F-061

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
An Improvement of Initial Variable Ordering of ZDDs for Representing Bayesian Networks

金崎 健之 † Takeshi Kanasaki
湊 真一 † Shin-ichi Minato

1. はじめに

ベイジアンネットワーク（以後BNと書く）は，グラフ構造による確率モデルの表現方法の一種であり，近年，様々な用途に広く用いられている．最近，VLSI CADの分野で大規模論理関数データの表現方法として広く用いられている二分決定グラフ (BDD:Binary Decision Diagrams)，その中でも「ゼロサプレス型BDD(ZDD:Zero-suppressed BDD)」[2] と呼ばれるデータ構造を用いて，BNを表現し，効率よく確率計算を行う手法が提案されている [3].

本稿では，与えられたBNに対して様々なZDDの初期変数順序付けを適用し，確率計算をより効率よく実現する手法を発見することを目的とする．

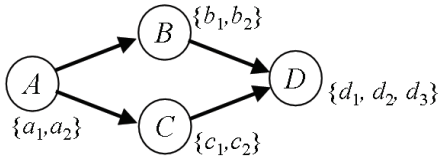
2. BNとMLF式

BNは図1に示すような確率モデルを表現する非巡回有向グラフである．各々のノード（BNノードと呼ぶ）は，それぞれ独立した個別の確率関数 X を持っている．X は一般的に多値の変数であり，$\{x_1, x_2, \ldots, x_k\}$ のいずれかの値を取るとする．また各BNノードは，これらの上流側のBNノードの確率変数の値に依存する条件付き確率テーブル（CPT: Conditional Probability Table）を持ち，これにより確率変数の確率分布が表現されている．

BNの確率分布を計算するための方法の一つとして，Multi-Linear Function（MLF）と呼ばれる数式を生成する方法が知られている．MLF式は1つの確率変数 x に対して,2種類の論理関数を使って表現する．1つは X の確率分布の数値を記号的に表現する λ 関数と，もう1つは X の値を表現する θ 関数である．以下にMLF式の例を示す.

$$
\begin{aligned}
&\lambda_{a1}\lambda_{b1}\lambda_{c1}\lambda_{d1}\theta_{a1}\theta_{b1|a1}\theta_{c1|a1}\theta_{d1|b1c1}\\
&+\lambda_{a1}\lambda_{b1}\lambda_{c1}\lambda_{d2}\theta_{a1}\theta_{b1|a1}\theta_{c1|a1}\theta_{d2|b1c1}\\
&+\lambda_{a1}\lambda_{b1}\lambda_{c1}\lambda_{d3}\theta_{a1}\theta_{b1|a1}\theta_{c1|a1}\theta_{d3|b1c1}\\
&+\lambda_{a1}\lambda_{b1}\lambda_{c2}\lambda_{d1}\theta_{a1}\theta_{b1|a1}\theta_{c2|a1}\theta_{d3|b1c2}\\
&+\ldots\\
&+\lambda_{a2}\lambda_{b2}\lambda_{c2}\lambda_{d3}\theta_{a2}\theta_{b2|a2}\theta_{c2|a2}\theta_{d3|b2c2}
\end{aligned}
$$

与えられたBNのMLF式を生成すれば，ある観測データに対する確率変数の確率分布を機械的に求めることができる．この確率計算に要する時間は，MLF式の長さに比例するが，一般にMLF式は元のBNサイズに対して指数関数的に大きくなるため，その計算は容易ではない．ただし，MLF式をうまく因数分解して，コンパクトな算術式として表現することができれば，確率計算を高速化することができる．この因数分解を行う一つの手段として，次に述べるZDDを用いた手法がある．



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

AB	$\Pr(B\|A)$
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	$\Pr(C\|A)$
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	$\Pr(D\|B,C)$
$b_1c_1d_1$	$\theta_{d_1\|b_1c_1} = 0.0$
$b_1c_1d_2$	$\theta_{d_2\|b_1c_1} = 0.5$
$b_1c_1d_3$	$\theta_{d_3\|b_1c_1} = 0.5$
$b_1c_2d_1$	$\theta_{d_1\|b_1c_2} = 0.2$
$b_1c_2d_2$	$\theta_{d_2\|b_1c_2} = 0.3$
$b_1c_2d_3$	$\theta_{d_3\|b_1c_2} = 0.5$
$b_2c_1d_1$	$\theta_{d_1\|b_2c_1} = 0.0$
$b_2c_1d_2$	$\theta_{d_2\|b_2c_1} = 0.0$
$b_2c_1d_3$	$\theta_{d_3\|b_2c_1} = 1.0$
$b_2c_2d_1$	$\theta_{d_1\|b_2c_2} = 0.2$
$b_2c_2d_2$	$\theta_{d_2\|b_2c_2} = 0.3$
$b_2c_2d_3$	$\theta_{d_3\|b_2c_2} = 0.5$

図 1: ベイジアンネットワークの例

3. ZDDによるBN表現

3.1 BDDとZDD

BDDは，図2に示すような論理関数のグラフによる表現である．一般に，論理関数のそれぞれの変数について，0,1の値を代入した結果を，二分岐の枝（0-枝 1-枝）で場合分けし，最終的に得られる論理関数の値を2値の定数節点（0-終端節点/1-終端節点）で表現すると，図2のような二分木状のグラフになる．このとき，場合分けする変数の順序を固定し，冗長な節点の削除と等価な節点の共有という2つの縮約規則を可能な限り適用することにより，既約」な形が得られ，論理関数をコンパクトかつ一意に表せることが知られている [1,4].

