𝕋ℂ𝕊-𝕋ℝ-𝔹-**15**-**11**

𝕋ℂ𝕊 Technical Report

Master's Thesis: Factorization of ZDDs for Fast Probability Calculation of Bayesian Networks

by

Shan Gao

Division of Computer Science

Report Series B

March 9, 2015

Hokkaido University
Graduate School of
Information Science and Technology

Email: gaoshan@ist.hokudai.ac.jp

Phone: +81-011-706-7681

Fax: +81-011-706-7681

Master’s Thesis
平成２６年度　修士学位論文

Factorization of ZDDs for Fast Probability Calculation of Bayesian Networks

ベイジアンネットワークの高速な確率計算のための ＺＤＤの因数分解

Shan Gao
高　サン

Laboratory for Algorithmics, Division of Computer Science
Graduate School of Information Science and Technology
Hokkaido University

北海道大学　情報科学研究科
コンピュータサイエンス専攻　アルゴリズム研究室

March 9, 2015

Contents

1 Introduction 2

1.1 Background 2

1.2 Contribution 2

1.3 Thesis Structure 2

2 Bayesian Networks and Multi-Linear Functions 4

2.1 Bayesian Networks 4

2.2 Inference in Bayesian Networks 5

2.2.1 Multi-Linear Functions 5

2.2.2 Related Work 5

3 Compiling BNs Based on ZDDs 9

3.1 BDDs 9

3.2 ZDDs 11

3.3 MLF Representation Using ZDD 13

3.4 Variable Ordering of ZDDs for Representing Bayesian Networks 14

4 Factorization of ZDDs 17

4.1 Weak Division Algorithm 17

4.2 Factorization of ZDDs Based on the Weak Division Algorithm 18

4.3 Factorization of MLFs 20

4.4 Factorization of MLFs Based on d-Separation of BNs Structure 21

4.4.1 d-Separation of BN 21

4.4.2 Factorization Based on d-Separation 21

5 Experimental Results 27

5.1 Experimental Results 27

6 Conclusions and Future Work 33

6.1 Conclusions 33

6.2 Future Work 33

Chapter 1

Introduction

1.1 Background

Compiling Bayesian Networks (BNs) [10] is a hot topic in probabilistic modeling and processing. Minato et al. [1] have shown an efficient method of compiling BNs into factored forms of Multi-Linear Functions (MLFs) based on Zero-suppressed Binary Decision Diagrams [2], which are a graph-based representation first used for VLSI logic design applications. In that method, they produce a set of ZDDs each of which represents MLF of each BN node. This method is more effective than conventional approach of [6] in some cases. However, the resulting ZDDs are still too large for the realistic of BNs. There is a need for more compact representation of MLFs.

The method in [4] has shown that the Weak Division Algorithm [3], a technique of logic synthesis and optimization, can be used to obtain a condensed representation of MLFs. It can be more efficient if we use the ZDD operations. Nevertheless, as the size of MLFs grows exponentially with the size of BNs, directly compressing MLFs with the Weak Division Algorithm is quite time consuming.

1.2 Contribution

In this thesis, we present an improvement of the Weak Division Algorithm by utilizing d-separation which is used to check conditional independence between variables in Bayesian Networks. First, we introduce the basic idea of representing MLFs using ZDDs and the factorization of ZDDs with the Weak Division Algorithm. Then we show how to improve the Weak Division Algorithm using d-separation structure.

1.3 Thesis Structure

The rest of this thesis is organized as follows. Section 2 introduces the Bayesian networks and MLFs, one of the most important ways of probability calculation based on

BNs. In Section 3 we introduce BDDs and ZDDs and how to compile MLFs of BN using ZDDs. Section 4 explains how to factor polynomials using the Weak Division Algorithm to condense their size based on ZDDs and our method to improve the factorization using d-separation structure of BNs. In Section 5 we show our experimental results comparing with an existing method. Finally we provide conclusions of our work and the future work in Section 6.

Chapter 2

Bayesian Networks and Multi-Linear Functions

In this chapter, we briefly introduce Bayesian networks and their Multi-linear Functions.

2.1 Bayesian Networks

A Bayesian Network is a directed acyclic graph which defines a joint distribution over a set of random variables [10]. BNs are used for representing uncertain knowledge across a number of fields. For a given BN and observed data, we calculate the probability distribution of the entire network by substituting the observed data into a portion of the BN. Modeling tools based on BNs are frequently used in real-world applications including diagnosis, forecasting, sensor fusion and manufacturing control [17].

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 its parent BN nodes. We can use the CPT to represent the probability distribution of the random variable and to predict the likelihood of uncertain events. Here we give a small example of a Bayesian network shown in Fig. 2.2 for convenience. This Bayesian Network has four nodes resulting to 24 different variable instantiations. If want to know the probability of $D = d_1$ given $A = a_1$ (we call $A = a_1$ as *evidence*). First we need to calculate the probability of all cases of B and C. Then using the result to multiply the addition of items in CPT(D) that satisfies $D = d_1$.

Although we can use CPT to answer queries, it is usually prohibitive if a BN is as huge as the example of an actual BN shown in Fig. 2.1 [8] since the size of CPT grows exponentially with the number of variables.

2.2 Inference in Bayesian Networks

For BNs have been applied to more and more complex real-world applications, the development of fast and flexible inference methods becomes increasingly significant. In the last decades, researchers have developed various kinds of algorithms for exact and approximate inference. Two of these methods are particularly well-known [19].

One of them proposes to view a BN as a CNF model counting problem [18]. It encodes the given BN into a CNF, and employs techniques used in the state-of-the-art SAT and model counting engines to solve the problem. Another one suggests to regard a BN as a *Multi-Linear Function* (MLF) [6], the so-called network polynomial. We can retrieve the answers to probabilistic queries by differentiating the MLFs. The network polynomial is exponential in size with a given BN, but it is possible to efficiently encode it. In this article, we mainly pay attention to how to encoding the MLFs.

2.2.1 Multi-Linear Functions

Multi-Linear Functions (MLFs) [6] are a well-known way to calculate probability based on BNs. An MLF 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 $P(x|\mathbf{u})$. 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. 2.2, the MLF has the following form:

$$\begin{aligned}
&\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_1|b_1c_1}\\
&+\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_2}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_2|b_1c_1}\\
&+\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_3}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_3|b_1c_1}\\
&\dots\\
&+\lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\lambda_{d_3}\theta_{a_2}\theta_{b_2|a_2}\theta_{c_2|a_2}\theta_{d_3|b_2c_2}.
\end{aligned}$$

Once we have generated the MLF for a given BN, the probability of instantiation $\mathbf{e}$ can be calculated by setting indicators that contradict $\mathbf{e}$ to 0 and other indicators to 1. Namely, we can calculate the probability in time linear in the size of MLF.

2.2.2 Related Work

Obviously, the MLF has exponential time and space complexity, so the calculation is quite time consuming. Darwiche et al. [6] have shown an efficient method for compiling BNs into factored forms of MLFs. In their method, a given BN structure is first 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) which subsumes the jointree (Fig. 2.3), one of the most influential methods based on tree-clustering for inference in BN.

Minato et al. [1] proposed a new method for compiling BNs. They directly translate a BN into a set of factored MLFs using a ZDD-based symbolic probability calculation. The MLFs may have exponential computational complexity, but ZDD-based data structure can provide a compact factored form of MLFs, and arithmetic operations can be executed in a time almost linear with the ZDD size. In their method, it is not necessary to generate the MLFs 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. In the next chapter, we will explain how this method works.

