Fast Message Passing Algorithm Using ZDD-Based Local Structure Compilation

Shan Gao GAOSHAN@IST.HOKUDAI.AC.JP
Gradate School of Information Science and Technology
Hokkaido University
Sapporo (Japan)

Masakazu Ishihata ISHIHATA.MASAKAZU@IST.HOKUDAI.AC.JP
Gradate School of Information Science and Technology
Hokkaido University
Sapporo (Japan)

Shin-ichi Minato MINATO@IST.HOKUDAI.AC.JP
Gradate School of Information Science and Technology
Hokkaido University
Sapporo (Japan)

Abstract

Compiling Bayesian Networks (BNs) into secondary structures to implement efficient exact inference is a hot topic in probabilistic modeling. One class of algorithms to compile BNs is to transform the BNs into junction tree structures utilizing the conditional dependency in the network. Performing message passing on the junction tree structure, we can calculate marginal probabilities for any variables in the network efficiently. However, the message passing algorithm does not consider the local structure in the network. Since the ability to exploit local structure to avoid redundant calculations has a significant impact on exact inference, in this article, we propose a fast message passing algorithm by exploiting local structure using Zero-suppressed Binary Decision Diagrams (ZDDs). We convert all the components used in message passing algorithm into Multi-linear Functions (MLFs), and then compile them into compact representation using ZDDs. We show that message passing on ZDDs can work more efficient than the conventional message passing algorithm on junction tree structures on some benchmark networks although it may be too memory consuming for some larger instances.

Keywords: Bayesian Network, local structure, ZDDs, message passing, exact inference.

1. Introduction

Developing efficient algorithms for inference in Bayesian Networks (BNs) has attracted much attention. Suppose all variables in a BN have domains over binary values. Compiling a BN into a secondary structure (Huang and Darwiche, 1996) then performing inference on the resulting structure (which is not necessarily exponential in the number of variables in the BN) has been proved as an efficient method in exact inference. One class of secondary structures is *junction tree* (Lauritzen and Spiegelhalter, 1988), also known as *jointree*, one of the most essential tree structure in exact inference of BNs, whose nodes and edges are labeled with subsets of variables in a BN. By utilizing the conditional dependencies in a BN and performing message passing algorithm (Nuel,

2012) on junction tree structures, the computation complexity of inference is bound to *treewidth*, the maximum size of a node in a junction tree.

Research in Chavira and Darwiche (2005) has shown that the ability of exploiting local structure, such as probability variables in the BN sharing same value, has a significant effect on the exact inference of Bayesian networks. Transforming the probability distributions in a BN into Multi-linear Functions (MLFs) and then compiling them using arithmetic circuits is a key idea for exploiting local structures. In this article, we consider to accelerate the essential inference algorithm, message passing algorithm by exploiting local structure with ZDDs (Minato, 1993). We transform message variables into MLFs to generate ZDDs. Then we show that using these MLFs, we can compute the same components in the message passing algorithm by dynamically changing values in message variables in MLFs.

We will show that through the compilation, we can calculate marginal probabilities for any given variable in a BN much more efficiency than the conventional message passing algorithm. Also, since our method depends on the size of separators in a junction tree, this suggests that, instead of directly using a junction tree to generate ZDDs, we can find good separators to decompose a BN into several subgraphs which form a tree and have the same properties as junction trees.

2. Preliminaries

2.1 Bayesian Networks

A *Bayesian Network* (BN)(Pearl, 2014) is a directed acyclic graph which defines a joint distribution over a set of random variables. A BN is defined as $B \triangleq \langle G, \Theta \rangle$, where $G \triangleq \langle V, E \rangle$ is a DAG that represents the dependencies of the variables and $\Theta \triangleq \{\theta_{i,j,k}\}_{i,j,k}$ is a set of parameters that defines conditional distributions. $V \triangleq \{X_1, \ldots, X_n\}$ is a set of random variables and $E \subset V \times V$ is a set of directed edges that represents dependency between variables. We use $x_{i,k}$ to denote the k-th value of X_i. $\Pi_i \triangleq \{X_i' \in V \mid (X_i', X_i) \in E\}$ denotes the parents of X_i and $\pi_{i,j}$ denotes the j-th instantiation of Π_i. We use $\theta_{i,j,k}$ to denote the probability that $X_i = x_{i,k}$ given $\Pi_i = \pi_{i,j}$: $\theta_{i,j,k} \triangleq P(X_i = x_{i,k} \mid \Pi_i = \pi_{i,j})$. Given B, a joint distribution over V is factorized as follows:

