truth assignments [11, 12]. In this paper we present yet another application of (Z)BDDs. We propose two new EM algorithms that work on BDDs and ZBDDs respectively. Since the EM algorithm is a fundamental parameter learning algorithm for maximum likelihood estimation in statistics [5], our proposal opens a way to apply (Z)BDDs to statistical learning in general and to machine learning in particular.

Both diagrams express boolean formulas compactly as a disjunction of exclusive disjuncts. To take further advantage of them however, we employ the decomposed BDD approach<sup>1</sup> with two features added. First we replace boolean variables by boolean random variables. We assume they are independent. Second we consider BDD fragments that constitute a decomposed BDD as subroutines. Consequently our new BDD-based data structure is comprised of a set of probabilistic BDD fragments that recursively “call” other fragments as subroutines. The purpose of this change is to gain exponential compression effect without which computing probabilistic context free grammars (PCFGs) [8] would be impossible in polynomial time. See Section 4 for more details. We similarly introduce a decomposed version of ZBDDs.

The two new EM algorithms, the *BDD-EM algorithm* and the *ZBDD-EM algorithm*<sup>2</sup> we develop achieve *generality* and *efficiency*. By generality we mean that they are applicable to arbitrary probabilistic boolean formulas, and thereby applicable to a variety of probabilistic models ranging from discrete Bayesian networks (BNs) to PCFGs which are representable by probabilistic boolean formulas. Nonetheless we can oftentimes verify that the achieved generality does not sacrifice computational efficiency. For example we prove in Section 4 that the probability computation of sentences in PCFGs performed by the ZBDD-EM algorithm is done in  $O(L^3)$  time where  $L$  is the sentence length, which is equal to the standard time complexity.

The significance of the proposed algorithms is twofold. First it establishes a connection between (Z)BDDs and EM learning for the first time as far as we know. (Z)BDDs can now be considered as a statistical learning device. Second it removes the traditional restriction to Horn clauses of logical formulas used in logic-based formalisms combined with probabilistic inference [15, 18, 4]. The proposed algorithms accept non-Horn clauses as well as Horn clauses, and thus contribute to expanding the realm of logic-based statistical formalisms.

In what follows, after reviewing the EM algorithm and BDDs in Section 2, we present the (Z)BDD-EM algorithm in Section 3. Examples of complexity analysis are shown in Section 4. Section 5 contains a simple learning experiment. Related work is described in Section 6 and Section 7 is conclusion.

---

<sup>1</sup>Given a boolean function  $F$ , we decompose it into smaller functions, *decomposed functions*, then build BDDs for each of them and finally put those BDDs together to represent  $F$ . This idea was used in VLSI logic design area to avoid the blow-up of the size of monolithic BDDs [7]. In this paper we call a BDD built for each decomposed function a *BDD fragment* for  $F$ , and the set of all BDD fragments for  $F$  simply a *decomposed BDD* for  $F$ .

<sup>2</sup>They have different characteristics and should be applied depending on the problem to be solved. For convenience, they are collectively referred to as the (Z)BDD-EM algorithm.

## 2 Preliminaries

### 2.1 The EM algorithm

In this section, we describe our learning setting and review the expectation-maximization (EM) algorithm [5]. First of all, we assume our problem domain is modeled with  $k$  boolean *random* variables  $X_1, X_2, \dots, X_k$ , each taking 1 (true) and 0 (false) independently of each other.<sup>3</sup> Let  $F$  be a boolean function composed of these  $k$  variables, and assume only the value of  $F$  is observable whereas those of the  $X_i$ 's are not. Hereafter, to make notations simple,  $F$  is treated as a boolean random variable as well that takes the value of (the function)  $F$ . We then call  $F$  an *observable variable*, and the  $X_i$ 's *basic variables*. For example, consider a boy who usually goes to school by bicycle, but takes a bus on a rainy day. It sometimes happens that he is late for school. We formalize his being late as  $F \Leftrightarrow O \vee (R \wedge B)$ , where  $F$ ,  $O$ ,  $R$  and  $B$  respectively indicate “he is late for school,” “he oversleeps,” “it rains,” and “the bus is late.” The EM algorithm proposed in this paper aims to estimate the probabilities of basic variables ( $O$ ,  $R$  and  $B$ ) being true from an observed value of  $F$ .<sup>4</sup> Note that the setting here is general and the boolean formula defining  $F$  is arbitrary. Also note that some of the  $X_i$ 's are allowed to be i.i.d. just like hidden Markov models (HMMs) have i.i.d. variables across the time slices. More formally, a set of basic variables  $\mathbf{X} = \{X_1, X_2, \dots, X_k\}$  is partitioned into  $\mathcal{S}$ , sets of i.i.d. variables, and each partition  $s \in \mathcal{S}$ , called an *i.i.d. group*, is given a *parameter*  $\theta_{s,x}$ , a probability of  $X \in s$  taking  $x$  (i.e.  $\sum_{x \in \{0,1\}} \theta_{s,x} = 1$ ).  $\mathcal{S}$  is in one-to-one correspondence to the set of parameters. We designate the vector of parameters  $\{\theta_{s,0}, \theta_{s,1}\}$  by  $\theta_s$ , and the vector of all parameters by  $\theta$ .

Let  $\phi$  be an assignment of basic variables  $\mathbf{X}$ , which maps a variable  $X \in \mathbf{X}$  to its value  $x \in \{0, 1\}$ . We use  $\Phi$  to stand for the set of all assignments. Since  $f \in \{0, 1\}$ , the value of  $F$ , is uniquely determined by  $\phi$ , an assignment,  $F$  is a function over assignments such that  $F(\phi) = f$ . We rewrite the set of assignments which make  $F = f$  as  $F^{-1}(f) = \{\phi \in \Phi \mid F(\phi) = f\}$ . For an assignment  $\phi$  and a parameter  $\theta_{s,x}$ , we introduce  $\sigma_{s,x}(\phi) = |\{X \in s \mid \phi(X) = x\}|$ , the number of occurrences in  $\phi$  of the variables with common parameters  $\theta_s$ , taking a value  $x$ . The EM algorithm we develop for the setting described above consists of two steps, the *expectation step* (E-step) and the *maximization step* (M-step), defined as follows:

- **E-step:** Compute the *conditional expectation*  $E_{\theta}[\sigma_{s,x}(\cdot) \mid F = f]$  by  $\eta_{\theta}^x[s] / P_{\theta}(F = f)$ , where:

$$\eta_{\theta}^x[s] = \sum_{\phi \in F^{-1}(f)} \sigma_{s,x}(\phi) \prod_{s' \in \mathcal{S}} \prod_{x' \in \{0,1\}} \theta_{s',x'}^{\sigma_{s',x'}(\phi)} \quad (1)$$

$$P_{\theta}(F = f) = \sum_{\phi \in F^{-1}(f)} \prod_{s \in \mathcal{S}} \prod_{x \in \{0,1\}} \theta_{s,x}^{\sigma_{s,x}(\phi)}. \quad (2)$$

- **M-step:** Update  $\theta$  to  $\hat{\theta}$  by  $\hat{\theta}_{s,x} \propto E_{\theta}[\sigma_{s,x}(\cdot) \mid F = f]$ .

The problem with this algorithm is that  $|F^{-1}(f)|$  grows exponentially in the number of variables. Our answer is to run the E-step in a dynamic programming manner on compact data structure, i.e. (Z)BDDs to combat this problem.

<sup>3</sup>This independence assumption does not put a restriction on modeling dependencies. We can always use a random variable indicating “A has a headache if A caught a cold,” which represents a dependency between a disease and a symptom.

<sup>4</sup>This single observation assumption is just for the sake of simplicity. There is no difficulty in extending the proposed algorithm(s) to multiple observations, i.e. a random sample of  $F$ . Also we can consider other situations. For example, some basic variables may be observed as evidences, or one may wish to infer the most probable reason why the boy is late based on the estimated probabilities. These situations are easy to handle, and hence we will concentrate on the *unsupervised* learning setting for the single observation case.