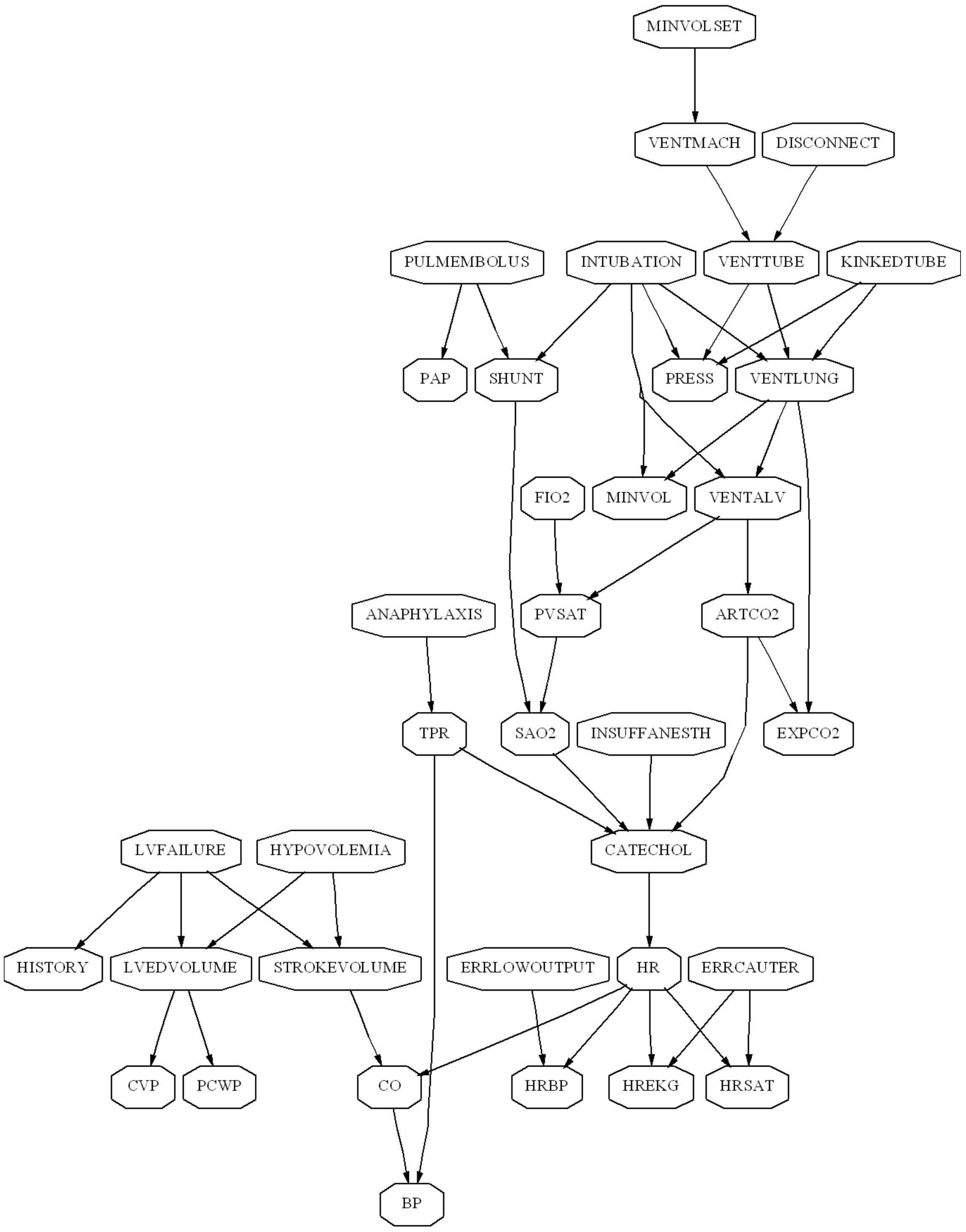
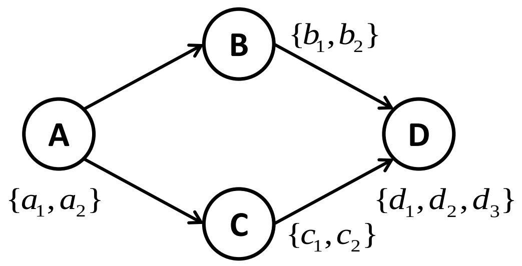


Figure 2.1. An example of an actual BN



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

AB	Prb($B\|A$)
a_1b_1	$\theta_{b_1\|a_1} = 0.2$
a_1b_2	$\theta_{b_2\|a_1} = 0.8$
a_2b_1	$\theta_{b_1\|a_2} = 0.8$
a_2b_2	$\theta_{b_2\|a_2} = 0.2$

AC	Prb($C\|A$)
a_1c_1	$\theta_{c_1\|a_1} = 0.5$
a_1c_2	$\theta_{c_2\|a_1} = 0.5$
a_2c_1	$\theta_{c_1\|a_2} = 0.5$
a_2c_2	$\theta_{c_2\|a_2} = 0.5$

BCD	Prb($D\|B,E$)
$b_1c_1d_1$	$\theta_{d_1\|b_1c_1} = 0.0$
$b_1c_1d_2$	$\theta_{d_2\|b_1c_1} = 0.5$
$b_1c_1d_3$	$\theta_{d_3\|b_1c_1} = 0.5$
$b_1c_2d_1$	$\theta_{d_1\|b_1c_2} = 0.2$
$b_1c_2d_1$	$\theta_{d_2\|b_1c_2} = 0.3$
$b_1c_2d_3$	$\theta_{d_3\|b_1c_2} = 0.5$
$b_2c_1d_1$	$\theta_{d_1\|b_2c_1} = 0.0$
$b_2c_1d_2$	$\theta_{d_2\|b_2c_1} = 0.0$
$b_2c_1d_3$	$\theta_{d_3\|b_2c_1} = 1.0$
$b_2c_2d_1$	$\theta_{d_1\|b_2c_2} = 0.2$
$b_2c_2d_2$	$\theta_{d_2\|b_2c_2} = 0.3$
$b_2c_2d_3$	$\theta_{d_3\|b_2c_2} = 0.5$

Figure 2.2. An example of BN

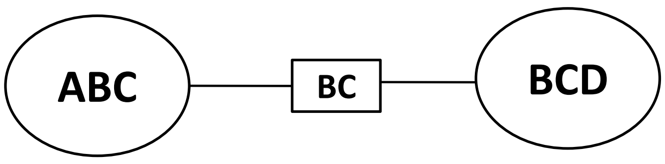


Figure 2.3. A jointree of Fig.2.1

Chapter 3

Compiling BNs Based on ZDDs

In this chapter, we first introduce Zero-suppressed BDDs (ZDDs) and then show how to compile a BN using ZDDs.

3.1 BDDs

A Binary Decision Diagram (*BDD*) is a directed graph representation of a Boolean function, as shown in Fig. 3.1(a). BDDs have two terminal nodes, which we call *0-terminal node* and *1-terminal node*, and many decision nodes with two edges, called *0-edge* and *1-edge*. A BDD is derived by reducing a binary tree graph as shown in Fig. 3.1(b). The reduction is based on the following two rules:

- Eliminate all the redundant nodes whose two edges point to the same node. (Fig. 3.2(a))
- Share all the equivalent sub-graphs. (Fig. 3.2(b))

We know that the reduction rules make BDDs compact and canonical for Boolean function under a suitable fixed variable ordering. BDDs are a powerful means for computer processing of Boolean functions. In many cases, this data structure requires less memory for storing Boolean functions and calculates values of functions faster than with truth tables or logic expressions.

BDDs were originally developed for handling Boolean function data, however, some kind of applications use BDDs not simply for representing Boolean functions, but for processing *sets of combinations* [12]. Sets of combinations often appear in solving combinatorial problems. The manipulations of sets of combinations are important techniques for many applications.

A combinatorial item set consists of elements each of which is a combination of a number of items. There are 2^n possible combinations of n items, so we have 2^{2^n} possible

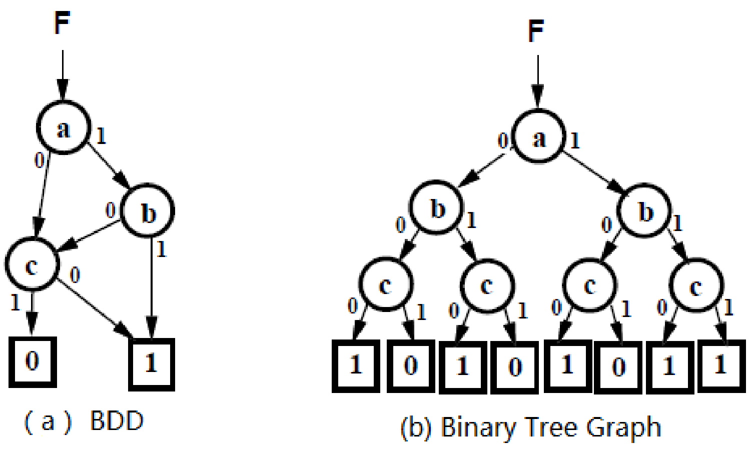


Figure 3.1. BDD and Binary Decision Tree.

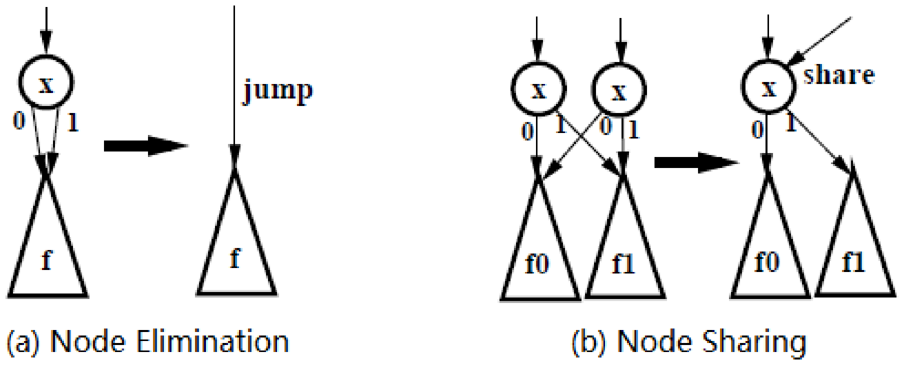


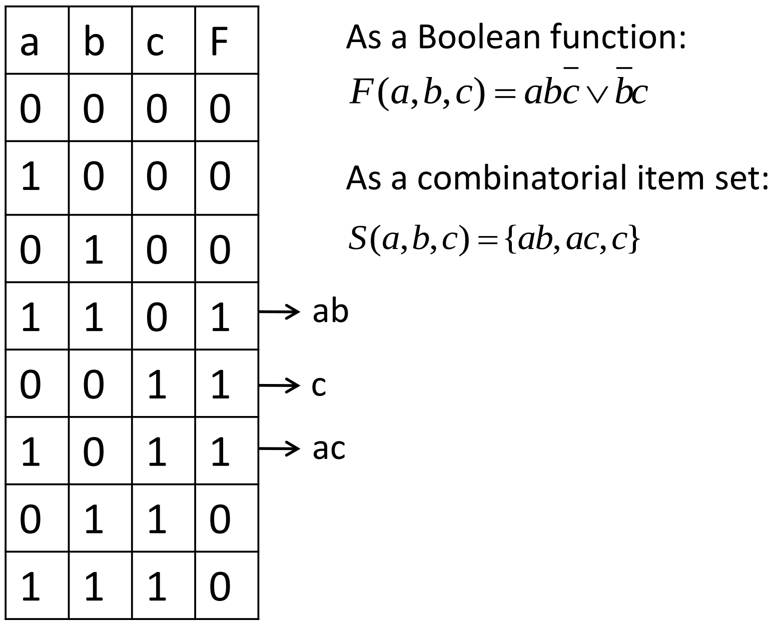
Figure 3.2. Reduction Rules on BDDs.

combinatorial item sets. For example, for a domain of five items *a*, *b*, *c*, *d*, and *e*, some 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 the empty set.