$$P(V) = \prod_{X_i \in V} P(X_i \mid \Pi_i). \tag{1}$$

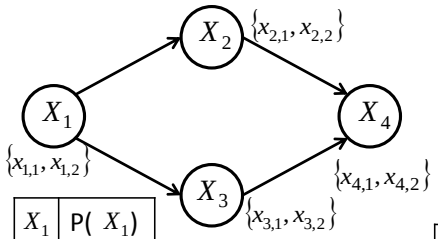
The marginal probability distribution $P(X_i)$ can be computed by marginalizing the other variables from the above joint distribution $P(V)$. We use the following expression to represent marginalization:

$$P(X_i) = \sum_{V \setminus \{X_i\}} P(V). \tag{2}$$

Fig. 1 shows a simple example of a BN over variables $\{X_1, X_2, X_3, X_4\}$ with domains $\{x_{1,1}, x_{1,2}\}$, $\{x_{2,1}, x_{2,2}\}$, $\{x_{3,1}, x_{3,2}\}$, and $\{x_{4,1}, x_{4,2}\}$, respectively. The joint probability of this BN is:

$$P(X_1, X_2, X_3, X_4) = P(X_1)\, P(X_2 \mid X_1)\, P(X_3 \mid X_1)\, P(X_4 \mid X_2, X_3)\,. \tag{3}$$

Obviously, computing a marginal $P(X_i)$ using Eq. (2) requires exponential time in the number of variables n. Thus an efficient algorithm for inference in BNs to convert the original BN into a junction tree structure and then calculate probabilities on the resulting junction tree is known as an essential approach.



X_1	$\mathsf{P}(X_1)$
$x_{1,1}$	$\theta_{1,1,1} = 0.4$
$x_{1,2}$	$\theta_{1,2,1} = 0.6$

$X_2 \mid X_1$	$\mathsf{P}(X_2 \mid X_1)$
$x_{2,1} \mid x_{1,1}$	$\theta_{2,1,1} = 0.8$
$x_{2,1} \mid x_{1,2}$	$\theta_{2,1,2} = 0.2$
$x_{2,2} \mid x_{1,1}$	$\theta_{2,2,1} = 1$
$x_{2,2} \mid x_{1,2}$	$\theta_{2,2,2} = 0$

$X_3 \mid X_1$	$\mathsf{P}(X_3 \mid X_1)$
$x_{3,1} \mid x_{1,1}$	$\theta_{3,1,1} = 0.5$
$x_{3,1} \mid x_{1,2}$	$\theta_{3,1,2} = 0.5$
$x_{3,2} \mid x_{1,1}$	$\theta_{3,2,1} = 0.5$
$x_{3,2} \mid x_{1,2}$	$\theta_{3,2,2} = 0.5$

$X_4 \mid X_2, X_3$	$\mathsf{P}(X_4 \mid X_2, X_3)$
$x_{4,1} \mid x_{2,1}, x_{3,1}$	$\theta_{4,1,1} = 0.0$
$x_{4,1} \mid x_{2,2}, x_{3,1}$	$\theta_{4,1,2} = 0.3$
$x_{4,1} \mid x_{2,1}, x_{3,2}$	$\theta_{4,1,3} = 0.5$
$x_{4,1} \mid x_{2,2}, x_{3,2}$	$\theta_{4,1,4} = 0.2$
$x_{4,2} \mid x_{2,1}, x_{3,1}$	$\theta_{4,2,1} = 0.3$
$x_{4,2} \mid x_{2,2}, x_{3,1}$	$\theta_{4,2,2} = 0.5$
$x_{4,2} \mid x_{2,1}, x_{3,2}$	$\theta_{4,2,3} = 0.2$
$x_{4,2} \mid x_{2,2}, x_{3,2}$	$\theta_{4,2,4} = 0.0$

Figure 1: An example of BN.

2.2 Junction tree and Message Passing

A *junction tree* $T = \langle N, A \rangle$ for BN B is a tree structure, where $N = \{1, 2, ..., m\}$ is a set of nodes and $A \subset N \times N$ is a set of undirected arcs [1] (a, b) where $a, b \in N$, $a \neq b$. Each node and each arc is labeled by a subset of variables V: $C_a \subseteq V$ denotes the label of node $a \in N$ that is called a *cluster*, and $S_{ab} \triangleq C_a \cap C_b$ denotes the label of arc $(a, b) \in A$ that is called a *separator*. We use $c_{a,m}$ to denote the m-th instantiation of C_a and $s_{ab.\ell}$ to denote the ℓ-th instantiation of S_{ab}. We here define a *family* $F_a \triangleq \{X_i \in V \mid X_i \cup \Pi_i \subseteq C_a\}$. Then, if T is a junction tree, clusters in T satisfies the following properties:

- *Running intersection*: $\forall a, b \in N, \forall c \in N$ on the unique path between a and b, $C_a \cap C_b \subseteq C_c$.
- *Family preserving*: $\forall X_i \in V, \exists a \in N, X_i \in F_a$.