## 2.2 Binary Decision Diagrams

A BDD is a rooted directed acyclic graph representing a boolean function  $F$  as a disjunction of exclusive conjunctions. It has some *non-terminal nodes* and two special *terminal nodes* called 0-terminal and 1-terminal, denoted by  $\boxed{1}$  and  $\boxed{0}$ , respectively. A non-terminal node  $n$  labeled with a boolean variable, say,  $X$ , appearing in the function  $F$  is referred to as  $Label(n)$ . Note that some nodes may have the same label, i.e. they stand for the same variable. Also  $n$  has two child nodes, the 1-child and the 0-child, denoted by  $Ch^1(n)$  and  $Ch^0(n)$ . An edge from a node  $n$  to the 0-child (resp. the 1-child) is called the 0-edge (resp. the 1-edge) from  $n$ , indicating an assignment of 0 (resp. 1) to the variable  $Label(n)$  labeling  $n$ . We usually depict 1-edges (resp. 0-edges) by arrows with solid (resp. dashed) line. A path from the root node to  $\boxed{1}$  (resp.  $\boxed{0}$ ) represents an assignment  $\phi$  making  $F = 1$  (resp.  $F = 0$ ). When a BDD has no node irrelevant to a value of  $F$ , the BDD is said to be *reduced*. Also if each variable occurs once in each path from the root to a terminal and if the order of the occurrences is common to all these paths, it is said to be *ordered* [2].

A reduced ordered BDD (ROBDD) is known to be a unique representation of the target boolean function. Fig. 1 illustrates some different representations of a boolean function  $F = (A \vee B) \wedge \bar{C}$ . Fig. 1 (a) is a truth table, in which a row corresponds to an assignment  $\phi$  for  $\mathbf{X} = \{A, B, C\}$ . One way to obtain the ROBDD for  $F$  is to consider a binary decision tree (BDT, a special form of a BDD) as shown in Fig. 1 (b) and apply the following two reduction rules as many times as possible:

1. **Deletion rule:** If  $n$  is a non-terminal node in the BDD such that  $Ch^1(n) = Ch^0(n)$ , delete  $n$  from the BDD (Fig. 2).
2. **Merging rule:** If  $n, n'$  are two non-terminal nodes such that  $Label(n) = Label(n')$ , put  $Ch^1(n) = Ch^1(n')$  and  $Ch^0(n) = Ch^0(n')$ , and merge  $n$  and  $n'$  (Fig. 3).

Fig. 1 (d) is the ROBDD for  $F$ , whereas Fig. 1 (c) is a BDD obtained by only applying the merging rule. The complexity issue will be discussed in Section 4.

## 3 Combining BDDs and the EM algorithm

### 3.1 Overview

Before presenting the (Z)BDD-EM algorithm, an EM algorithm based on (Z)BDDs, we specifically remark three points, while introducing some notations.

#### 3.1.1 Three level probability/expectation computations

Let us recall that i.i.d. boolean random variables share a parameter. Also recall that some nodes in a (Z)BDD may share a boolean random variable. So in our algorithm(s), the computation of probabilities or expectations is describable in three levels — the parameter level, the variable level, and the node level. To illustrate this, look at the product in Eq. 1 computed at the parameter level, along with i.i.d. groups  $\mathcal{S}$  where all  $\sigma_{s,x}(\phi)$ s are given. Now rewrite Eq. 1 as:<sup>5</sup>

$$\eta_{\theta}^x[s] = \sum_{\phi \in F^{-1}(f)} \sum_{X \in \mathbf{X}: X \in s} \mathbf{1}_{\phi(X)=x} \prod_{Z \in \mathbf{X}} \theta_{[Z]}^{\phi(Z)} \theta_{[\bar{Z}]}^{1-\phi(Z)}, \quad (3)$$

<sup>5</sup>Eq. 2 is similarly rewritten to  $P(F=f) = \sum_{\phi \in F^{-1}(f)} \prod_{Z \in \mathbf{X}} \theta_{[Z]}^{\phi(Z)} \theta_{[\bar{Z}]}^{1-\phi(Z)}$ .

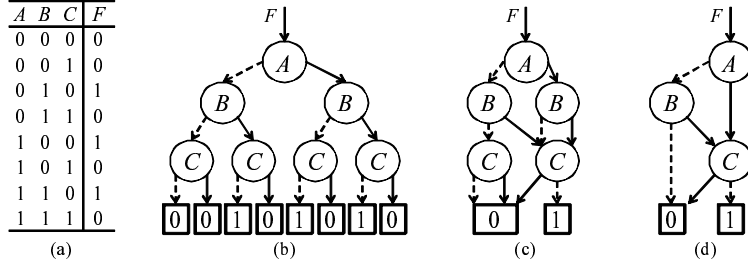


Figure 1: Examples of (a) a truth table, (b) a binary decision tree (BDT), (c) a BDD which is ordered but is not reduced, (d) the ROBDD, for  $F = (A \vee B) \wedge \bar{C}$ .

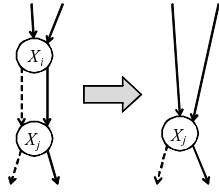


Figure 2: Deletion rule.

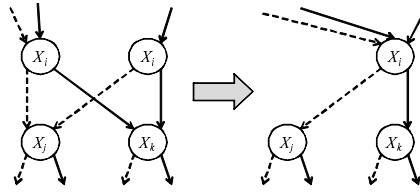


Figure 3: Merging rule.

where we refer to an i.i.d. group  $s \in \mathcal{S}$  by  $[X]$  as long as  $X$  is a variable belonging to  $s$ , and introduce  $\theta_{[X]} = \theta_{s,1} = P(X=1)$  and  $\theta_{[\bar{X}]} = \theta_{s,0} = P(X=0)$ .  $\mathbf{1}_{\phi(X)=x}$  takes 1 when  $\phi(X) = x$  is true, and 0 otherwise. This is the variable level description. Eq. 3 is procedurally interpreted that, for an assignment  $\phi$  making  $F = f$ , we add  $P(\phi) = \prod_{Z \in \mathbf{X}} \theta_{[Z]}^{\phi(Z)} \theta_{[\bar{Z}]}^{1-\phi(Z)}$  to  $\eta_{\theta}^x[s]$ , every time we find a variable  $X$  in  $\mathbf{X}$  such that  $X$  has parameters  $\theta_s$  and takes a value  $x$  under  $\phi$ . Note that, in Eq. 3,  $\sigma_{s,x}(\cdot)$  is not used explicitly, and the products are no longer computed along with  $\mathcal{S}$  but with variables  $\mathbf{X}$ .

We further elaborate the expectation computation above so that it runs on the nodes in a (Z)BDD. Consider a BDT like the one in Fig. 1 (b). In a BDT, there is a unique path  $\pi_{\phi}$  from the root to a terminal for an assignment  $\phi$ , in which every basic variable appears once as a node label. So we rewrite Eq. 3 as:

$$\eta_{\theta}^x[s] = \sum_{\pi_{\phi}: \phi \in F^{-1}(f)} \sum_{n' \in \pi_{\phi}: L_{n'} \in s} \mathbf{1}_{\phi(L_{n'})=x} \prod_{n \in \pi_{\phi}} \theta_{[L_n]}^{\phi(L_n)} \theta_{[\bar{L}_n]}^{1-\phi(L_n)}, \quad (4)$$

where  $n \in \pi_{\phi}$  says that the node  $n$  is on the path  $\pi_{\phi}$  and  $L_n = \text{Label}(n)$ . A similar interpretation for Eq. 4 is possible, i.e. we accumulate  $P(\phi)$  to  $\eta_{\theta}^x[s]$  at the  $x$ -edge from the node  $n$  with a variable  $L_n$ , the one labeling  $n$  which has parameters  $\theta_s$  and takes  $x$  under  $\phi$  ( $x \in \{0, 1\}$ ).  $P(\phi)$  is computed by multiplications along  $\pi_{\phi}$ . We extend this computation strategy to BDDs whose paths are compressed by the reduction rules.