A combinatorial item set can be mapped into a Boolean space of n input variables. For example, Fig. 3.3 shows the truth table of the Boolean function $F = (ab\bar{c}) \vee (\bar{b}c)$, 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. Due to the effect of node sharing, BDDs compactly represent sets of a huge number of combinations.



a	b	c	F	
0	0	0	0	
1	0	0	0	
0	1	0	0	
1	1	0	1	$\rightarrow$ ab
0	0	1	1	$\rightarrow$ c
1	0	1	1	$\rightarrow$ ac
0	1	1	0	
1	1	1	0	

As a Boolean function:

$$F(a,b,c) = ab\bar{c} \vee \bar{b}c$$

As a combinatorial item set:

$$S(a,b,c) = \{ab, ac, c\}$$

Figure 3.3. A Boolean function and a combinatorial item set.

3.2 ZDDs

Zero-suppressed BDDs (ZDDs) [2] are a variant of BDDs for efficient manipulation of combinatorial item sets. An example of a ZDD is shown in Fig. 3.4 on the left. ZDDs 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. 3.4 on the right.
- Share equivalent nodes as in ordinary BDDs.

Notice that we do not delete nodes whose two edges point to the same node, which used to be deleted by the original BDD rule. The Zero-suppressed deletion rule is asymmetric for the two edges, as we do not delete the nodes whose 0-edges points to a terminal node. It is proved that ZDDs also give canonical forms under a fixed variable ordering like ordinary BDDs.

Here we summarize the features of ZDDs.

- In ZDDs, the nodes of irrelevant items are automatically deleted by ZDDs reduction rule.
- ZDDs are especially effective for representing sparse combinations (Fig. 3.5). For instance, sets of combinations selecting 10 out 1000 items can be represented by ZDDs up to 100 times more compact than ordinary BDDs.

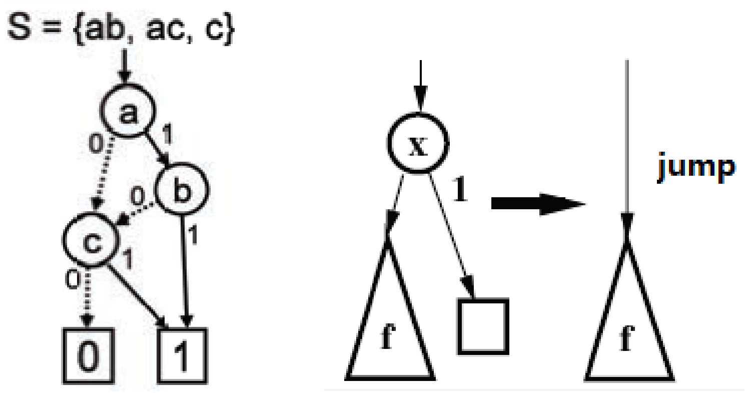


Figure 3.4. An example of a ZDD and ZDD reduction rules

- Each path from the root node to the 1-terminal node corresponds to exactly one combination in the set. Namely, the number of such paths in the ZDDs equals the number of combinations in the set. In ordinary BDDs, this property does not always hold.
- When no equivalent nodes exist in a ZDD, that is the worst case, the ZDDs structure explicitly stores all items in all combinations, as well as using an explicit linear linked list data structure. Namely, the size of ZDDs never exceeds the explicit representation.

$S(abc)$:

abc	S
000	0
100	1
010	1
110	0
001	0
101	0
011	0
111	0

$S(abcd)$:

abcd	S
0000	0
1000	1
0100	1
1100	0
0010	0
1010	0
0110	0
1110	0
0001	0
1001	0
0101	0
1101	0
0011	0
1011	0
0111	0
1111	0

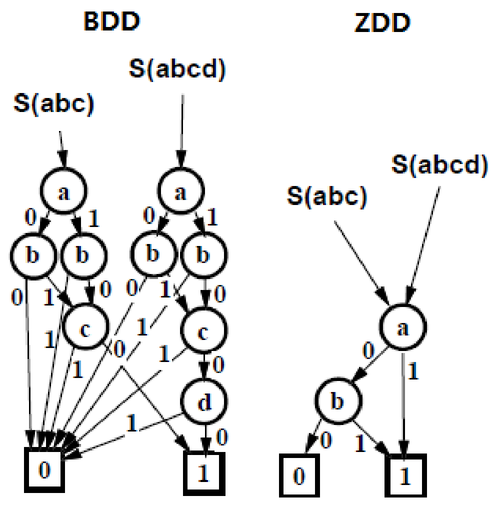


Figure 3.5. Comparing ZDD to BDD

Table 3.1 shows most of the primitive operations of ZDDs. In these operations, $\emptyset, \mathbf{1}, P$.top are executed in constant time, and the others are almost linear in the size of the graph. We can describe various processing on sets of combinations by composing these primitive operations.

Table 3.1. Primitive ZDD operations

"ϕ"	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 set. (in P but not in Q.)
P.count	Counts the number of combinations

3.3 MLF Representation Using ZDD

An MLF is a polynomial in the indicator and parameter variables. It can be regarded as a combinatorial item set. Since each term is simply a combination of variables, it can be represented compactly by a ZDD [1]. For example, the MLF at node B in Fig. 2.2 can be written as follows:

$$\begin{aligned} MLF_B = {} & \lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1} \\ & + \lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2} + \lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2}. \end{aligned}$$

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

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

An example of the ZDD for MLF_B is shown in Fig. 3.6. 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 ZDD also represents a compact factored form of MLF by sharing nodes. For the entire BN in Fig. 2.2, as shown in Fig. 3.7, 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 ZDDs (Fig. 3.8). For each BN node X, the MLF is calculated by the following operations using the MLFs at the parent nodes of X.

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

Here MLF_{X_i} denotes the value of MLF for the node X when X has the value x_i.

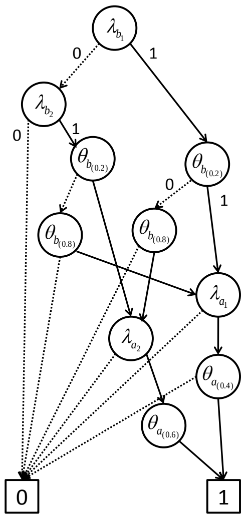


Figure 3.6. An example of a ZDD for the MLF_B.

3.4 Variable Ordering of ZDDs for Representing Bayesian Networks

Although using shared ZDDs we can condense the size of ZDDs to some extent, in some cases the size of ZDDs are still too large. Therefore we require more compact ZDDs for representing BNs. One method for simplifying ZDDs is to determine a good order of the initial variables since the size of ZDDs is sensitive to the order of nodes.

Techniques such as sequential improvement based on the exchange of the adjacent variables [9] and annealing method [16] have been proposed. Isomatsu et al. [11] tried an improvement of variable ordering method similar to these. In their method, they produce a ZDD bottom-up according to the post-order appearance of nodes using depth-first search (DFS). Then since λ variables and θ variables belonging to the same probability variable hold strong correlation, they leave these variables close to each other and recognize them as one block. Then they perform the replacement of blocks in the ZDD so the size of the ZDD may be reduced (Fig. 3.10).

Kanesaki et al. [15] proposed a improvement of this method utilizing the data that BN nodes have. In their method, they use the same algorithm to generate a ZDD as Isomatsu. The difference is when performing the DFS, they compare the number of CPT items in descending order while deciding which node to be traced. For the example in Fig. 3.9, the same as the example shown in Fig. 3.11 we start the search from node C, among nodes A, D, F adjacent to C, we proceed to node D which has the most number of items in CPT. Then between node B and E adjacent to D, we choose node E for the same reason. Repeating this process, we finally get the order of E, D, B, G, F, A, C.

Using the previous two methods, we can get a relatively better order of variables so the size of ZDDs can be somewhat reduced. However, both methods have restrictions. For the method of Isomatsu et al. [11], if a BN is too huge to be represented by ZDDs, this

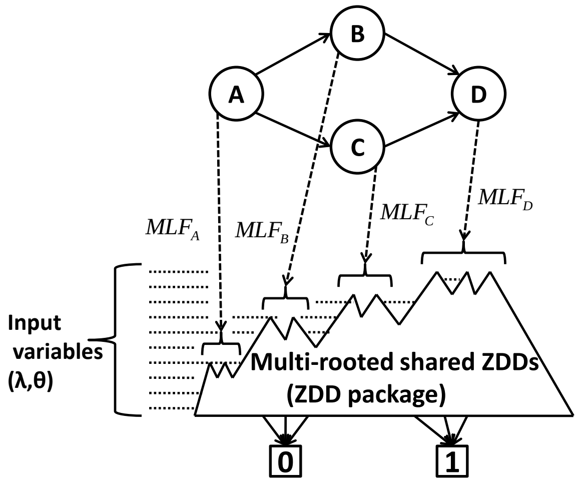


Figure 3.7. ZDD construction procedure for BN

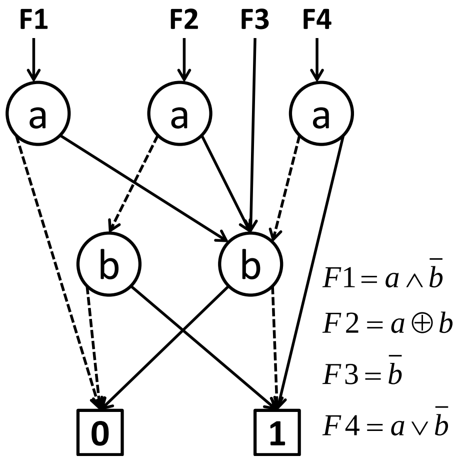


Figure 3.8. Shared ZDDs

approach can not be used. For the method of Kanesaki et al. [15], it can not work for all BNs such as the data set of *pathfinder*. Unfortunately, that approach increase the size of ZDDs in the case of data set *water*. In this thesis, we present a radically different approach to condense the size of ZDDs.