In general, X_i can belong to multiple families. For each X_i, here we choose one family F_a s.t. $X_i \in F_a$ in arbitrary manner and say that X_i is *assigned* to C_a. We use $SN(C_a)$ to denote the set of all variables that are assigned to C_a. In this paper, without loss of generality, we assume that T is rooted and use r to denote the root node of T. We use $d(a)$ to denote the depth of a that is the distance from r, and use $N_d \triangleq \{a \in N \mid d(a) = d\}$ and $d(T) \triangleq \max_{a \in N} d(a)$. We use $par(a) \in N$ and $child(a) \subset N$ to denote the parent and the set of children of a in T, respectively.

Message passing is an algorithm that computes all marginal probabilities $P(X_i)$ $(X_i \in V)$ on a BN B using junction tree T (Lauritzen and Spiegelhalter, 1988). In message passing, a *potential*, which is a function over a set of variables, is recursively updated by *messages* that are computed from the other potentials. We define the potential over C_a and S_{ab} as $\Phi_{C_a}(C_a)$ and $\Phi_{S_{ab}}(S_{ab})$, respectively. We use $\phi_{c_{a,m}} \triangleq \Phi_{C_a}(c_{a,m})$ and $\phi_{s_{ab,\ell}} \triangleq \Phi_{S_{ab,\ell}}(s_{ab,\ell})$, respectively. We use Φ to denote the set of all potentials $\Phi_{C_a}(C_a)$ and $\Phi_{S_{ab}}(S_{ab})$. For the sake of simplicity, we omit the arguments of potentials afterward. The message passing is summed up in Algorithm 1.

1. To distinguish with the vertexes and edges in $B = \langle G, \Theta \rangle$, we use nodes and arcs in a junction tree.

Algorithm 1 MessagePassing(T)

// Initialize
$\phi_{c_{a,m}} \leftarrow \prod_{X_i \in SN(C_a)} P(x_{i,k} \mid \pi_{i,j})$ for all $a \in N$ and m, where $x_{i,k} \cup \pi_{i,j} \subseteq c_{a,m}$
$\phi_{s_{ab,\ell}} \leftarrow 1$ for all $(a,b) \in A$ and ℓ
// Collect messages : child to parent
for $d = d(T) - 1$ to 0 **do**
  for all $b \in N_d$ **do**
    for all $a \in child(b)$ **do**
      Pass$(a, b, \Phi_{C_a}, \Phi_{C_b}, \Phi_{S_{ab}})$
    end for
  end for
end for
// Distribute messages : parent to child
for $d = 1$ to $d(T)$ **do**
  for all $b \in N_d$ **do**
    for all $a \in par(b)$ **do**
      Pass$(a, b, \Phi_{C_a}, \Phi_{C_b}, \Phi_{S_{ab}})$
    end for
  end for
end for
return Φ

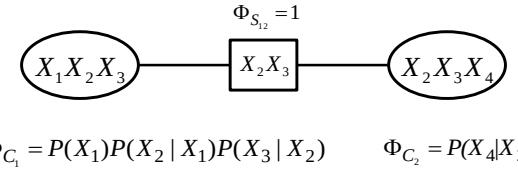


Figure 2: A junction tree for the BN in Fig 1.

When the message passing ends, these potentials satisfy the following equations:

$$\Phi_{C_a} = P(C_a)\,, \qquad \Phi_{S_{ab}} = P(S_{ab})\,, \qquad P(V) = \frac{\prod_{a \in N} \Phi_{C_a}}{\prod_{(a,b) \in A} \Phi_{S_{a,b}}}. \tag{4}$$

Algorithm 2 Pass$(a, b, \Phi_{C_a}, \Phi_{C_b}, \Phi_{S_{ab}})$

$\phi^{\text{old}}_{s_{ab,\ell}} \leftarrow \phi_{s_{ab,\ell}}$ for all ℓ
$\phi_{s_{ab,\ell}} \leftarrow \sum_{c_{a,m} \setminus s_{ab,\ell}} \phi_{c_{a,m}}$, for all ℓ and m where $s_{ab,l} \subset c_{a,m}$
$\phi_{c_{b,m}} \leftarrow \phi_{c_{b,m}} \frac{\phi_{s_{ab,\ell}}}{\phi^{old}_{s_{ab,\ell}}}$ for all ℓ and m where $s_{ab,l} \subset c_{b,m}$

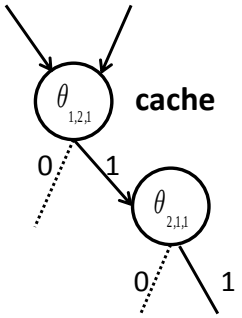


Figure 3: An example of local computation.