### 3.1.2 Different treatments between probabilities and expectations

We need special care in computing expectations, which is unnecessary in computing probabilities. To see this, consider again the boolean formula  $F \Leftrightarrow (A \vee B) \wedge \bar{C}$  and see Fig. 1. The truth table in Fig. 1 (a) says that  $P(F=1)$  is computed by  $\theta_{[\bar{A}]} \theta_{[B]} \theta_{[\bar{C}]} + \theta_{[A]} \theta_{[\bar{B}]} \theta_{[\bar{C}]} + \theta_{[A]} \theta_{[B]} \theta_{[\bar{C}]}$ . On the other hand, our generalization of the backward procedure for HMMs, which will also be shown later, computes  $P(F=1)$  by recursively computing probabilities on all paths from

$\boxed{1}$ 's to the root, respecting the BDD structure. For instance, using the BDDs in Fig. 1 (c) and (d),  $P(F = 1)$  is computed as  $(\theta_{[\bar{A}]} \theta_{[B]} + \theta_{[A]}(\theta_{[\bar{B}]} + \theta_{[B]})) \theta_{[C]}$  and also as  $(\theta_{[\bar{A}]} \theta_{[B]} + \theta_{[A]}) \theta_{[C]}$ , respectively. Obviously they are identical, meaning that the deletion rule for BDDs is “safe” in computing  $P(F = f)$ . However, this does not hold when computing expectations. To clarify the issue, suppose, temporarily, that no boolean random variables share parameters. So a variable  $X$  is a unique member of  $[X]$ , and it holds for an assignment  $\phi$  that  $\sigma_{[X],x}(\phi) = 1$  if  $\phi(X) = x$  and  $\sigma_{[X],x}(\phi) = 0$  otherwise. Thus for  $X \in s$ , Eq. 3 is simplified as:

$$\eta_{\theta}^x[[X]] = \sum_{\phi \in F^{-1}(f)} \mathbf{1}_{\phi(X)=x} \prod_{Z \in \mathbf{X}} \theta_{[Z]}^{\phi(Z)} \theta_{[Z]}^{1-\phi(Z)} = P_{\theta}(F = f, X = x). \quad (5)$$

Then, let us compute  $\eta_{\theta}^1[[B]]$  for the observation  $F = 1$ . It follows from Eq. 5 and the truth table that  $\eta_{\theta}^1[[B]] = \theta_{[\bar{A}]} \theta_{[B]} \theta_{[C]} + \theta_{[A]} \theta_{[B]} \theta_{[\bar{C}]}$ , by picking up the rows satisfying  $F = 1$  and  $B = 1$ . On the other hand, there is no explicit information in the ROBDD in Fig. 1 (d) concerning  $\theta_{[A]} \theta_{[B]} \theta_{[\bar{C}]}$ , since the 1-edge from the node labeled  $A$  is directly connected to the node labeled  $C$ . This mismatch is caused by the deletion rule of BDDs, and hence we need to “numerically recover” the deleted nodes as if we did not use the deletion rule in the BDD construction like Fig. 1(c). Similar care must be taken for ZBDDs *both* in the probability computation and in the expectation computation, as described later.

### 3.1.3 Decomposed BDDs

Our target boolean function  $F$  is represented by a *decomposed BDD*. It is a collection of *BDD fragments* made up of basic variables and *intermediate variables* that represent other BDD fragments. In a probabilistic context, we assume the variables in a BDD fragment are independent of those in another BDD fragment.

Suppose  $F$  is a boolean function of a set of basic variables  $\mathbf{X}$ . We introduce a set of non-basic boolean variables  $\mathbf{U}$  together with an indexing map  $Level: \mathbf{U} \rightarrow \{1, 2, \dots, |\mathbf{U}|\}$  such that  $F \in \mathbf{U}$  and  $Level(F) = |\mathbf{U}|$ .  $\mathbf{U}$  becomes layered by this map and every  $X \in \mathbf{U}$  is a function of some of variables in  $\mathbf{X}$  and those in  $\mathbf{U}$  “below”  $X$ , i.e.  $\{Y \in \mathbf{U} \mid Level(Y) < Level(X)\}$ . These variables constitute a *BDD fragment* for  $X$ , denoted by  $\delta_X$ , which is a BDD defining  $X$ . We call an element of  $\mathbf{U} \setminus \{F\}$  an *intermediate variable*. We denote by  $\mathbf{N}(\delta_X)$  the set of all nodes in  $\delta_X$  and by  $\mathbf{V}(\delta_X)$  the set of all variables in  $\delta_X$  respectively. The set of all BDD fragments, given by  $\Delta_F = \{\delta_X \mid X \in \mathbf{U}\}$ , is called a *decomposed BDD*.<sup>6</sup> We introduce a local variable ordering in a BDD fragment  $\delta_X$  as an indexing map  $Ord_X$  from  $\mathbf{N}(\delta_X)$  to  $\{1, 2, \dots, |\mathbf{N}(\delta_X)|\}$ .<sup>7</sup> When  $Ord_X(A) < Ord_X(B)$ , we write it as  $A \prec_X B$ .  $Ord_X(n) = 1$  holds iff  $n$  is the root node. For brevity, we omit  $X$  from  $\prec_X$  if  $X$  is understood from the context. The purpose of adopting decomposed BDDs is to gain exponential compression effect without which computing PCFGs would be impossible in polynomial time.

Fig. 4 illustrates a decomposed BDD  $\Delta_F$ . We see  $\mathbf{U} = \{F, X, Y\}$  and  $\mathbf{X} = \{A, B, C, D, E\}$ . We also see  $\mathbf{V}(\delta_Y) = \{D, E\}$ , while  $\mathbf{N}(\delta_Y)$  contains three nodes, in which one has a label  $D$ , and each of the rest has a label  $E$ . The parameters of  $\mathbf{X}$  will be estimated by the (Z)BDD-EM algorithm when we observe  $F = f$ .

## 3.2 The BDD-EM algorithm

We here present the BDD-EM algorithm which is an EM algorithm working on BDDs. There are five auxiliary procedures for the procedure BDD-EM(), i.e. ITERATEEM(), GETBACKWARD( $Y$ ),

<sup>6</sup>Caveat: Multiple occurrences of a variable in a BDD refers to the same value while variables in different BDD fragments are independent even they bear the same name.

<sup>7</sup> $Ord$  is an indexing for variables in each BDD fragment, while  $Level$  is an indexing for BDD fragments.

GETFORWARD( $Y$ ), GETINSIDE( $Y$ ) and GETOUTSIDEEXPE( $Y$ ). In the following, we describe these procedures in turn, assuming that no variable shares a parameter with the others as in Eq. 5. Of course, this assumption is *only for explanatory purpose*, and the algorithm presented as pseudo code work for general cases.

```

1: Procedure: BDD-EM()
2:   Initialize all parameters  $\theta$ ;
3:   repeat
4:     ITERATEEM();
5:   until the parameters  $\theta$  converge;
6: end

```

```

1: Procedure: ITERATEEM()
2:    $\mathcal{D} = \Delta_F$ ;
3:   // E-step
4:   while  $\mathcal{D} \neq \phi$  do
5:      $Y = \operatorname{argmin}_{Y' \in \mathcal{D}} \operatorname{Level}(Y')$ ;
6:     GETBACKWARD( $Y$ );
7:     GETFORWARD( $Y$ );
8:     GETINSIDE( $Y$ );
9:      $\mathcal{D} = \mathcal{D} \setminus \{Y\}$ ;
10:  end while
11:   $\mathcal{D} = \Delta_F$ ;
12:  INITIALIZEQ(), INITIALIZEETA();
13:  while  $\mathcal{D} \neq \phi$  do
14:     $Y = \operatorname{argmax}_{Y' \in \mathcal{D}} \operatorname{Level}(Y')$ ;
15:    GETOUTSIDEEXPE( $Y$ );
16:     $\mathcal{D} = \mathcal{D} \setminus \{Y\}$ ;
17:  end while
18:  // M-step
19:  for each  $s \in \mathcal{S}$  do
20:     $\theta_{s,1} \propto \eta_{\theta}^1[s] / \mathcal{P}_{\theta}^f[F]$ ;
21:     $\theta_{s,0} \propto \eta_{\theta}^0[s] / \mathcal{P}_{\theta}^f[F]$ ;
22:  end for
23: end

```

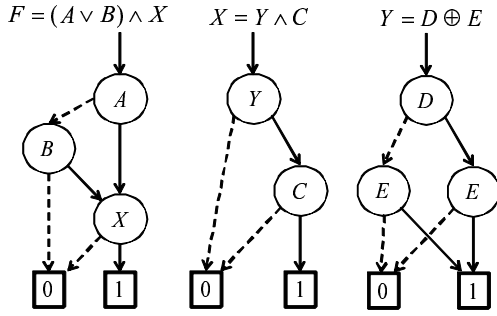


Figure 4: A decomposed BDD for  $F$ .