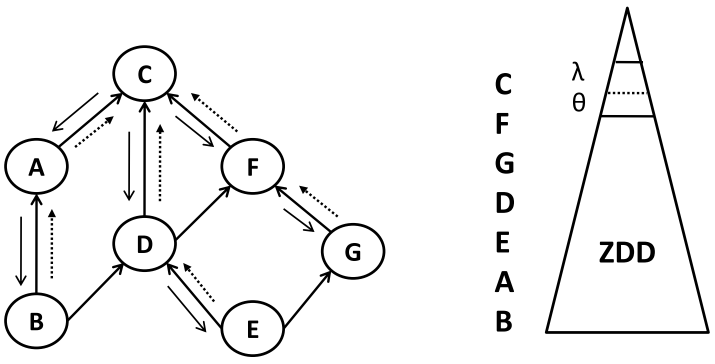


Figure 3.9. Variable Order

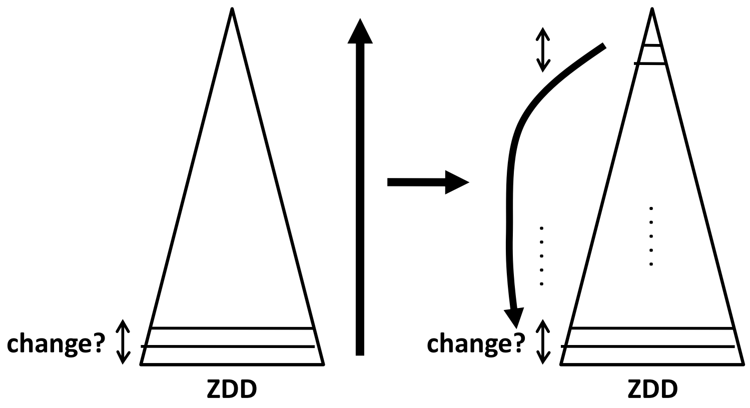


Figure 3.10. Changes of adjacent blocks

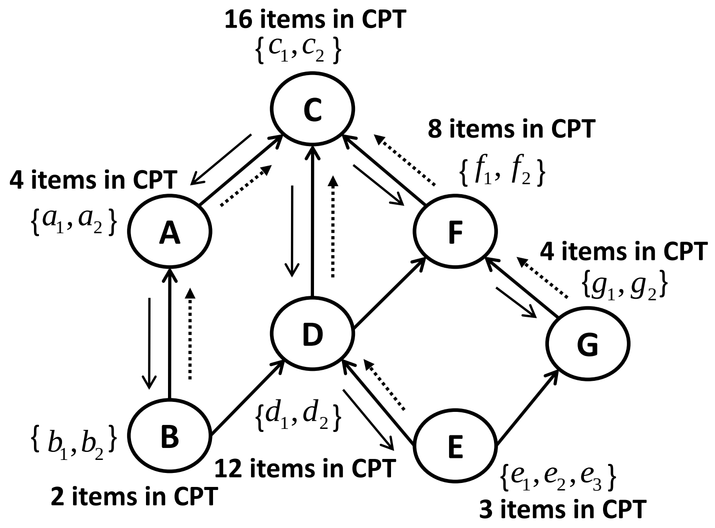


Figure 3.11. DFS considering number of items

Chapter 4

Factorization of ZDDs

For a huge polynomial function consists of literals, we can condense its size by factoring it. For example, if we rewrite function $F = abd + abe + cd + ce + acd$ as $F = pq + acd$ where $p = ab + c$ and $q = d + e$, the number of literals of F is reduced from thirteen to ten. For an MLF of a given BN, we want to condense its size through factorization.

As shown in section 3.3, MLFs can be regarded as combinatorial item sets and represented by ZDDs. In this chapter, we will discuss how to efficiently perform this factorization of MLFs based on ZDDs manipulations using a Weak Division Algorithm, the most successful and prevalent way in logic synthesis and optimization techniques. [9].

4.1 Weak Division Algorithm

Logic synthesis and optimization techniques have been used successfully for practical design of VLSI circuits in recent years. The most successful and prevalent way to attain this optimization is based on two-level logic (a form of the Boolean expressions with the AND-OR two level structure) minimization and generates multi-level logic form two-level logics by applying the Weak Division Algorithm [9].

In a two-level logic, each item is formed by a combination of literals for input variables. In general, two-level logics can be factorized into more compact multi-level logics. As illustrated in Fig. 4.1, the initial logics are represented with large two-level logics for primary output functions. When we determine a good intermediate logic, we make a two-level logic for it and reduce the other existing logics by using a new intermediate variable. Eventually, we construct a multi-level logic that consists of a number of small two-level logics.

Whenever we find a good intermediate logic, we can execute the Weak Division Algorithm by computing the common part of quotients for respective items in the divisor. For example, suppose the two expressions are :

$$f = abd + abe + abg + cd + ce + ch.$$

$$p = ab + c.$$

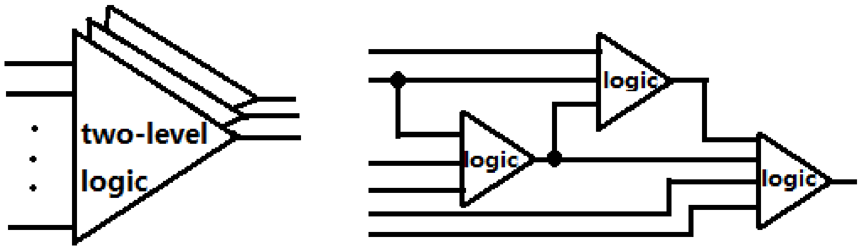


Figure 4.1. Factorization of two-level logics.

Then f can be rewritten as:

$$f = ab(d+e+g)+c(d+e+h).$$

We regard (f/p) as a quotient which can then be computed as :

$$\begin{aligned}(f/p) &= (f/(ab)) \cap (f/c)\\ &= (d+e+g) \cap (d+e+h)\\ &= d+e.\end{aligned}$$

We refer to $(f\%p)$ as remainder which is computed using the quotient as follows:

$$\begin{aligned}(f\%p) &= f - p\,(f/p)\\ &= abg+ch.\end{aligned}$$

Using the quotient and the remainder, we can reduce f:

$$\begin{aligned}f &= p(p/f)+(f\%p)\\ &= pd+pe+abg+ch.\end{aligned}$$

4.2 Factorization of ZDDs Based on the Weak Division Algorithm

The Weak Division Algorithm is efficient for functions that can be expressed in a feasible size of two-level logics, but we are sometimes faced with functions whose representations grow exponentially with the number of inputs. The use of BDDs provided a break though for that problem. By mapping a two-level logic into the Boolean space, a two-level logic can be represented as a Boolean function using a BDD. Using this method, we can represent a huge number of logics implicitly in a small storage space.

However, since BDDs were originally designed to represent Boolean functions, it can sometimes be inefficient. Minato et al. [3] used ZDDs to represent two-level logics more efficiently. ZDDs are especially effective when we manipulate two-level logics using intermediate variables to represent multi-level logic since ZDDs were adapted for representing sets of combinations. The multi-level logic consists of hundreds of two-level logics, each of which is very small. This yields so sparse combinations that the use of ZDDs is quite effective [3]. By using ZDDs, we can represent any two-level logic simply, efficiently, and uniquely.

The Weak Division Algorithm requires an execution time that depends on the length of expressions (or the number of literals in f and p) because we have to compute a number of quotients for all cubes in the divisor. This method is therefore impracticable when we deal with very large two-level logics. In Minato's method, the Fast Weak Division Algorithm has been proposed based on ZDD. The basic algorithm is described as follows:

```
$procedure(P/Q)$ {
  $if(Q = 1)\ return\ P;$
  $if(P = 0\ or\ P = 1)\ return\ 0;$
  $if(P = Q)\ return\ 1;$
  $R \leftarrow cache(``P/Q"); \ if(R\ exists)\ return\ R\ ;$
  $v \leftarrow Q.top; \ /*the\ highest\ variable\ in; Q*/$
  $(p_0, p_1) \leftarrow factors\ of\ P\ by\ v;$
  $(Q_0, Q_1) \leftarrow factors\ of\ Q\ by\ v; \ /*(Q_1 \neq 0)*/$
  $R \leftarrow P_1/Q_1;$
  $if(R \neq 0)\ if(Q_0 \neq 0)\ R \leftarrow R \cap P_0/Q_0;$
  $cache(``P/Q") \leftarrow R;$
  $return\ R;$
}
```

The basic idea is that we do not compute quotients for respective logics in the cubes in the divisor, but rather for subsets of cubes factored by an input variable. The quality of the results of this algorithm greatly depends on the choice of divisors and Minato [3] has developed a simple and fast method for finding divisors, which is shown as follows:

```
$Divisor(F)${
 $v \leftarrow a\ literal\ that\ appears\ twice\ in\ F;$
 $if(v\ exists)\ \ return\ Divisor(F/v);$
 $else\ \ return\ F;$
}
```

If there is a literal that appears more than once in a two-level logic, we compute the factor for the literal. Repeating this recursively, we eventually obtain a divisor.