Once we compute $P(C_a)$ by the message passing, we can easily compute $P(X_i)$ ($X_i \in C_a$) by marginalizing out all $X_j \in C_a \backslash \{X_i\}$ from $P(C_a)$ as follows:

$$P(X_i) = \sum_{C_a \backslash \{X_i\}} P(C_a). \tag{5}$$

2.3 Local Structures

Though message passing on junction trees is an efficient method, for each cluster in a junction tree, we need to prepare a table for the potential whose size is exponential to the size of the cluster. Operations on these tables incur in considerable costs in message passing. If we can exploit all the local structures, in the form of equal parameters and local computation, during message passing, the algorithm can be significantly accelerated.

For the example CPTs in Fig 1, there are 18 parameters in total in the CPTs and yet only 8 of them are distinct. Compact representations for these CPTs are expected. Also while calculating different probabilities who are sharing the same local computation, for example calculating $P(x_{1,2}, x_{2,1}, x_{3,1}) = \theta_{1,2,1}\theta_{2,1,1}\theta_{3,1,1}$ and $P(x_{1,2}, x_{2,1}, x_{3,2}) = \theta_{1,2,1}\theta_{2,1,1}\theta_{3,2,1}$, multiplication for $\theta_{1,2,1}\theta_{2,1,1}$ are repetitive. If we can cache local computations in advance, there is no need to calculate it again while evaluating a new query which shares the same local computations with queries already calculated.

Compiling a Bayesian network with ZDDs has been proposed by Minato et al. (2007). In their method, they first convert the probability distribution of a BN into MLFs. Then they treat variables in MLFs as combination itemsets so that they can compile all the MLFs into ZDDs using inter-ZDD operations. They showed that symbolic probability calculation for inference based on arithmetic operations can be executed in a time almost linear with the size of ZDDs.

Thus we consider to introduce ZDD techniques into message passing algorithm. Node-sharing in ZDDs helps in representing CPTs compactly and cache memories in ZDDs help to avoid repetitive computations of local structures. The example of a ZDD for local structure is shown in Fig 3.

3. Proposed Method

In this section, we introduce our fast message passing algorithm using ZDDs. We first convert all updates of potentials of naive message passing into MLFs. Then, we compile them into ZDDs in

the same manner as the conventional ZDD-based method (Minato et al., 2007). In Section 3.1, we define MLFs that we use in our method, then in Section 3.2 we explain how we can simulate the message passing using those MLFs.

3.1 MLFs for Message Passing

Before generating the ZDD, we need to convert potentials in junction tree into MLFs. While converting these components into MLFs, we introduce three types of variables: indicator variable $\lambda_{i,k}$; parameter variable $\theta_{i,j,k}$, with the same definition in Section 2; and *message variables* $\beta^a_{s_{ab,\ell}}$, $\beta^b_{s_{ab,\ell}}$ for instantiation $s_{ab,\ell}$.

In our method, we need to generate MLFs for $P(X_i \mid \Pi_i)$, $\Phi_{S_{ab}}$ and Φ_{C_a}, for each $X_i \in V, (a,b) \in A, a \in N$. We use MLF_{X_i}, $\mathrm{MLF}_{S_{ab}}$, MLF_{C_a} to represent MLFs for them respectively. We define the corresponding MLFs for these components as follows:

- For all $X_i \in V$, we define

$$\mathrm{MLF}_{X_i} \triangleq \sum_{j,k} \lambda_{i,k}\Lambda_{i,j}\theta_{i,j,k}, \qquad \Lambda_{i,j} \triangleq \prod_{x_{i',k'} \in \pi_{i,j}} \lambda_{i',k'}. \tag{6}$$

- For all $(a,b) \in A$, define

$$\mathrm{MLF}^a_{S_{ab}} \triangleq \sum_{l} \Lambda_{ab,\ell}\beta^a_{s_{ab,\ell}}, \quad \mathrm{MLF}^b_{S_{ab}} \triangleq \sum_{l} \Lambda_{ab,\ell}\beta^b_{s_{ab,\ell}}, \quad \Lambda_{ab,\ell} \triangleq \prod_{x_{i,k} \in s_{ab,\ell}} \lambda_{i,k}. \tag{7}$$

- For all $a \in N$, using the above MLF_{X_i} and $\mathrm{MLF}^a_{S_{ab}}$, we define

$$\mathrm{MLF}_{C_a} \triangleq \prod_{X_i \in SN(C_a)} \mathrm{MLF}_{X_i} \prod_{(a,b)\in A} \mathrm{MLF}^a_{S_{ab}}. \tag{8}$$

Using the multi-valued multiplication algorithm (Minato et al., 2007) in ZDD operations, the product of these MLFs produces all possible combinations of their terms. Variables such as $\lambda_{i,k}$ and $\lambda_{i,k+1}$ (variables representing different instances of the same variable) does not coexist in the same term as they are mutually exclusive. Also, union operations of ZDDs make sure that no term can contain the same variable more than once, so instead of $\lambda^2_{1,1}$ for duplicate variables, simply $\lambda_{1,1}$ will appear in the result.