### 3.2.1 Backward and forward probabilities

The procedure GETBACKWARD( $Y$ ) calculates *backward probabilities* for each node in  $\mathbf{N}(\delta_Y)$ . A *backward probability*  $\mathcal{B}_{\theta}^1[n]$  is the sum of the probabilities of all paths from node  $n$  to  $\boxed{1}$ . Similarly  $\mathcal{B}_{\theta}^0[n]$  is the sum of the probabilities of all paths from node  $n$  to  $\boxed{0}$ . We set  $\mathcal{B}_{\theta}^1[\boxed{1}] = 1$  and  $\mathcal{B}_{\theta}^0[\boxed{0}] = 1$  respectively. Backward probabilities are calculated from terminals to the root. Contrastingly the procedure GETFORWARD( $Y$ ) calculates *forward probabilities* for each node in  $\mathbf{N}(\delta_Y)$  from the root to terminals. A *forward probability*  $\mathcal{F}_{\theta}[n]$  is the sum of the probabilities of all paths from the *root* to node  $n$ . The procedure INITIALIZEF() initializes  $\mathcal{F}_{\theta}[n] = 0$  for all  $n$ . We can see the time complexity of GETBACKWARD( $Y$ ) and GETFORWARD( $Y$ ) is  $O(N)$ , where  $N = |\mathbf{N}(\delta_Y)|$ .<sup>8</sup>

### 3.2.2 Inside probabilities

The procedure GETINSIDE( $Y$ ) calculates the *inside probability*  $\mathcal{P}_{\theta}^y[Y]$  of  $Y$ .  $\mathcal{P}_{\theta}^1[Y]$  (resp.  $\mathcal{P}_{\theta}^0[Y]$ ) is the probability of  $Y \in \mathbf{U}$  being true (resp. false) under the current parameters  $\theta$ . It is calculated as the sum of the probabilities of all paths from *root* to  $\boxed{1}$ .  $\mathcal{P}_{\theta}^0[Y]$  is calculated similarly except that  $\boxed{1}$  is replaced with  $\boxed{0}$ . So  $\mathcal{P}_{\theta}^1[Y] = P_{\theta}(Y=1) = \mathcal{B}_{\theta}^1[\text{root}]$  (resp.  $\mathcal{P}_{\theta}^0[Y] =$

<sup>8</sup>To be precise, the pseudo code of GETBACKWARD( $Y$ ) or GETFORWARD( $Y$ ) takes  $O(N^2)$  time. So actually we use a version that makes topological sorting on each BDD fragment in advance. Similar preprocessing is also taken for the other procedures.

<pre> 1: <b>Procedure:</b> GETBACKWARD(<math>Y</math>) 2:   <math>\mathcal{B}_\theta^1[\bar{1}] = 1, \mathcal{B}_\theta^1[\bar{0}] = 0;</math> 3:   <math>\mathcal{B}_\theta^0[\bar{1}] = 0, \mathcal{B}_\theta^0[\bar{0}] = 1;</math> 4:   <math>\mathcal{N} = \text{Par}(\bar{1}) \cup \text{Par}(\bar{0});</math> 5:   // <math>\text{Par}(n)</math>: the set of parents of <math>n</math>. 6:   <b>while</b> <math>\mathcal{N} \neq \phi</math> <b>do</b> 7:     <math>n = \text{argmax}_{n' \in \mathcal{N}} \text{Ord}_Y(n');</math> 8:     <math>X = \text{Label}(n);</math> 9:     <math>\mathcal{B}_\theta^1[n] = \theta_{[X]} \mathcal{B}_\theta^1[\text{Ch}^1(n)] + \theta_{[\bar{X}]} \mathcal{B}_\theta^1[\text{Ch}^0(n)];</math> 10:    <math>\mathcal{B}_\theta^0[n] = \theta_{[X]} \mathcal{B}_\theta^0[\text{Ch}^1(n)] + \theta_{[\bar{X}]} \mathcal{B}_\theta^0[\text{Ch}^0(n)];</math> 11:    <math>\mathcal{N} = \mathcal{N} \setminus \{n\} \cup \text{Par}(n);</math> 12:  <b>end while</b> 13: <b>end</b> </pre>	<pre> 1: <b>Procedure:</b> GETFORWARD(<math>Y</math>) 2:   INITIALIZEF(); 3:   <math>\mathcal{F}_\theta[\text{root}] = 1;</math> 4:   <math>\mathcal{N} = \{\text{root}\};</math> 5:   <b>while</b> <math>\mathcal{N} \neq \phi</math> <b>do</b> 6:     <math>n = \text{argmin}_{n' \in \mathcal{N}} \text{Ord}_Y(n');</math> 7:     <math>X = \text{Label}(n);</math> 8:     <math>\mathcal{F}_\theta[\text{Ch}^1(n)] += \mathcal{F}_\theta[n] \theta_{[X]};</math> 9:     <math>\mathcal{F}_\theta[\text{Ch}^0(n)] += \mathcal{F}_\theta[n] \theta_{[\bar{X}]};</math> 10:    <math>\mathcal{N} = \mathcal{N} \setminus \{n\} \cup \{\text{Ch}^1(n), \text{Ch}^0(n)\};</math> 11:  <b>end while</b> 12: <b>end</b>  1: <b>Procedure:</b> GETINSIDE(<math>Y</math>) 2:   <math>\mathcal{P}_\theta^1[Y] = \mathcal{B}_\theta^1[\text{root}];</math> 3:   <math>\mathcal{P}_\theta^0[Y] = 1 - \mathcal{B}_\theta^1[\text{root}];</math> 4: <b>end</b> </pre>
---	---

$P_\theta(Y=0) = 1 - \mathcal{B}_\theta^1[\text{root}]$ ).  $\mathcal{P}_\theta^f[F]$  is the likelihood of  $F = f$ . All inside probabilities being computed, we consider  $\mathcal{P}_\theta^1[Y]$  and  $\mathcal{P}_\theta^0[Y]$  as the “parameters” of  $Y$ , and denote them by  $\theta_{[Y]}$  and  $\theta_{[\bar{Y}]}$ , respectively.

### 3.2.3 Outside probabilities and conditional expectations

The procedure GETOUTSIDEEXPE( $Y$ ) updates  $\eta_\theta^x[[X]]$  for each  $X \in \mathbf{V}(Y) \cap \mathbf{X}$ , and  $\mathcal{Q}_\theta^z[Z]$  for each  $Z \in \mathbf{V}(Y) \cap \mathbf{U}$ .  $\eta_\theta^x[[X]]$  is defined in Section 2.1 and  $\mathcal{Q}_\theta^z[Z]$  is a temporary value used to calculate  $\eta_\theta^x[[X]]$ .  $\eta_\theta^x[[X]]$  are initialized as 0 for all  $X \in \mathbf{X}$  and  $x \in \{0, 1\}$  by the procedure INITIALIZEETA() in ITERATEEM(). Also,  $\mathcal{Q}_\theta^z[Z]$  are initialized as  $\mathcal{Q}_\theta^f[F] = 0$  and  $\mathcal{Q}_\theta^z[Z] = 0$  ( $Z \neq F$ ), where  $f \in \{1, 0\}$  is the observed value of  $F$ , by INITIALIZEQ(). To denote the nodes deleted by the deletion rule, we introduce  $\text{Del}_Y^1(n)$  and  $\text{Del}_Y^0(n)$  in GETOUTSIDEEXPE( $Y$ ).  $\text{Del}_Y^x(n)$  ( $x \in \{1, 0\}$ ) stands for the set of labels (i.e. variables) of deleted nodes between  $n$  and  $\text{Ch}^x(n)$ . So we have  $\text{Del}_Y^x(n) = \{X \in \mathbf{V}(\delta_Y) \mid \text{Label}(n) \prec X \prec \text{Label}(\text{Ch}^x(n))\}$ .

Here, we explain that the value of  $\eta_\theta^x[X]$  calculated by GETOUTSIDEEXPE( $Y$ ) coincides with the one in Eq. 5, i.e.  $P_\theta(X=x, F=f)$ . First consider the case  $Y = F$ .  $\mathcal{Q}_\theta^f[F]$  and  $\mathcal{Q}_\theta^f[F]$  are initialized to 1 and 0, respectively. We see  $e_n^1 = \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[\text{Ch}^f(n)] \theta_{[X]}$  and  $e_n^0 = \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[\text{Ch}^f(n)] \theta_{[\bar{X}]}$ , where  $X = \text{Label}(n)$ . From the definitions of  $\mathcal{F}_\theta[n]$  and  $\mathcal{B}_\theta^1[n]$ , it follows  $e_n^1$  is the sum of the probabilities of all paths from  $\text{root}$  to  $f$ -terminal through the 1-edge (resp. the 0-edge) of  $n$ , and thus equals  $P_\theta(F=f, X^n=1)$ . Here  $X^n$  is an auxiliary random variable on  $\Phi$ , the set of assignments for all variables.  $X^n$  takes  $x \in \{0, 1\}$  on  $\Phi' \subseteq \Phi$  where  $\Phi'$  is the set of assignments represented by paths through  $x$ -edge from  $n$  in the current BDD fragment. We let it take an arbitrary value other than  $\{0, 1\}$  on  $\Phi \setminus \Phi'$ . We further write the value of  $\eta_\theta^x[[X]]$  calculated by the procedure as