4.3 Factorization of MLFs

If we consider an MLF as a polynomial, all the variables appear more than once, so we can repeatedly extract variables that appear more than once by using the Fast Weak Division Algorithm to condense its size.

```
$Divisor(MLF)\{$
  $v \leftarrow a\ literal\ appears\ twice\ in\ MLF;$
  $if(v\ exist)\ \ return\ Divisor(MLF/v);$
  $else\ \ return\ MLF;$
$\}$
```

Here we use MLF_B in Fig. 2.2 as an example of the algorithm. First we extract the divisor in MLF_B:

$$\begin{aligned}
MLF_B &= \lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1} + \\
&\quad \lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2} + \lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2} \\
&= \lambda_{a_1}\left(\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1}\right) + \\
&\quad \lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2} + \lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2} \\
&= \lambda_{a_1}\theta_{a_1}\left(\lambda_{b_1}\theta_{b_1|a_1} + \lambda_{b_2}\theta_{b_2|a_1}\right) + \\
&\quad \lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2} + \lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2}.
\end{aligned}$$

Once we find divisor $\lambda_{b_1}\theta_{b_1|a_1} + \lambda_{b_2}\theta_{b_2|a_1}$, we can factor MLF_B as follows:

$$\begin{aligned}
&MLF_B/(\lambda_{b_1}\theta_{b_1|a_1} + \lambda_{b_2}\theta_{b_2|a_1}) \\
&= (MLF_B/(\lambda_{b_1}\theta_{b_1|a_1}) \cap (MLF_B/\lambda_{b_2}\theta_{b_2|a_1}) \\
&= (\lambda_{a_1}\theta_{a_1}) \cap (\lambda_{a_1}\theta_{a_1}) \\
&= \lambda_{a_1}\theta_{a_1}.
\end{aligned}$$

The example shows that the number of variables and terms of MLF_B are reduced from 16 and 4 to 14 and 3 respectively, which means the size of the MLF has decreased so the calculation of probability inference can be faster.

However, as there are still many identical variables remaining, we can make a further factorization of the remainder which is obtained from this algorithm. In other words, divisor extraction here is inadequate. Also, since the number of characters in an MLF grows exponentially with the size of the BN, it costs too much time if we just extract one single variable from the MLF every time. In the next chapter, we will introduce an improvement of the factorization for MLF utilizing the d-separation structure of BN.

4.4 Factorization of MLFs Based on d-Separation of BNs Structure

4.4.1 d-Separation of BN

The structure of d-separation is used to check conditional independence between variables in Bayesian Networks. d-separation can be defined as following three graph patterns [7] in Fig. 4.2. In (a) is a tail-to-head or serial pattern. A affects B and B has an effect on C. If B is instantiated, A and C become independent. In (b) is a tail-to-tail or diverging pattern. If A is instantiated, all the children of A become independent. In (c) is a head-to-head or converging pattern. As long as we do not know the state of A or its descendants, B, C, ..., E can be regarded as independent sets.

The d-separation has an important property that if we substitute the observed values to the d-separation node, nodes in both sides cut by the d-separation become independent so the calculation of probability inference is simplified. Considering this advantage, we propose an improvement of divisor extraction based on d-separation of serial pattern. In our method, we use the Tarjan's algorithm [13] to find a d-separation which consists of one node in linear time with the size of the BN. However, it is not easy if we want to find a d-separation that consists of more than one node. Here we manually find a d-separation which consists of two nodes to improve the divisor extraction.

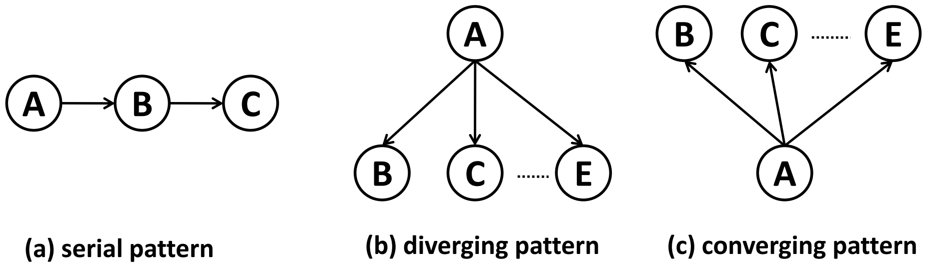


Figure 4.2. An example of d-separation.

4.4.2 Factorization Based on d-Separation

For the node B in Fig. 2.2, MLF_B contains information about node A. Here we refer to this information with parameters a_1 and a_2. Also, if the number of parameters of node A and B are given, we can forecast the size of MLF_B and the frequency of characters λ and θ. That is to say, the MLF of a node in a BN is based on its parents nodes. Therefore, we consider factoring an MLF of a node with the MLF of its parents directly instead of using the divisor extraction algorithm to extract repreated characters one at a time. But, this fails when we implement MLF_B/MLF_A. We give the details next.

$$\begin{aligned}
&MLF_B/MLF_A\\
&= MLF_B/(MLF_{a_1} + MLF_{a_2})\\
&= (MLF_B/(MLF_{a_1}) \cap (MLF_B/(MLF_{a_2})\\
&= \{(\lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1})/\lambda_{a_1}\theta_{a_1}\} \cap\\
&\quad \{(\lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1})/\lambda_{a_2}\theta_{a_2}\}\\
&= \left(\lambda_{b_1}\theta_{b_1|a_1} + \lambda_{b_2}\theta_{b_2|a_1}\right) \cap \left(\lambda_{b_1}\theta_{b_1|a_2} + \lambda_{b_2}\theta_{b_2|a_2}\right)\\
&= \emptyset.
\end{aligned}$$

We refer to the division of MLF_B/MLF_A as blotting out information about node A. Why we get the empty set is that though we try to blot out a_1 by MLF_B/MLF_{a_1}, a_1 is still left in $\theta_{b_1|a_1}$ and $\theta_{b_2|a_1}$. The same applies to a_2. When we intersect the quotients, which are obtained by factoring MLF_B with MLF_{a_1} and MLF_{a_2}, a_1 and a_2 are contrary, hence we obtain the empty set. But, if we omit the intersection, which means we perform the factorization as MLF_B/MLF_{a_1}, MLF_B/MLF_{a_2}, MLF_B can be rewritten as

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

However, even this works only in the case of a node which has only one parent node like node B. If it has more than one parent node, for example, node C in Fig. 2.2, the representations of MLF_{b_1}, MLF_{b_2} are not capable of factoring MLF_C.

$$\begin{aligned}
&MLF_D/MLF_{b_1}\\
&= (\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_1|b_1c_1} +\\
&\quad \lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_2}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_2|b_1c_1} +\\
&\quad \lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_3}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_3|b_1c_1} +\\
&\quad \ldots\\
&\quad \lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\lambda_{d_3}\theta_{a_2}\theta_{b_2|a_2}\theta_{c_2|a_2}\theta_{d_3|b_2c_2})\\
&\quad /(\lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1} + \lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2})\\
&= (\lambda_{c_1}\lambda_{d_1}\theta_{c_1|a_1}\theta_{d_1|b_1c_1} + \ldots + \lambda_{c_2}\lambda_{d_3}\theta_{c_2|a_1}\theta_{d_3|b_1c_2}) \cap\\
&\quad (\lambda_{c_1}\lambda_{d_1}\theta_{c_1|a_2}\theta_{d_1|b_1c_1} + \ldots + \lambda_{c_2}\lambda_{d_3}\theta_{c_2|a_2}\theta_{d_3|b_1c_2})\\
&= \emptyset
\end{aligned}$$

$$\begin{aligned}
&MLF_D/MLF_{b_2}\\
&=(\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_1|b_1c_1}+\\
&\quad\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_2}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_2|b_1c_1}+\\
&\quad\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_3}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_3|b_1c_1}+\\
&\quad\dots\\
&\quad\lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\lambda_{d_3}\theta_{a_2}\theta_{b_2|a_2}\theta_{c_2|a_2}\theta_{d_3|b_2c_2})\\
&\quad/(\lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1}+\lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2})\\
&=(\lambda_{c_1}\lambda_{d_1}\theta_{c_1|a_1}\theta_{d_1|b_2c_1}+\dots+\lambda_{c_2}\lambda_{d_3}\theta_{c_2|a_1}\theta_{d_3|b_2c_2})\cap\\
&\quad(\lambda_{c_1}\lambda_{d_1}\theta_{c_1|a_2}\theta_{d_1|b_2c_1}+\dots+\lambda_{c_2}\lambda_{d_3}\theta_{c_2|a_2}\theta_{d_3|b_2c_2})\\
&=\emptyset.
\end{aligned}$$

The reason we get the empty set is since MLF_B is based on node A, when we try to blot out the information about MLF_{b_1} by MLF_C/MLF_{b_1}, we are also blotting out information about a_1 and a_2 contained in MLF_D. The blotting out is inadequate because for a_1 and a_2 are also contained in MLF_C and they contradict to each other when we intersect the quotients. Thus, this motivates us to find a node set that can separate node A and node D as independent nodes so that after we factoring MLF_D, the information about node A can be cleared up thoroughly. According to the definition of d-separation, we know the set of d-separation nodes satisfies our request.