For example the junction tree in Fig 2, first we generate the MLF for every CPT as follows:

$$\mathrm{MLF}_{X_1} = \lambda_{1,1}\theta_{1,1,1} + \lambda_{1,2}\theta_{1,1,2}. \tag{9}$$
$$\mathrm{MLF}_{X_2} = \lambda_{1,1}\lambda_{2,1}\theta_{2,1,1} + \lambda_{1,1}\lambda_{2,2}\theta_{2,1,2} + \lambda_{1,2}\lambda_{2,1}\theta_{2,2,1} + \lambda_{1,2}\lambda_{2,2}\theta_{2,2,2}. \tag{10}$$
$$\mathrm{MLF}_{X_3} = \lambda_{1,1}\lambda_{3,1}\theta_{3,1,1} + \lambda_{1,1}\lambda_{3,2}\theta_{3,1,2} + \lambda_{1,2}\lambda_{3,1}\theta_{3,2,1} + \lambda_{1,2}\lambda_{3,2}\theta_{3,2,2}. \tag{11}$$
$$\mathrm{MLF}_{X_4} = \lambda_{2,1}\lambda_{3,1}\lambda_{4,1}\theta_{4,1,1} + \lambda_{2,1}\lambda_{3,1}\lambda_{4,2}\theta_{4,2,1} + ... + \lambda_{2,2}\lambda_{3,2}\lambda_{4,2}\theta_{4,4,2}. \tag{12}$$

Then we have $\mathrm{MLF}^1_{S_{12}}$ and $\mathrm{MLF}^2_{S_{12}}$ for C_1 and C_2 respectively:

$$\mathrm{MLF}^1_{S_{12}} = \lambda_{2,1}\lambda_{3,1}\beta^1_{x_{21},x_{31}} + \lambda_{2,1}\lambda_{3,2}\beta^1_{x_{21},x_{32}} + \lambda_{2,2}\lambda_{3,1}\beta^1_{x_{22},x_{31}} + \lambda_{2,2}\lambda_{3,2}\beta^1_{x_{22},x_{32}}. \tag{13}$$
$$\mathrm{MLF}^2_{S_{12}} = \lambda_{2,1}\lambda_{3,1}\beta^2_{x_{21},x_{31}} + \lambda_{2,1}\lambda_{3,2}\beta^2_{x_{21},x_{32}} + \lambda_{2,2}\lambda_{3,1}\beta^2_{x_{22},x_{31}} + \lambda_{2,2}\lambda_{3,2}\beta^2_{x_{22},x_{32}}. \tag{14}$$

By multiplying these MLFs, we get MLFs for Φ_{C_1} and Φ_{C_2} respectively as:

$$\begin{aligned}
\mathrm{MLF}_{C_1} &= \mathrm{MLF}_{X_1}\mathrm{MLF}_{X_2}\mathrm{MLF}_{X_3}\mathrm{MLF}^1_{S_{12}} && (15)\\
&= \lambda_{1,1}\lambda_{2,1}\lambda_{3,1}\theta_{1,1,1}\theta_{2,1,1}\theta_{3,1,1}\beta^1_{x_{21},x_{31}} + \lambda_{1,1}\lambda_{2,1}\lambda_{3,2}\theta_{1,1,1}\theta_{2,1,1}\theta_{3,1,2}\beta^1_{x_{21},x_{32}} && (16)\\
&+ \lambda_{1,1}\lambda_{2,2}\lambda_{3,1}\theta_{1,1,1}\theta_{2,1,2}\theta_{3,2,1}\beta^1_{x_{22},x_{31}} + \lambda_{1,1}\lambda_{2,2}\lambda_{3,2}\theta_{1,1,1}\theta_{2,1,2}\theta_{3,2,2}\beta^1_{x_{22},x_{32}} && (17)\\
&... && (18)\\
&+ \lambda_{1,2}\lambda_{2,2}\lambda_{3,1}\theta_{1,1,2}\theta_{2,2,2}\theta_{3,4,1}\beta^1_{x_{22},x_{31}} + \lambda_{1,2}\lambda_{2,2}\lambda_{3,2}\theta_{1,1,2}\theta_{2,2,2}\theta_{3,4,2}\beta^1_{x_{22},x_{32}}. && (19)\\
& && (20)\\
\mathrm{MLF}_{C_2} &= \mathrm{MLF}_{X_4}\mathrm{MLF}^2_{S_{21}} && (21)\\
&= \lambda_{2,1}\lambda_{3,1}\lambda_{4,1}\theta_{4,1,1}\beta^2_{x_{21},x_{31}} + \lambda_{2,1}\lambda_{3,1}\lambda_{4,2}\theta_{4,1,2}\beta^2_{x_{21},x_{31}} && (22)\\
&... && (23)\\
&+ \lambda_{2,2}\lambda_{3,2}\lambda_{4,1}\theta_{4,4,1}\beta^2_{x_{22},x_{32}} + \lambda_{2,2}\lambda_{3,2}\lambda_{4,2}\theta_{4,4,2}\beta^2_{x_{22},x_{32}}. && (24)
\end{aligned}$$