$$\eta_\theta^1[[X]] = \sum_{n \in \mathbf{N}_X(\delta_Y)} e_n^1 + \sum_{n \in \text{Del}_Y^1(X)} e_n^1 \theta_{[X]} + \sum_{n \in \text{Del}_Y^0(X)} e_n^0 \theta_{[X]}, \quad (6)$$

$$\eta_\theta^0[[X]] = \sum_{n \in \mathbf{N}_X(\delta_Y)} e_n^0 + \sum_{n \in \text{Del}_Y^1(X)} e_n^1 \theta_{[\bar{X}]} + \sum_{n \in \text{Del}_Y^0(X)} e_n^0 \theta_{[\bar{X}]}, \quad (7)$$

where  $\mathbf{N}_X(\delta_Y) = \{n \in \mathbf{N}(\delta_Y) \mid \text{Label}(n) = X\}$ ,  $\text{Del}_Y^1(X) = \{n \in \mathbf{N}(\delta_Y) \mid X \in \text{Del}_Y^1(n)\}$  and  $\text{Del}_Y^0(X) = \{n \in \mathbf{N}(\delta_Y) \mid X \in \text{Del}_Y^0(n)\}$ . Recall that, in BDDs, there may be a path  $\pi$  from which a node  $n$  labeled  $X$  has been deleted. Such a path is created by merging two paths, one including the 1-edge from  $n$  and the other including the 0-edge from  $n$ . Then,  $p_\pi$  being

```

1: Procedure: GETOUTSIDEEXPE( $Y$ )
2:   for each  $n \in \mathbf{N}(\delta_Y)$  do
3:      $X = \text{Label}(n)$ ;
4:      $e_n^1 = \mathcal{Q}_\theta^1[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^1(n)] \theta_{[X]}$ 
5:        $+ \mathcal{Q}_\theta^0[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^0[Ch^1(n)] \theta_{[X]}$ ;
6:      $e_n^0 = \mathcal{Q}_\theta^1[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^0(n)] \theta_{[\bar{X}]}$ 
7:        $+ \mathcal{Q}_\theta^0[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^0[Ch^0(n)] \theta_{[\bar{X}]}$ ;
8:     if  $X \in \mathbf{U}$  then
9:        $\mathcal{Q}_\theta^1[X] += e_n^1 / \theta_{[X]}$ ;
10:       $\mathcal{Q}_\theta^0[X] += e_n^0 / \theta_{[\bar{X}]}$ ;
11:     else if  $X \in \mathbf{X}$  then
12:        $\eta_\theta^1[[X]] += e_n^1$ ;
13:        $\eta_\theta^0[[X]] += e_n^0$ ;
14:     end if
15:     for each  $Z \in \text{Del}_Y^1(n)$  do
16:       if  $Z \in \mathbf{U}$  then
17:          $\mathcal{Q}_\theta^1[Z] += e_n^1$ ;
18:          $\mathcal{Q}_\theta^0[Z] += e_n^1$ ;
19:       else if  $Z \in \mathbf{X}$  then
20:          $\eta_\theta^1[[Z]] += e_n^1 \theta_{[Z]}$ ;
21:          $\eta_\theta^0[[Z]] += e_n^1 \theta_{[Z]}$ ;
22:       end if
23:     end for
24:     for each  $Z \in \text{Del}_Y^0(n)$  do
25:       if  $Z \in \mathbf{U}$  then
26:          $\mathcal{Q}_\theta^1[Z] += e_n^0$ ;
27:          $\mathcal{Q}_\theta^0[Z] += e_n^0$ ;
28:       else if  $Z \in \mathbf{X}$  then
29:          $\eta_\theta^1[[Z]] += e_n^0 \theta_{[Z]}$ ;
30:          $\eta_\theta^0[[Z]] += e_n^0 \theta_{[Z]}$ ;
31:       end if
32:     end for
33:   end for
34: end

```

```

1: Procedure: GETOUTSIDEEXPE*( $Y$ )
2:   for each  $n \in \mathbf{N}(\delta_Y)$  do
3:      $X = \text{Label}(n)$ ;
4:      $e_n^1 = \mathcal{Q}_\theta^1[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^1(n)] \theta_{[X]}$ 
5:        $+ \mathcal{Q}_\theta^0[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^0[Ch^1(n)] \theta_{[X]}$ ;
6:      $e_n^0 = \mathcal{Q}_\theta^1[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^0(n)] \theta_{[\bar{X}]}$ 
7:        $+ \mathcal{Q}_\theta^0[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^0[Ch^0(n)] \theta_{[\bar{X}]}$ ;
8:     if  $X \in \mathbf{U}$  then
9:        $\mathcal{Q}_\theta^1[X] += e_n^1 / \theta_{[X]}$ ;
10:       $\mathcal{Q}_\theta^0[X] += e_n^0 / \theta_{[\bar{X}]}$ ;
11:     else if  $X \in \mathbf{X}$  then
12:        $\eta_\theta^1[[X]] += e_n^1$ ;
13:        $\eta_\theta^0[[X]] += e_n^0$ ;
14:     end if
15:      $X' : \text{Ord}_Y(X') = \text{Ord}_Y(X) + 1$ 
16:      $\zeta[X'] += e_n^1 + e_n^0$ ;
17:      $\zeta[\text{Label}(Ch^1(n))] -= e_n^1$ ;
18:      $\zeta[\text{Label}(Ch^0(n))] -= e_n^0$ ;
19:   end for
20:    $X = \text{Label}(\text{root})$ ;
21:    $\mathcal{V} = \mathbf{V}(\delta_Y) \setminus \{X\}$ ,  $T = 0$ ;
22:   while  $\mathcal{V} \neq \emptyset$  do
23:      $X = \text{argmin}_{X' \in \mathcal{V}} \text{Ord}_Y(X')$ ;
24:      $T += \zeta[X]$ ;
25:     if  $X \in \mathbf{U}$  then
26:        $\mathcal{Q}_\theta^1[X] += T$ ;
27:        $\mathcal{Q}_\theta^0[X] += T$ ;
28:     else if  $X \in \mathbf{X}$  then
29:        $\eta_\theta^1[[X]] += T \theta_{[X]}$ ;
30:        $\eta_\theta^0[[X]] += T \theta_{[\bar{X}]}$ ;
31:     end if
32:      $\mathcal{V} = \mathcal{V} \setminus \{X\}$ ;
33:   end while
34: end

```

the probability of the path  $\pi$ , the probabilities of the recovered paths are computed as  $p_\pi \theta_{[X]}$  and  $p_\pi \theta_{[\bar{X}]}$ . Therefore, Eq. 6 (resp. Eq. 7) says that  $\eta_\theta^1[[X]]$  (resp.  $\eta_\theta^0[[X]]$ ) is the sum of the probabilities of the recovered paths and the other paths from  $\text{root}$  to  $\boxed{1}$  that include the 1-edge (resp. the 0-edge) of nodes labeled  $X$ . So, noting  $\mathcal{Q}_\theta^z[Z] = P_\theta(F=f, Z=z) / \theta_{[Z]}$ , we conclude

$$\eta_\theta^x[[X]] = P_\theta(X=x, F=f), \quad \mathcal{Q}_\theta^z[Z] = P_\theta(F=f \mid Z=z).$$

Consequently, the value of  $\eta_\theta^x[[X]]$  calculated by GETOUTSIDEEXPE( $F$ ) coincides with  $P_\theta(X=x, F=f)$ .

Now let us consider GETOUTSIDEEXPE( $Y$ ), where  $Y$  is an intermediate variable. Suppose  $\mathcal{Q}_\theta^y[Y]$  has already been calculated and equals  $P_\theta(F=f \mid Y=y)$ . Similarly to the above,  $\mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^1(n)] \theta_{[X]}$  (resp.  $\mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^0(n)] \theta_{[X]}$ ) amounts to  $P_\theta(Y=1, X^n=1)$  (resp.  $P_\theta(Y=0, X^n=1)$ ).