We use the BN of Fig. 2.2 as an example of the factorization based on d-separation. For nodes A and D, nodes B and C are the d-separation node set that separates them as independent nodes. Since both node B and C have two values, there are four combinations of their information b_1c_1, b_1c_2, b_2c_1 and b_2c_2. According to [1], we multiply their MLFs as follows. There are two terms in each of these MLFs, so the number of terms after multiplication should be $2*2=4$. But since the parameters λ are eliminated if they contradict each other, only two of the four terms are left. Following shows the details of the multiplication.

$$
\begin{aligned}
&MLF_{b_1}MLF_{c_1}\\
&=(\lambda_{b_1}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}+\lambda_{b_1}\lambda_{a_2}\theta_{a_2}\theta_{b_1|a_2})\\
&\quad(\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{c_1|a_1}+\lambda_{c_1}\lambda_{a_2}\theta_{a_2}\theta_{c_1|a_2})\\
&=\lambda_{b_1}\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}+\lambda_{b_1}\lambda_{c_1}\lambda_{a_2}\theta_{a_1}\theta_{b_1|a_2}\theta_{c_1|a_2}.
\end{aligned}
$$

$$
\begin{aligned}
&MLF_{b_1}MLF_{c_2}\\
&=(\lambda_{b_1}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}+\lambda_{b_1}\lambda_{a_2}\theta_{a_2}\theta_{b_1|a_2})\\
&\quad(\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{c_2|a_1}+\lambda_{c_2}\lambda_{a_2}\theta_{a_2}\theta_{c_2|a_2})\\
&=\lambda_{b_1}\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_2|a_1}+\lambda_{b_1}\lambda_{c_2}\lambda_{a_2}\theta_{a_1}\theta_{b_1|a_2}\theta_{c_2|a_2}.
\end{aligned}
$$

$$
\begin{aligned}
&MLF_{b_2}MLF_{c_1}\\
&=(\lambda_{b_2}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}+\lambda_{b_2}\lambda_{a_2}\theta_{a_2}\theta_{b_2|a_2})\\
&\quad(\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{c_1|a_1}+\lambda_{c_1}\lambda_{a_2}\theta_{a_2}\theta_{c_1|a_2})\\
&=\lambda_{b_2}\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}\theta_{c_1|a_1}+\lambda_{b_2}\lambda_{c_1}\lambda_{a_2}\theta_{a_1}\theta_{b_2|a_2}\theta_{c_1|a_2}.
\end{aligned}
$$

$$
\begin{aligned}
&MLF_{b_2}MLF_{c_2}\\
&=(\lambda_{b_2}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}+\lambda_{b_2}\lambda_{a_2}\theta_{a_2}\theta_{b_2|a_2})\\
&\quad(\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{c_2|a_1}+\lambda_{c_2}\lambda_{a_2}\theta_{a_2}\theta_{c_2|a_2})\\
&=\lambda_{b_2}\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}\theta_{c_2|a_1}+\lambda_{b_2}\lambda_{c_2}\lambda_{a_2}\theta_{a_1}\theta_{b_2|a_2}\theta_{c_2|a_2}.
\end{aligned}
$$

After these multiplications, we factor MLF_D with the four combinations respectively as follows.

$$
\begin{aligned}
&MLF_D/MLF_{b_1}MLF_{c_1}\\
&=MLF_D/(\lambda_{b_1}\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}+\\
&\qquad\qquad\lambda_{b_1}\lambda_{c_1}\lambda_{a_2}\theta_{a_1}\theta_{b_1|a_2}\theta_{c_1|a_2})\\
&=MLF_D/(\lambda_{b_1}\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1})\cap\\
&\quad MLF_D/(\lambda_{b_1}\lambda_{c_1}\lambda_{a_2}\theta_{a_1}\theta_{b_1|a_2}\theta_{c_1|a_2})\\
&=\lambda_{d_1}\theta_{d_1|b_1c_1}+\lambda_{d_2}\theta_{d_2|b_1c_1}+\lambda_{d_3}\theta_{d_3|b_1c_1}.
\end{aligned}
$$

$$\begin{aligned}
&MLF_D/MLF_{b_1}MLF_{c_2}\\
&=MLF_D/(\lambda_{b_1}\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_2|a_1}+\\
&\qquad\qquad \lambda_{b_1}\lambda_{c_2}\lambda_{a_2}\theta_{a_1}\theta_{b_1|a_2}\theta_{c_2|a_2})\\
&=MLF_D/(\lambda_{b_1}\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_2|a_1})\cap\\
&\quad MLF_D/(\lambda_{b_1}\lambda_{c_2}\lambda_{a_2}\theta_{a_1}\theta_{b_1|a_2}\theta_{c_2|a_2})\\
&=\lambda_{d_1}\theta_{d_1|b_1c_2}+\lambda_{d_2}\theta_{d_2|b_1c_2}+\lambda_{d_3}\theta_{d_3|b_1c_2}.
\end{aligned}$$

$$\begin{aligned}
&MLF_D/MLF_{b_2}MLF_{c_1}\\
&=MLF_D/(\lambda_{b_2}\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}\theta_{c_1|a_1}+\\
&\qquad\qquad \lambda_{b_2}\lambda_{c_1}\lambda_{a_2}\theta_{a_1}\theta_{b_2|a_2}\theta_{c_1|a_2})\\
&=MLF_D/(\lambda_{b_2}\lambda_{c_1}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}\theta_{c_1|a_1})\cap\\
&\quad MLF_D/(\lambda_{b_2}\lambda_{c_1}\lambda_{a_2}\theta_{a_1}\theta_{b_2|a_2}\theta_{c_1|a_2})\\
&=\lambda_{d_1}\theta_{d_1|b_2c_1}+\lambda_{d_2}\theta_{d_2|b_2c_1}+\lambda_{d_3}\theta_{d_3|b_2c_1}.
\end{aligned}$$

$$\begin{aligned}
&MLF_D/MLF_{b_2}MLF_{c_2}\\
&=MLF_D/(\lambda_{b_2}\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}\theta_{c_2|a_1}+\\
&\qquad\qquad \lambda_{b_2}\lambda_{c_2}\lambda_{a_2}\theta_{a_1}\theta_{b_2|a_2}\theta_{c_2|a_2})\\
&=MLF_D/(\lambda_{b_2}\lambda_{c_2}\lambda_{a_1}\theta_{a_1}\theta_{b_2|a_1}\theta_{c_2|a_1})\cap\\
&\quad MLF_D/(\lambda_{b_2}\lambda_{c_2}\lambda_{a_2}\theta_{a_1}\theta_{b_2|a_2}\theta_{c_2|a_2})\\
&=\lambda_{d_1}\theta_{d_1|b_2c_2}+\lambda_{d_2}\theta_{d_2|b_2c_2}+\lambda_{d_3}\theta_{d_3|b_2c_2}.
\end{aligned}$$

Finally, we can rewrite MLF_D as follows.

$$\begin{aligned}
MLF_D = &MLF_{b_1}MLF_{c_1}(\lambda_{d_1}\theta_{d_1|b_1c_1}+\lambda_{d_2}\theta_{d_2|b_1c_1}+\lambda_{d_3}\theta_{d_3|b_1c_1})+\\
&MLF_{b_1}MLF_{c_2}(\lambda_{d_1}\theta_{d_1|b_1c_2}+\lambda_{d_2}\theta_{d_2|b_1c_2}+\lambda_{d_3}\theta_{d_3|b_1c_2}.)+\\
&MLF_{b_2}MLF_{c_1}(\lambda_{d_1}\theta_{d_1|b_2c_1}+\lambda_{d_2}\theta_{d_2|b_2c_1}+\lambda_{d_3}\theta_{d_3|b_2c_1}.)+\\
&MLF_{b_2}MLF_{c_2}(\lambda_{d_1}\theta_{d_1|b_2c_2}+\lambda_{d_2}\theta_{d_2|b_2c_2}+\lambda_{d_3}\theta_{d_3|b_2c_1}.).
\end{aligned}$$

In this thesis, for a given BN, first we find a d-separation node set that consists of one node using Tarjan's algorithm and manually find a d-separation node set that consists of tow nodes. Then we multiply their MLFs and consider the result of multiplications as a divisor to factor MLFs represented by ZDD using the Weak Division Algorithm.

In the original method, they first factor each output of ZDDs with every other output. This is for saving time of finding divisor literal by literal. Then they perform a further factorization of the results by using algorithm Divisor(). As we have shown in the factorization of MLF_B/MLF_A, none output of ZDDs is able to factor other outputs, that is to say there is no need to perform the first step in previous method. Therefore we refine this algorithm by using d-separation node sets as divisors which can effectively avoid finding divisor literal by literal. Then we perform a further factorization using algorithm Divisor().

Chapter 5

Experimental Results

We implemented our experiments using an Intel Core2 Quad CPU Q9550@2.83GHz running Ubuntu 12.04LTS with 3.8GiB of main memory. We manipulate up to 40,000,000 nodes of ZDDs. We use data set of BN Benchmark [8] *alarm* and *hailfinder* to implement our experiment.

5.1 Experimental Results

Table 1 shows the original specifications of MLFs and ZDDs before factorization. The experimental results are shown in Table 2 and Table 3. The nodes corresponding to the label in *alarm* and *hailfinder* are shown in Fig. 5.1, Fig. 5.2, and Fig. 5.3 [8].