3.2 Message Passing with MLFs

If we can manipulate MLFs to compute the same components as in the message passing algorithm, we can accelerate the algorithm by compiling MLFs into ZDDs and utilizing ZDD techniques.

For the three types of variables $\lambda_{i,k}$, $\theta_{i,j,k}$, and $\beta^a_{s_{ab,\ell}}$ in MLFs as defined above, $\lambda_{i,k} \in \{0,1\}$, $\theta_{i,j,k}, \beta^a_{s_{ab,\ell}} \in [0,1]$. During our method, parameter variables $\theta_{i,j,k}$ keep the values consistent to $P(x_{i,k} \mid \pi_{i,j})$ all the time. Values in message variables $\beta^a_{s_{ab,\ell}}$ dynamically change according to the messages passed between nodes $a, b \in N$. For any set of variables $W \subseteq V$, we define MLF_W representing probability distribution over W in the following sense. For any instantiation w of W, we can evaluate MLF_W so it returns the probability over w denoted by $\alpha_W(w)$.

Definition 1 *The value of* MLF_W *at instantiation* w*, denoted by* $\alpha_W(w)$*, is the result of replacing each indicator variable* $\lambda_{i,k}$ *in* MLF_W *with 1 if* $\lambda_{i,k}$ *is consistent with* w*, and with* 0 *otherwise.*

While using MLFs, messages over variables S_{ab} passed from a to b are obtained by calculating $\alpha_{C_a}(s_{ab,\ell})$ for all ℓ. The message passing algorithm with MLFs is summed up in Algorithm 3. It works in accordance with the message passing algorithm introduced in Section 2 except operations on potentials are transformed to operations on MLFs. We evaluate and change the values in message variables of MLFs to implement the same computation on potentials.

In the initialization step (line 2 in Algorithm 3), we initialize all message variables to 1. With this assignment, all MLFs satisfy the following equations which are consistent with the initialization in Algorithm 1 (line1-7):

$$\mathrm{MLF}_{C_a}(\beta^a_{s_{ab,\ell}} = 1) = \prod_{X_i \in SN(C_a)} P(X_i \mid \Pi_i), \text{ for all } a \in N, \tag{25}$$

$$\mathrm{MLF}^a_{S_{ab}}(\beta^a_{s_{ab,\ell}} = 1) = \sum_l \Lambda_{ab,\ell},\ \mathrm{MLF}^b_{S_{ab}}(\beta^b_{s_{ab,\ell}} = 1) = \sum_l \Lambda_{ab,\ell}, \text{ for all } (a,b) \in A. \tag{26}$$

Compared to the conventional algorithm, a message passing from node a to b in the collecting and distributing operations is performed differently. **In the message collecting operation, a message passing from** a **to** b done as follows:

Algorithm 3 MessagePassingwithMLF(T)

1: // Initialize
2: $\beta^a_{s_{ab,\ell}} \leftarrow 1, \beta^b_{s_{ab,\ell}} \leftarrow 1$, For all $(a,b) \in A$, and ℓ
3: // Collect messages : child to parent
4: **for** $d = d(T) - 1$ to 0 **do**
5: **for all** $b \in N_d$ **do**
6: **for all** $a \in child(b)$ **do**
7: Collect$(a, b, \mathrm{MLF}_{C_a}, \mathrm{MLF}_{C_b})$
8: **end for**
9: **end for**
10: **end for**
11: // Distribute messages : parent to child
12: **for** $d = 1$ to $d(T)$ **do**
13: **for all** $b \in N_d$ **do**
14: **for all** $a \in par(b)$ **do**
15: Distribute$(a, b, \mathrm{MLF}_{C_a}, \mathrm{MLF}_{C_b})$
16: **end for**
17: **end for**
18: **end for**
19: **return** evaluate all MLFs and return the results