Hence, noting that  $F$  and  $X^n$  are conditionally independent when  $Y$  is given, we have

$$\begin{aligned}
e_n^1 &= \mathcal{Q}_\theta^1[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^1[Ch^1(n)] \theta_{[X]} + \mathcal{Q}_\theta^0[Y] \mathcal{F}_\theta[n] \mathcal{B}_\theta^0[Ch^1(n)] \theta_{[X]} \\
&= P_\theta(F=f | Y=1) P_\theta(Y=1, X^n=1) + P_\theta(F=f | Y=0) P_\theta(Y=0, X^n=1) \\
&= P_\theta(F=f | Y=1, X^n=1) P_\theta(Y=1, X^n=1) \\
&\quad + P_\theta(F=f | Y=0, X^n=1) P_\theta(Y=0, X^n=1) = P_\theta(F=f, X^n=1), \\
e_n^0 &= P_\theta(F=f, X^n=0),
\end{aligned}$$

where  $e_n^1$  and  $e_n^0$  respectively represent the quantities in Lines 5-8 in GETOUTSIDEEXPE( $F$ ). It follows  $\eta_\theta^x[[X]] = \sum_n P_\theta(F=f, X^n=x) = P_\theta(F=f, X=x)$  and  $\mathcal{Q}_\theta^z[Z] = P_\theta(F=f | Z=z)$ , and hence the value of  $\eta_\theta^x[[X]]$  calculated by GETOUTSIDEEXPE( $Y$ ) is the intended one.

The time complexity of GETOUTSIDEEXPE( $Y$ ) is  $O(N(D_1 + D_0))$  where  $N = |\mathbf{N}(\delta_Y)|$ ,  $D_1 = \max_{n \in \mathbf{N}(\delta_Y)} Del_Y^1(n)$  and  $D_0 = \max_{n \in \mathbf{N}(\delta_Y)} Del_Y^0(n)$ . This is because, for each  $n \in \mathbf{N}(\delta_Y)$ , the procedure updates  $\eta_\theta^x[[X]]$  where  $X = Label(n)$  and  $\eta_\theta^z[[Z]]$  for  $Z \in Del_Y^1(n) \cup Del_Y^0(n)$ . Furthermore we replace GETOUTSIDEEXPE( $Y$ ) with GETOUTSIDEEXPE\*( $Y$ ) to reduce the time complexity to  $O(N + V)$  where  $V = |\mathbf{V}(\delta_Y)|$ . In GETOUTSIDEEXPE( $Y$ ),  $\eta_\theta^x[[X]]$  is updated for each node and for each deleted node labeled  $X$ , whereas in GETOUTSIDEEXPE\*( $Y$ ), we have only to update  $\eta_\theta^x[[X]]$  and  $\zeta[X]$  for each node, where  $\zeta[X]$  is the quantity satisfying

$$\sum_{Z \in \mathbf{V}(\delta_Y): Z \prec X} \zeta[X] = \sum_{n \in Del_Y^1(X)} e_n^1 + \sum_{n \in Del_Y^0(X)} e_n^0.$$

$\sum_{Z \in \mathbf{V}(\delta_Y): Z \prec X} \zeta[X] \theta_{[X]}$  equals the sum of the second and the third terms of Eq. 6 and the value of  $\eta_\theta^x[[X]]$  calculated by GETOUTSIDEEXPE\*( $Y$ ) equals the one calculated by GETOUTSIDEEXPE( $Y$ ). Finally the total time complexity of executing the E-step is  $O(D(N_{\max} + V_{\max}))$ , where  $D = |\Delta_F|$ ,  $N_{\max} = \max_{Y \in \Delta_F} |\mathbf{N}(\delta_Y)|$  and  $V_{\max} = \max_{Y \in \Delta_F} |\mathbf{V}(\delta_Y)|$ .

### 3.3 The ZBDD-EM algorithm

There are some problems BDDs are not good at dealing with. Truth assignments in which variables are sparsely true (e.g. boolean functions where exclusive ORs are dominantly used [12]) are one of them. ZBDDs, BDDs based on a different principle, are better suited for such sparse truth assignments. In this section, we present the ZBDD-EM algorithm, an adaptation of the BDD-EM algorithm to ZBDDs. ZBDDs are based on two reduction rules like BDDs but the deletion rule of ZBDDs differs from that of BDDs:

1. **Deletion rule:** If  $n$  is a non-terminal node in the ZBDD such that  $Ch^1(n) = \boxed{0}$ , eliminate  $n$  from the ZBDD.

This rule makes the probability computation more complicated however. For example, think of  $Del_Y^1(n) = \{X_i, X_j\}$ . There are two nodes, labeled  $X_i$  and labeled  $X_j$ , between  $n$  and  $Ch^1(n)$ , deleted by the deletion rule above. The probability from  $n$  to  $Ch^1(n)$  is not  $\theta_{[Label(n)]}$  but  $\theta_{[Label(n)]} \theta_{[\bar{X}_i]} \theta_{[\bar{X}_j]}$  because the deleted variables are supposed to have taken 0 in the ZBDD. To compute probabilities correctly even in the presence of deleted variables, we introduce *deletion parameters*  $\theta_{Del_Y^1(n)}$  and  $\theta_{Del_Y^0(n)}$  for  $n \in \mathbf{N}(F)$  as follows:

$$\theta_{Del_Y^1(n)} = \prod_{X \in Del_Y^1(n)} \theta_{[\bar{X}]} \quad \theta_{Del_Y^0(n)} = \prod_{X \in Del_Y^0(n)} \theta_{[\bar{X}]}.$$

Then the probability from  $n$  to  $Ch^1(n)$  is computed as  $\theta_{[Label(n)]} \theta_{Del_Y^1(n)}$ . While the deletion rule of BDDs merely merges two paths into a single path, the deletion rule of ZBDDs completely

eliminates the path including a 1-edge directly pointing  $\square$ . So we have to recover these deleted paths for the correct probability/expectation computation.

Taking the above into account, we rewrite the BDD-EM algorithm to the ZBDD-EM algorithm. First, two procedures GETFORWARDZ( $Y$ ) and GETBACKWARDZ( $Y$ ) calculating  $\mathcal{F}_{Z,\theta}[n]$  and  $\mathcal{B}_{Z,\theta}^1[n]$ , respectively, are introduced corresponding to GETFORWARD( $Y$ ) and GETBACKWARD( $Y$ ) by replacing  $\theta_{[X]}$  (resp.  $\theta_{[\bar{X}]}$ ) with  $\theta_{[X]}\theta_{Del_Y^1(n)}$  (resp.  $\theta_{[X]}\theta_{Del_Y^0(n)}$ ). We also replace GETOUTSIDEEXPE\*( $Y$ ) with GETOUTSIDEEXPEZ( $Y$ ). These replacements enable to run EM algorithm on ZBDDs without increasing the computation time of the E-step, though we have to omit further details due to the space limitation. In conclusion, we should choose an appropriate data structure, BDD or ZBDD, depending on problems by considering their time complexity  $O(D(N_{\max} + V_{\max}))$  where  $D = |\Delta_F|$ ,  $N_{\max} = \max_{Y \in \Delta_F} |\mathbf{N}(\delta_Y)|$  and  $V_{\max} = \max_{Y \in \Delta_F} |\mathbf{V}(\delta_Y)|$ .

```

1: Procedure: GETOUTSIDEEXPEZ( $Y$ )
2:   INITIALIZEQ(), INITIALIZEETA();
3:   for each  $n \in \mathbf{N}(\delta_Y)$  do
4:      $X = \text{Label}(n)$ ;
5:      $e_n^z = \mathcal{Q}_\theta^1[Y]\mathcal{F}_\theta[n]$ ;
6:      $e_n^1 = (\mathcal{Q}_\theta^1[Y] - \mathcal{Q}_\theta^0[Y])\mathcal{F}_{Z,\theta}[n]\mathcal{B}_{Z,\theta}^1[Ch^1(n)]\theta_{[X]}\theta_{Del_Y^1(x)}$ ;
7:      $e_n^0 = (\mathcal{Q}_\theta^1[Y] - \mathcal{Q}_\theta^0[Y])\mathcal{F}_{Z,\theta}[n]\mathcal{B}_{Z,\theta}^1[Ch^0(n)]\theta_{[\bar{X}]}\theta_{Del_Y^0(x)}$ ;
8:     if  $X \in \mathbf{U}$  then
9:        $\mathcal{Q}_\theta^1[X] += e_n^z + e_n^1$ ,  $\mathcal{Q}_\theta^0[X] += e_n^z + e_n^0$ ;
10:    else if  $X \in \mathbf{X}$  then
11:       $\eta_\theta^1[[X]] += (e_n^z + e_n^1)\theta_{[X]}$ ,  $\eta_\theta^0[[X]] += (e_n^z + e_n^0)\theta_{[\bar{X}]}$ ;
12:    end if
13:     $X' : \text{Ord}_Y(X') = \text{Ord}_Y(X) + 1$ 
14:     $\zeta_Z[X'] += e_n^z\theta_{[X]} + e_n^z\theta_{[\bar{X}]}$ ,  $\zeta[X'] += e_n^1\theta_{[X]} + e_n^0\theta_{[\bar{X}]}$ ;
15:     $\zeta_Z[\text{Label}(Ch^1(n))] -= e_n^z\theta_{[X]}$ ,  $\zeta[\text{Label}(Ch^1(n))] -= e_n^1\theta_{[X]}$ ;
16:     $\zeta_Z[\text{Label}(Ch^0(n))] -= e_n^z\theta_{[\bar{X}]}$ ,  $\zeta[\text{Label}(Ch^0(n))] -= e_n^0\theta_{[\bar{X}]}$ ;
17:  end for
18:   $\mathcal{V} = \mathbf{V}(\delta_Y)$ ;
19:   $X = \text{argmin}_{X' \in \mathcal{V}} \text{Ord}_Y(X')$ ;
20:   $T_Z = \zeta_Z[X]$ ,  $T = \zeta[X]$ ;
21:   $\mathcal{V} = \mathcal{V} \setminus \{X\}$ ;
22:  while  $\mathcal{V} \neq \emptyset$  do
23:     $X = \text{argmin}_{X' \in \mathcal{V}} \text{Ord}_Y(X')$ ;
24:    if  $X \in \mathbf{U}$  then
25:       $\mathcal{Q}_\theta^1[X] += T_Z$ ,  $\mathcal{Q}_\theta^0[X] += T_Z + T/\theta_{[X]}$ ;
26:    else if  $X \in \mathbf{X}$  then
27:       $\eta_\theta^1[[X]] += T_Z\theta_{[X]}$ ,  $\eta_\theta^0[[X]] += T_Z\theta_{[\bar{X}]} + T$ ;
28:    end if
29:     $T_Z += \zeta_Z[X]$ ,  $T += \zeta[X]$ ;
30:     $\mathcal{V} = \mathcal{V} \setminus \{X\}$ ;
31:  end while
32: end