The number of nodes in *alarm* is 37 and there are 105 indicator λ and 187 parameter θ variables. The number of nodes in *hailfinder* is 56 and the number of indicator λ and parameter θ is 223 and 835, respectively.

Table 5.1. Original MLFs and ZDDs before factorization.

Dataset and node ID	Before factorization			
	ZDD size	items	total literals	time to generate ZDD from MLF
alarmN35(n6)	4448	$\geq$ 210 million	$\geq$ 1500 million	0.650
alarmN9(n34)	5803	$\geq$ 794 million	$\geq$ 1537 million	0.730s
alarmN36(n14n33)	10116	$\geq$ 500 million	$\geq$ 1000 million	0.647s
alarmN36(n20n32)	9541	$\geq$ 600 million	$\geq$ 40 million	0.614s
hailfinderN12(n4n7)	2097	$\geq$ 2 million	$\geq$ 50 million	0.633s
hailfinderN20(n4n12)	23980	$\geq$ 400 million	$\geq$ 2100 million	0.666s

In Table 2, we show the results of factorization based on d-separation consisting of one node. We find the one-node d-separation using Tarjan's algorithm [13] which is used to

Table 5.2. Experimental results of one-node d-separation.

Dataset and node ID	Previous factorization method				Proposed factorization method based on d-separation			
	ZDD size	items	total literals	factorization time	ZDD size	items	total literals	factorization time
alarmN35 (n6)	3771	1927	7330	135.765s	2934	1602	5686	78.923s
alarmN9 (n34)	8097	4136	16866	102.105s	5150	2694	10287	48.818s

Table 5.3. Experimental results of two-node d-separation.

Dataset and node ID	Previous factorization method				Proposed factorization method based on d-separation			
	ZDD size	items	total literals	factorization time	ZDD size	items	total literals	factorization time
alarmN36 (n14n33)	8007	4201	16004	2662.773s	5178	2662	9796	51.425s
alarmN36 (n20n32)	7477	3983	15139	2905.949s	4247	2423	8776	4.825s
hailfinderN12 (n4n7)	1149	859	2386	2.169s	1081	871	2358	0.304s
hailfinderN20 (n4n12)	4969	3465	11148	800.767s	4194	3187	9139	5.168s

find out cut-off nodes in an undirected graph. We chose node 36 which has the biggest ZDD in our data, but there is no one-node d-separation that can separate node 36, since no one-node d-separation is precious few in an actual BN. So we use two other nodes instead. (Fig. 5.1

In Table 3, we show the result of factorization based on d-separation consisting of two nodes. This time we manually chose the two-node d-separation without any algorithm. We choose node 36 in alarm because it has largest ZDD (Fig. 5.2). Since it would be too time consuming to factor the entire BN for hailfinder, we select the nodes in hailfinder which have ZDD size between 2,000 and 25,000 as experimental data (Fig. 5.3).

The first column in Table 2 and Table 3 shows the number of nodes and d-separation set we choose in BN. For instance, 'alarmN36(n14n33)' means factoring the MLF of node 36 with d-separation set node 14 and node 33. The two columns respectively show the specifications, such as the size of ZDD, the number of items, which represents the sum of the number of original λ, θ variables and intermediate variables representing the factored divisor, and the number of total literals in the MLF.

According to Table 2 and Table 3, we could achieve somewhat smaller ZDDs and condensed MLFs using our method. The most important property of our method is that we have saved plenty of time when factoring MLFs based on d-separation. However, we did not factor in the time of finding d-separation.

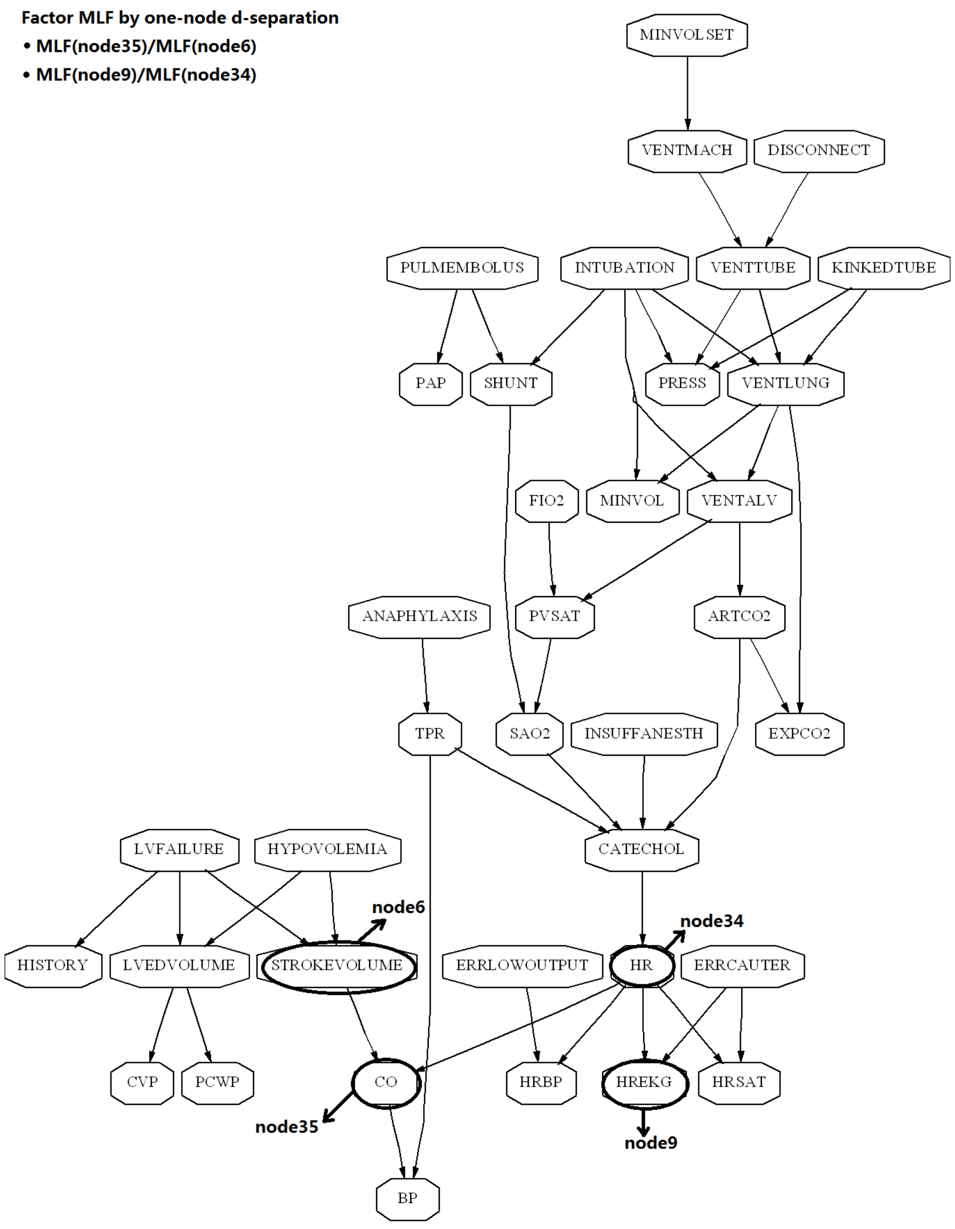


Figure 5.1. One-node d-separation in alarm.