- evaluate MLF_{C_a} on all instantiations of S_{ab} to get the messages $\alpha_{C_a}(s_{ab,\ell})$ for all ℓ, and preserve them in corresponding message variables $\beta^a_{s_{ab,\ell}}$ in MLF_{C_a} (refer to line 1 in Algorithm 4)
- Node b absorbs messages $\alpha_{C_a}(s_{ab,\ell})$ by updating corresponding messages variables $\beta^b_{s_{ab,\ell}}$ for all ℓ in MLF_{C_b}. (refer to line 3 in Algorithm 4)

Algorithm 4 Collect$(a, b, \mathrm{MLF}_{C_a}, \mathrm{MLF}_{C_b})$

1: $\beta^a_{s_{ab,\ell}} \leftarrow \alpha_{C_a}(s_{ab,\ell})$, for all ℓ // equivalent to $\phi_{s_{ab,\ell}} \leftarrow \sum_{c_{a,m} \setminus s_{ab,\ell}} \phi_{c_{a,m}}$ for all ℓ
2: $\beta^b_{s_{ab,\ell}} \leftarrow \frac{\alpha_{C_a}(s_{ab,\ell})}{\beta^b_{s_{ab,\ell}}}$, for all ℓ

In the message distributing operation, a message passing from a to b is evaluated as follows:

- evaluate MLF_{C_a} on all instantiations of S_{ab} to get messages $\alpha_{C_a}(s_{ab,\ell})$, and node b absorbs messages by updating message variables (refer to line 1 in Algorithm 5).

Algorithm 5 Distribute$(a, b, \mathrm{MLF}_{C_a}, \mathrm{MLF}_{C_b})$

1: $\beta^b_{s_{ab,\ell}} \leftarrow \frac{\alpha_{C_a}(s_{ab,\ell})}{\beta^b_{s_{ab,\ell}}}$, for all $(a,b) \in A$ and ℓ

Note that in the distributing operation, since a has absorbed messages in the collecting operation, there is no need to change the message variables in MLF_{C_a}, thus we only need to pass messages out to MLF_{C_b}.

After a full round of message passing, the MLFs for any Φ_{C_a} represents the joint distributions over variables in C_a:

$$\mathrm{MLF}_{C_a} = P(C_a)\,. \tag{27}$$

Also, if messages over S_{ab} are passed outward from a and collected inward to a, then we have:

$$\mathrm{MLF}^{a}_{S_{ab}} = P(S_{ab})\,. \tag{28}$$

Similarly, we can calculate the probability of any variable $X_i \in V$ by evaluating the MLF_{C_a} where $X_i \in C_a$. For the example in Fig 2, after performing the message passing algorithm on MLFs generated according to the procedure explained above, we can get the MLFs satisfying:

$$\mathrm{MLF}_{C_1} = P(X_1, X_2, X_3)\,, \mathrm{MLF}_{C_2} = P(X_2, X_3, X_4)\,, \tag{29}$$

$$\mathrm{MLF}^{1}_{S_{12}} = P(X_2, X_3)\,. \tag{30}$$

4. Experiment and Results

We implement our method on an Intel Core Quad CPU Q9550@2.83GHz * 4 PC with Ubuntu 12.04LTS and 3.8GiB of main memory. We can manipulate up to 40,000,000 nodes of ZDDs using the ZDD package implemented by Minato (Minato, 2001). We used dataset of BN Benchmark (HebrewUniversity) ALARM, HAILFINDER, INSURANCE, etc. In our experiment. BN specifications with number of nodes and parameters are shown in Table 1. The experiment results of our method comparing with conventional message passing algorithm are shown in Table 2.

Table 1: BN specifications

Dataset	BN nodes	indicators	parameters
ALARM	37	105	509
ASIA	8	16	18
HAILFINDER	56	223	2656
INSURANCE	27	89	984
HEPAR2	70	162	1453
WIN95PTS	76	152	574
PIGS	441	1323	5618
WATER	32	116	10083

We first show the time of generating a junction tree and time of message passing algorithm. We generated a junction tree for a Bayesian network using the heuristic algorithm known as *min-fill-in* (Li and Ueno, 2015). We show the number of clusters and separators in a junction tree in the first and second columns. We show the maximum size of a separator in the third column. The first number in the third column is the number of variables in the maximum separator and the second number in the bracket is the number of all instantiations of these variables. The fourth column shows time for performing message passing algorithm once on the junction tree to get joint distribution over variables in clusters and separators.

The last four columns are the results of our proposed method with the size of ZDDs, time for generating ZDDs and time of message passing on ZDDs. As the same in junction trees, we perform a full round message passing on ZDDs and get all MLFs representing joint distributions over variables in clusters and separators. And then we evaluate these MLFs to get these joint probabilities. We

Table 2: Results for the proposed method and the conventional algorithm.