```

## 4 Time complexities for specific models

The time complexity of building BDDs is NP-hard in general [6]. However, there are efficient techniques to build BDDs using the *Apply operation* [2] and those to find good variable orderings, be they *dynamic* or *static* [6, 13]. So building BDDs can be done efficiently in practice. In this

section, we evaluate the time complexity of both building (Z)BDDs and running the (Z)BDD-EM algorithm for two models, noisy-OR models and probabilistic context free grammars. We show that their time complexities are equal to the standard ones.

## 4.1 Noisy-OR models

A noisy-OR model [17] represents a relation between multiple causes and an effect. Let  $F$  be an observable variable representing an effect, and  $C_1, C_2$  and  $C_3$  basic variables representing possible causes which make  $F$  true. While the logical OR relation is represented as  $F \Leftrightarrow C_1 \vee C_2 \vee C_3$ , the noisy-OR relation allows for a situation where  $C_1$  is true but  $F$  is false. For this noisy-OR model, we introduce *inhibition variables*,  $I_1, I_2$  and  $I_3$ , which inhibit  $F$  to be true with probabilities  $\theta_{[I_1]} = P(F=0 \mid C_1=1, C_2=0, C_3=0)$ ,  $\theta_{[I_2]} = P(F=0 \mid C_1=0, C_2=1, C_3=0)$  and  $\theta_{[I_3]} = P(F=0 \mid C_1=0, C_2=0, C_3=1)$ , respectively. An  $N$ -input noisy-OR model between  $F$  and  $C_1, C_2, \dots, C_N$  is described by:

$$F = (C_1 \wedge \bar{I}_1) \vee (C_2 \wedge \bar{I}_2) \vee \dots \vee (C_N \wedge \bar{I}_N).$$

Fig. 5 shows a BDD representing  $F$  under the variable ordering  $Ord_F$  such that  $C_i \prec C_j, I_i \prec I_j$  ( $i < j$ ) and  $C_i \prec I_k$  ( $i \leq k$ ). We construct a BDD from  $F$  using the Apply operation [2], denoted by  $\text{Apply}(\delta_X, \delta_Y, \langle \text{op} \rangle)$ , that builds a BDD representing  $X \langle \text{op} \rangle Y$  where  $\delta_X$  and  $\delta_Y$  represent the boolean functions  $X$  and  $Y$ , respectively. Although the time complexity of  $\text{Apply}(\delta_X, \delta_Y, \langle \text{op} \rangle)$  is  $O(N_X N_Y)$  in general, where  $N_X = |\mathbf{N}(\delta_X)|$  and  $N_Y = |\mathbf{N}(\delta_Y)|$ , we can see an application of  $\text{Apply}(\cdot)$  for an  $N$ -input noisy-OR model takes just  $O(1)$ . So the BDD is obtained by applying the Apply operation  $N$  times, and the time complexity becomes  $O(N)$  under  $Ord_F$ . Also the time complexity of the E-step is  $O(N)$  because  $\Delta_F = 1$ ,  $\mathbf{N}(F) = 2N$  and  $\mathbf{V}(F) = 2N$ .

## 4.2 Probabilistic context free grammars

A CFG is a quadruplet  $\langle V_N, V_T, R, S \rangle$  where  $V_N$  a set of non-terminal symbols,  $V_T$  a set of terminal symbols,  $R$  a set of derivation rules and  $S$  the initial symbol respectively. A PCFG is a CFG in which derivation rules are chosen probabilistically. Each  $r \in R$  has a parameter, i.e. the probability of  $r$  being chosen. Suppose we attempt to estimate the parameters of a PCFG from a sentence represented as a sequence of  $L$  words  $w(1), w(2), \dots, w(L)$  ( $w(i) \in V_T$ ). We assume the PCFG has  $V_N = \{X_1, X_2, \dots, X_N\}$ ,  $V_T = \{a_1, a_2, \dots, a_M\}$ ,  $R = \{r_{ijk}, r_{il} \mid 1 \leq i, j, k \leq N, 1 \leq l \leq M\}$  and  $S = X_1$ , where  $r_{ijk}$  and  $r_{il}$  represents derivation rules  $X_i \rightarrow X_j X_k$  and  $X_i \rightarrow a_l$  respectively. We introduce  $D_{ijk}$  and  $D_{jl}$  as basic boolean variables and  $A_i(d, d')$  as an intermediate boolean variable.<sup>9</sup> The assignment  $D_{ijk} = 1$  (resp.  $D_{il} = 1$ ) means the rule  $r_{ijk}$  (resp.  $r_{il}$ ) is applied. Similarly the assignment of  $A_i(d, d') = 1$  says that a non-terminal symbol  $X_i$  spans from  $w(d+1)$  to  $w(d')$ .  $A_i(d, d')$  is defined as:

$$A_i(d, d+1) = D_{il} \text{ such that } w(d+1) = a_l$$

$$A_i(d, d') = \bigvee_{1 \leq i, j \leq N} \left( \bigvee_{d \leq d'' \leq d'} (D_{ijk} \wedge A_j(d, d'') \wedge A_k(d'', d')) \right),$$

where  $D_{ijk}$  and  $D_{ij'k'}$  ( $j \neq j'$  or  $k \neq k'$ ), and also  $A_i(j, k)$  and  $A_{i'}(j', k')$  ( $i \neq i'$ ,  $j \neq j'$  or  $k \neq k'$ ) are *exclusive*. In such a case, we do not add probabilities to the assignments that violate the exclusiveness in computing conditional expectations. The observable variable  $F$  is given as  $F = A_1(1, L)$ .

<sup>9</sup>Strictly speaking, we must consider underlying derivation sequences in parsing. That is, we should classify  $A_i(d, d')$  into  $A_i^{t,k}(d, d')$ s where  $t$  is some derivation sequence and  $k$  is the index of a rewriting in  $t$  so that  $A_i^{t,k}(d, d')$  and  $A_j^{t,k}(d, d')$  ( $i \neq j$ ) are exclusive. Fortunately we can entirely omit superscripts,  $t$  and  $k$ , in describing the computation of expected occurrences of basic variables in derivation sequences.

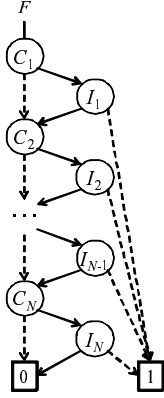


Figure 5: A BDD representing the noisy-OR model.

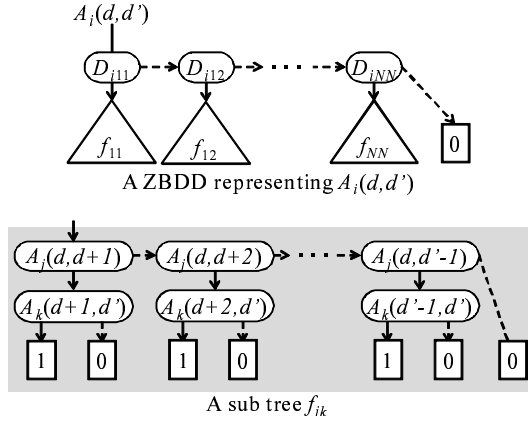


Figure 6: A ZBDD representing  $A_i(d, d')$ .