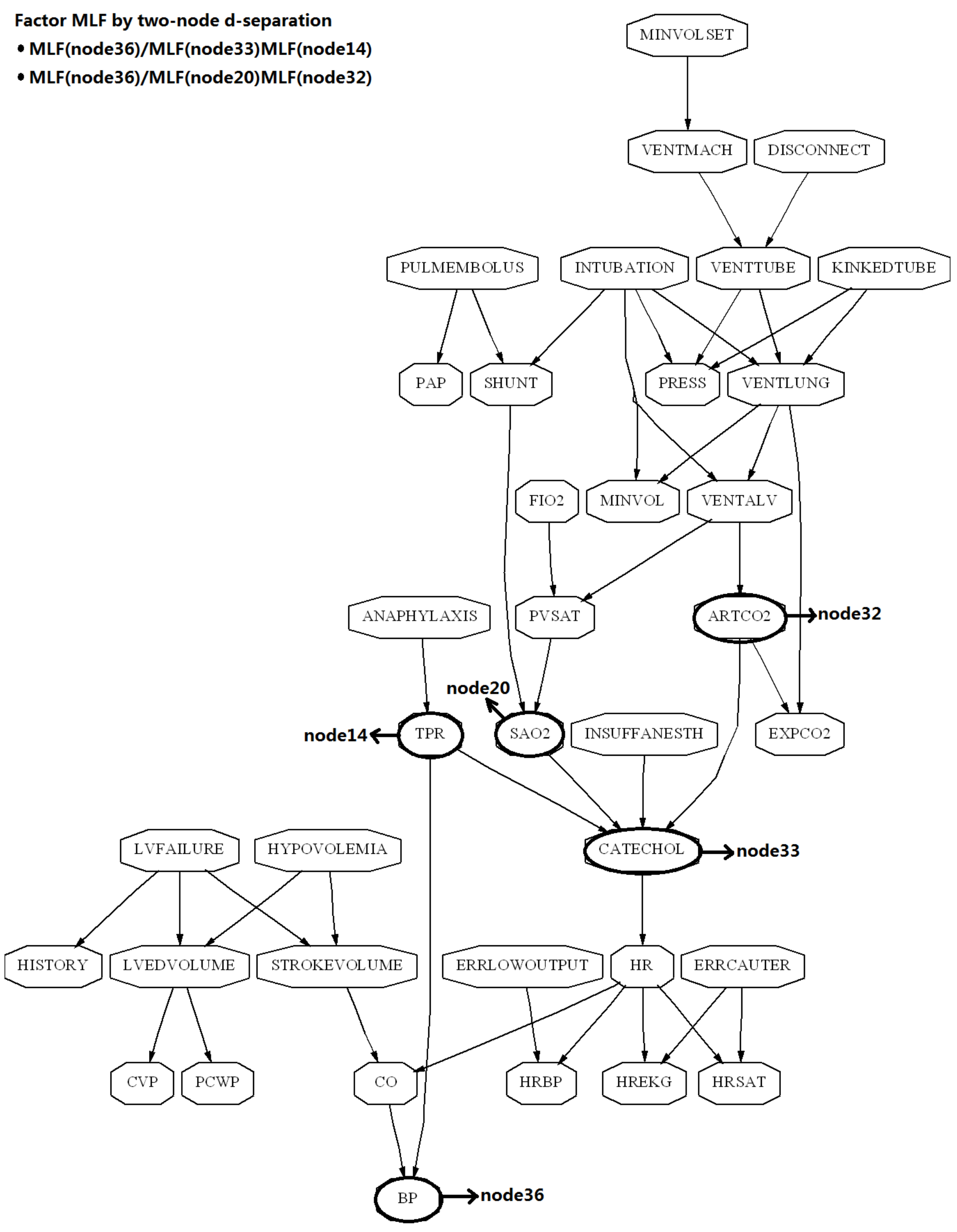


Figure 5.2. Two-node d-separation in alarm.

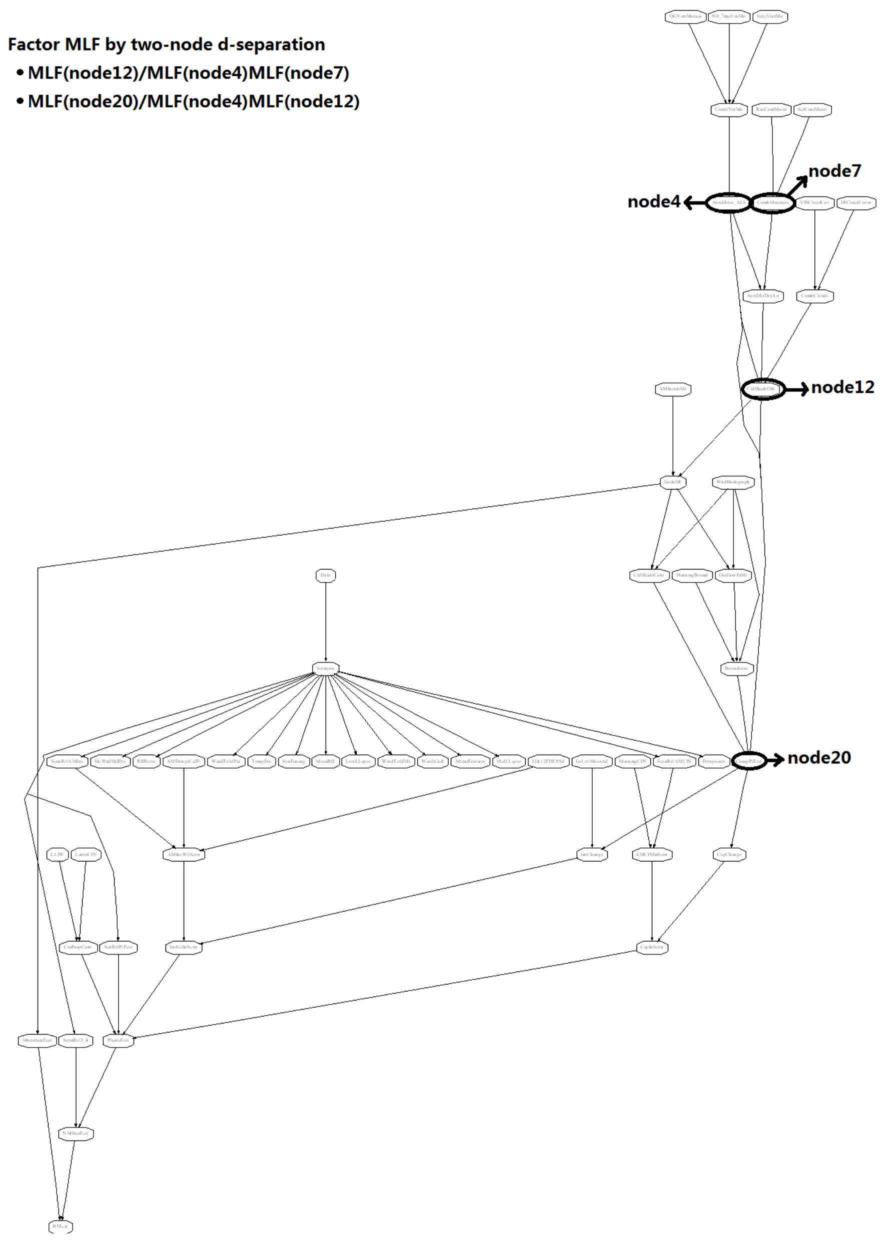

Factor MLF by two-node d-separation
• MLF(node12)/MLF(node4)MLF(node7)
• MLF(node20)/MLF(node4)MLF(node12)
node7
node4
node12
node20

Figure 5.3. Two-node d-separation in hailfinder.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have developed an improvement of the Weak Division Algorithm which is used in factorizing the MLF of a BN. In our method, we proposed to use the d-separation structure in BN as divisors to factor MLFs into compact representations. Our method effectively avoids finding divisor literal by literal and achieves somewhat more condensed MLFs while using considerably less time than directly using the original Weak Division Algorithm.

6.2 Future Work

As future work, we will consider not to generate the ZDD for the whole network but just generating the ZDD for the newly-found divisor. For a large BN, generating its ZDD is quite time consuming, so just generating the ZDD for the newfound divisor may reduce time and memory cost for probability inference calculation because the ZDD nodes of divisor can be shared. Also, we consider to use proper algorithms to find suitable d-separation node sets rather than finding d-separation manually.

Acknowledgements

I would like to express my heartfelt gratitude to Prof. Shin-ichi Minato who has instructed and helped me a lot in my research. I am grateful to Prof. Thomas Zeugmann and Assistant Prof. Charles Jordan who gave a lot of suggests and helped me improve my writing skills a lot. I am also indebted Yuma Inoue and all members in the Laboratory of Algorithm who helped me with my research and this thesis.

Bibliography

[1] S. Minato, K. Satoh, and T. Sato Compiling Bayesian Networks by Symbolic Probability Calculation Based on Zero-suppressed BDDs. *IJCAI*, 2007: 2550-2555.

[2] S. Minato Zero-suppressed BDDs for set manipulation in combinatorial problems. *Design Automation, IEEE*, 1993: 272-277.

[3] S. Minato Binary decision diagrams and applications for VLSI CAD. *Springer Science & Business Media*, pp.81-94, 1996.

[4] 高サン, 湊真一. ベイジアンネットワークの確率計算のための ZDD 分解法に関する実験と考察. *FIT*, 2014: 59-60.

[5] P. Shenoy and G. Shafer Propagating belief functions with local computations. *IEEE*, 1986, 1(3): 43-52.

[6] A. Darwiche A differential approach to inference in Bayesian networks. *JACM*, 2003, 50(3): 280-305.

[7] Word Press Entries (RSS) and Comments (RSS), D-separation.
http://www.bayesnets.com/D-separation.html.

[8] Marco S. Bayesian network Repository. http://www.bnlearn.com/bnrepository/

[9] M. Fujita, Y. Matsunaga, and T. Kakuda On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. *Proceedings of the conference on European design automation, IEEE*, 1991: 50-54.

[10] A. Darwiche A logical approach to factoring belief networks *[J]. KR*, 2002, 2: 409-420.

[11] K. Isomatsu, S. Minato ベイジアンネットワークを表現するゼロサプレス型 BDD の変数順序付けに関する実験と考察. *FIT*, 2008: 433-435.

[12] S. Minato Zero-suppressed BDDs and their applications *International Journal on Software Tools for Technology Transfer*, 2001, 3(2): 156-170.

[13] R. Tarjan Depth-first search and linear graph algorithms *[J]. SIAM journal on Computing*, 1972, 1(2): 146-160.

[14] S. Acid, L. M. De Campos An algorithm for finding minimum d-separating sets in belief networks. *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*,1996: 3-10.

[15] 金崎健之, 湊真一. ベイジアンネットワークを表現する ZDD の初期変数順序付け方法の改良*(*人工知能・ゲーム, 一般論文*)[J].*, 情報科学技術フォーラム講演論文集, 2009, 8(2): 553-555.

[16] N. Ishiura, H. Sawada, S. Yajima . Minimazation of Binary Decision Diagrams Based on Exchanges of Variables *ACM/IEEE International Conf. on Computer-Aided Design (ICCAD-91)*, 1991, 91: 472-475.

[17] N.E. Fenton, and M. Neil Risk Assessment with Bayesian Networks *CRC Press*, 2012.

[18] T. Sang, P. Beame, H. Kautz Solving Bayesian networks by weighted model counting *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*. 2005, 1: 475-482.

[19] M. Wachter, R. Haenni Logical compilation of Bayesian networks with discrete variables *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Springer Berlin Heidelberg, 2007: 536-547.