Dataset	junction tree Algorithm					ZDD-based Message Passing			
	Clus -ters	Separ -ators	Compile (ms)	Maximum separator	Inference (ms)	ZDD size	Message Variables	Compile (ms)	Infer -ence(ms)
ALARM	27	26	117	3(36)	56	6770	482	100	4
ASIA	6	5	67	2(4)	5	278	64	4	1
HAILFINDER	43	42	160	4(297)	321	45815	3116	3571	83
INSURANCE	19	18	114	6(2400)	4273	359963	13712	6313	545
HEPAR2	58	57	123	6(96)	109	18678	1376	336	67
WIN95PTS	50	49	190	7(128)	103	26477	1756	548	31
PIGS	368	367	697	10(59049)	3148153	–	248922	–	–
WATER	19	18	197	9(110592)	6119	–	515856	–	–

also show the number of message variables introduced in our approach which play an important role in our ZDD-based message passing algorithm. While passing messages with ZDDs, we need to evaluate all instantiations of separators. Thus time consumption of our method is linear time to the ZDD size and exponential to the size of separators.

From the results, we can see that the inference with our ZDD-based message passing algorithm are more efficient comparing to the conventional one. Especially for the network INSURANCE, time for message passing accelerates about 8 times.

Unfortunately, for the networks PIGS and WATER, we could not conduct our method since the numbers of massage variables become too large for our ZDD package. This is mainly because there are too many message variables to generate ZDDs. While constructing a junction tree for a given Bayesian network, most research considers to construct a junction tree that has a $treewidth$, the maximum size of a cluster, as small as possible. However in our approach, the size of separators factors the most since we have to generate two message variables for every instantiation of the separators (see Eq. (7)). According to the result of message variable numbers, for the BN of WATER, it has only about 10000 parameters but we need to generate about 1 million message variables.

Since our ZDD-based message passing is not only applicable to the conventional junction tree structure, but can also be used for any tree structure that satisfies junction tree property. Thus we consider not directly using junction trees to construct ZDDs, but selecting some of the separators in a junction tree to decompose a Bayesian network into a tree structure who also satisfies running intersection property and family preserving property. Using junction tree structure to generate ZDDs, node sharing and cache memory can not be fully utilized because there are too many independent clusters and the corresponding ZDDs nodes.

5. Related Work

Darwiche (2013) has shown a different efficient approach to factor MLFs of BN based on an arithmetic circuit called Conjunctive Normal Forms (CNFs). They also show how to compile the smallest circuits possible using a given junction tree. The main difference is that in their method, they do not use the message passing algorithm, but attribute probabilistic semantics to the partial derivatives of a network polynomial. They proposed an efficient way to evaluate and differentiate CNFs in time and space which is linear in the size of CNFs. In our method, we did not consider using the partial derivatives of polynomial functions based on ZDDs to calculate probability. We hope that if we take

into account this idea, it can bring us a significant improvement to our ZDD-based method while comping BNs.

6. Conclusion and Future Work

We proposed an improved method for exact inference in Bayesian networks to perform the message passing algorithm on ZDDs to accelerate the inference efficiency by exploiting local structures in the message passing algorithms. In some cases, the junction tree with small clusters works well in the conventional message passing algorithm, but it gives rise to a considerable number of ZDDs in our method.

As a future work, we will find proper d-separation structure to partition a given Bayesian network into several conditional independent subgraphs and then compile these small graphs into ZDDs to improve our method.

Acknowledgments

This work is partly supported by JSPS KAKENHI(S) 15H05711.

References

Mark Chavira and Adnan Darwiche. Compiling bayesian networks with local structure. In *IJCAI*, volume 5, pages 1306–1312, 2005.

Adnan Darwiche. A differential approach to inference in bayesian networks. pages 123–132, 2013.

HebrewUniversity. Bayesian network repository. `http://www.cs.huji.ac.il/~galel/Repository/`.

Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International journal of approximate reasoning*, 15(3):225–263, 1996.

Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

Chao Li and Maomi Ueno. A fast clique maintenance algorithm for optimal triangulation of bayesian networks. In *Workshop on Advanced Methodologies for Bayesian Networks*, pages 152–167. Springer, 2015.

Shin-ichi Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *DAC*, pages 272–277, 1993.

Shin Ichi Minato. Zero-suppressed bdds and their applications. *International Journal on Software Tools for Technology Transfer*, 3(2):156–170, 2001.

Shin-ichi Minato, Ken Satoh, and Taisuke Sato. Compiling Bayesian Networks by Symbolic Probability Calculation Based on Zero-Suppressed BDDs. In *IJCAI 2007, Proceedings of the 20th*

International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007, pages 2550–2555, 2007.

G. Nuel. Tutorial on exact belief propagation in bayesian networks: from messages to algorithms. *Statistics*, 2012.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.