Fig. 6 shows a ZBDD fragment  $\delta_{A_i(d, d')}$ .<sup>10</sup> In this figure, we adopt the variable ordering  $Ord_{A_i(d, d')}$  such that

$$\begin{aligned} D_{ijk} < A_{j'}(d, l) < A_{k'}(l', d') & \quad \text{for any } j, k, l, j', k' \text{ and } l' \\ D_{ijk} < D_{ij'k'} & \quad j < j' \text{ or } (j = j') \wedge (k < k') \end{aligned}$$

$$\begin{aligned} A_j(d, l) < A_{j'}(d, l') & \quad j < j' \text{ or } (j = j') \wedge (l < l') \\ A_j(l, d') < A_{j'}(l', d') & \quad j < j' \text{ or } (j = j') \wedge (l < l'). \end{aligned}$$

We can build  $\delta_{A_i(d, d')}$  from the boolean function  $A_i(d, d')$  using the *Change operation* and the *Union operation* [12]. Under  $Ord_{A_i(d, d')}$ , the time complexity of building  $\delta_{A_i(d, d')}$  is  $O(N^2L)$ , where  $N$  is the number of non-terminal symbols and  $L$  is the length of the observed sentence. The total time complexity of building  $\delta_{A_i(d, d')} \in \Delta_F$  is  $O(N^3L^3)$  because  $|\Delta_F|$  is linear in  $NL^2$ .  $\delta_F$  and  $\delta_{A_i(d, d+1)}$  contain only one node in each and we can ignore them. In Fig. 6, we see  $|\mathbf{V}(A_i(d, d'))|$  and  $|\mathbf{N}(\delta_{A_i(d, d')})|$  are  $O(N^2L)$ , and  $|\Delta_F|$  is  $O(NL^2)$ . So the time complexity of running ZBDD-EM() is  $O(N^3L^3)$ . Consequently, the time complexity of building  $F$  and the E-step is  $O(N^3L^3)$ , respectively, which equals the standard one.

## 5 Learning experiment

To confirm that the BDD-EM algorithm with GETOUTSIDEEXPE\*() works correctly, we conducted a simple learning experiment that estimates the parameters of a 4-input noisy-OR model from artificial data sampled under a random parameter setting. Although we have been concentrating on the single observation case up to now, the implemented BDD-EM algorithm can deal with i.i.d. observations of the values of  $F$ . So in the experiment, we used 100 i.i.d. samples of the values of  $F$  in which  $F$  is true 60 times and false 40 times.

Table. 1 shows the learning result, where the first columns shows the number of EM iterations. The next eight columns show values of parameters  $\theta_{[C_1]}, \dots, \theta_{[\bar{I}_4]}$ . The last column is the log likelihood. We see that the log likelihood increases as iteration goes on.

<sup>10</sup>One may think that we can introduce new ZBDD fragments, each representing a subtree  $f_{jk}$ . This is not allowed, however, because a variable  $A_j(d, d')$  appearing in  $f_{jk}$  and  $A_j(d, d')$  (the same variable label) appearing in  $f_{j'k'}$  are not independent. If we separated  $f_{jk}$  and  $f_{j'k'}$  as new BDD fragments, they would be independent.

Table 1: A result of learning parameters of the 4-input noisy-OR model.

Step	$\theta_{[C_1]}$	$\theta_{[\bar{I}_1]}$	$\theta_{[C_2]}$	$\theta_{[\bar{I}_2]}$	$\theta_{[C_3]}$	$\theta_{[\bar{I}_3]}$	$\theta_{[C_4]}$	$\theta_{[\bar{I}_4]}$	$L$
0	0.42403	0.46887	0.27365	0.41808	0.45813	0.45969	0.17264	0.94743	-82.746725
5	0.35718	0.40723	0.22933	0.38257	0.39060	0.39235	0.10243	0.94297	-67.312570
10	0.35529	0.40548	0.22804	0.38154	0.38869	0.39045	0.10077	0.94286	-67.301176
15	0.35523	0.40543	0.22800	0.38151	0.38863	0.39039	0.10072	0.94286	-67.301167
20	0.35523	0.40542	0.22800	0.38151	0.38863	0.39039	0.10072	0.94286	-67.301167

## 6 Related work

Our work is considered as a succession to the previous work done by Minato et al. [14]. It shows how to compile BNs into ZBDDs to compute probabilities but probability learning is left untouched. In this paper we supplemented a necessary algorithm to apply ZBDDs to EM learning. The introduction of (Z)BDDs solves a long-standing problem of PRISM [18], a logic-based modeling language for generative modeling. It employs data structure called *explanation graphs* similar to decomposed BDDs to represent boolean formulas in disjunctive normal form. The current PRISM however assumes the *exclusiveness condition* that the disjuncts are exclusive to make sum-product probability computation possible. Since the proposed algorithms are applicable to explanation graphs as well, it allows PRISM to abolish the exclusiveness condition. ProbLog is a recent logic-based formalism that computes probabilities via BDDs [4]. A ProbLog program computes the probability of a query atom from a disjunction of conjunctions made up of independent probabilistic atoms by converting the disjunction to a BDD and applying the sum-product computation to it. Since our (Z)BDD-EM algorithm works on (Z)BDDs, integrating it with ProbLog for probability learning seems an interesting future research topic.

In a broader context, the (Z)BDD-EM algorithm is an example of *propositionalized probability computation* (PPC), a recent trend that uses propositional formulas to compute and learn probabilities in a probabilistic model. Examples include BNs [14, 3, 9], Markov random fields [10, 16], and probabilistic logic programming [18].

## 7 Conclusions

We have proposed the (Z)BDD-EM algorithm that runs on decomposed (Z)BDDs made out of probabilistic propositional variables. Inside and outside probability computation in PCFGs is adaptively propositionalized for decomposed (Z)BDDs and EM learning is performed by iterating propositionalized inside and outside probability computation. The (Z)BDD-EM algorithm is generic in the sense that it is applicable to any decomposed (Z)BDDs, and at the same time can be efficient thanks to dynamic programming though actual efficiency depends on the variable ordering. We also have proved that the noisy-OR model expressed by BDDs and PCFGs expressed by ZBDDs are computed in time linear to the number of causes in the former case and in time cubic to the sentence length in the latter case. In future work, we expect that it is not very difficult to extend the (Z)BDD-EM algorithm by combining it with variational Bayesian learning. We also hope to develop a propositionalized parameter learning algorithm for log-linear models following the (Z)BDD-EM algorithm.

## References

- [1] S. B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.
- [2] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
- [3] M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *Proc. of IJCAI'05*, pages 1306–1312, 2005.
- [4] L. De Raedt, K. Angelika, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proc. of IJCAI'07*, pages 2468–2473, 2007.
- [5] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society B*, 39:1–38, 1977.
- [6] R. Drechsler and D. Sieling. Binary decision diagrams in theory and practice. *Int'l J. on Software Tools for Technology Transfer*, 3:112–136, 2001.
- [7] J. Jain, A. Narayan, C. Coelho, S.P. Khatri, A. Sangiovanni-Vincentelli, R.K. Brayton, and M. Fujita. Decomposition techniques for efficient ROBDD construction. *Int'l Conf. on Formal Methods in Computer-Aided Design*, 1996.
- [8] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
- [9] R. Mateescu and R. Dechter. The relationship between AND/OR search spaces and variable elimination. In *Proc. of UAI'05*, pages 380–387, 2005.
- [10] D. McAllester, M. Collins, and F. Pereira. Case-factor diagrams for structured probabilistic modeling. In *Proc. of UAI'04*, pages 382–391, 2004.
- [11] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th Int'l Conf. on Design automation (DAC '93)*, pages 272–277, 1993.
- [12] S. Minato. Zero-suppressed BDDs and their applications. *Int'l J. on Software Tools for Technology Transfer*, 3(2):156–170, 2001.
- [13] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges. *Proc. ACM/IEEE Design Automation Conf.*, pages 52–57, 1990.
- [14] S. Minato, K. Satoh, and T. Sato. Compiling Bayesian networks by symbolic probability calculation based on Zero-suppressed BDDs. In *Proc. of IJCAI'07*, pages 2550–2555, 2007.
- [15] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.
- [16] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.
- [17] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003. 2nd edition.
- [18] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. of Artificial Intelligence Research*, 15:391–454, 2001.