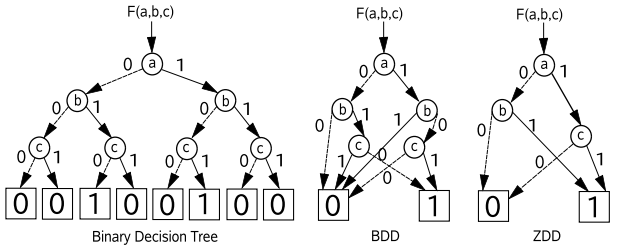


図 2: 二分決定木と BDD，ZDD

†北海道大学大学院情報科学研究科

BDD は元々は論理関数を表現するために考案されたものだが，これを用いて組み合わせ集合データを表現・操作することもできる．組合せ集合を BDD で表現するとき，類似する組合せが多ければ，部分的に共通する組合せがグラフ上で共有されて，記憶量や計算時間が大幅に削減される可能性があり, さらに, 組合せ集合に特化した ZDD を用いると, 一層効率よく扱うことができる．

ZDD では，冗長な節点を削除する簡約化規則が通常の BDD と異なる．ZDD では図 2 のように 1-枝が 0-終端節点を直接指している節点を取り除く，という規則になっている．この簡約化規則により，組合せ集合に影響を与えない（一度も選ばれることのない）アイテムに関する節点が自動的に削除されることになり，通常の BDD よりも効率よく組合せ集合を表現・操作することができる．

3.2 ZDD による BN 表現

MLF 式は λ 変数と，θ 変数の値からなる多項式で，それぞれの項が単に変数の組み合わせであるため，組み合わせ集合とみなすことができる．よって ZDD としてコンパクトに表現することができ，また，確率計算においても ZDD サイズに比例した時間で行うことができる．例えば，図 1 のノード B についての MLF 式を見てみると，以下のようになる．

$$MLF(B) = \lambda_{a1}\lambda_{b1}\theta_{a1}\theta_{b1|a1} + \lambda_{a1}\lambda_{b2}\theta_{a1}\theta_{b2|a1} + \lambda_{a2}\lambda_{b1}\theta_{a2}\theta_{b1|a2} + \lambda_{a2}\lambda_{b2}\theta_{a2}\theta_{b2|a2}$$

ここで，等しいパラメーターが同じ変数を共有するようにパラメーター変数を置き換える．

$$MLF(B) = \lambda_{a1}\lambda_{b1}\theta_{a(0.4)}\theta_{b(0.2)} + \lambda_{a1}\lambda_{b2}\theta_{a(0.4)}\theta_{b(0.8)} + \lambda_{a2}\lambda_{b1}\theta_{a(0.6)}\theta_{b(0.8)} + \lambda_{a2}\lambda_{b2}\theta_{a(0.6)}\theta_{b(0.2)}$$

MLF(B) による ZDD の構造により，指数関数的に MLF 式が巨大になったとしても，類似した部分を共有する事で場合によって数十倍もの縮約効果を得ることができる．なお，MLF 式は一つの確率変数が多くの λ 変数，θ 変数を持つため，関係のある変数を並べて配置している．ZDD のノードは，それぞれの単純な変換規則を備えた算術演算手順と解釈することができる．これは，MLF 式の ZDD を生成した後に，ZDD のサイズに比例する回数の算術演算手順が容易に得られるということを意味している．

4. BN を表現する ZDD の初期変数順序付け

BN が中規模以上になると，MLF 式が指数的に膨れ上がって行くため，ZDD を用いても，BN を表現する MLF 式を一度で全て表現することが非常に困難になってくる．第 3 章で述べた通り，ZDD の性質として, 変数の順序がサイズに大きく影響していることが分かっており, この順序付けによるサイズの影響は, 場合によっては数十倍にまで及ぶことがある．このため，ZDD をコンパクトに表現するための変数順序付けが非常に重要な要素となっている．

4.1 従来の順序付け方法

従来の順序付け方法では，ZDD を作成する際の変数の初期順序は，BN データでの変数の出現順に深さ優先で帰りがけの出現順で，ZDD の最下層から積み上げていく順序付けとなっている．ここで, 図 3 に示す BN を例に挙げ，従来の順序付け方法について説明する．

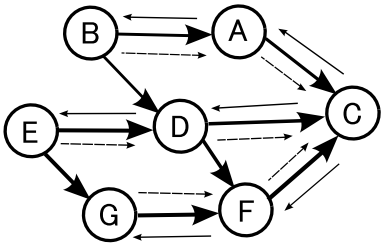


図 3: 従来手法による例

この BN の例では，BN データにおける出現順をアルファベットの表記の通りであるとし, 探索開始点は BN ノード C とする．BN ノード C から探索を開始し，C から見て最も深い位置にある BN ノード B の ZDD を最初に積み上げ, 次に BN ノード A の ZDD を積み上げ…と繰り返して行くと, 最終的に B,A,E,D,G,F,C の順になる．上流ノードの変数が ZDD の下位に来る方が良い傾向があるので，この順序付けは比較的良いとされている．ただし，従来手法では上流ノードが複数存在する際にどちらを先に辿るかは元のデータ出現順としていた．

4.2 新たに提案する順序付け方法

新たに提案する手法は，深さ優先で帰りがけの出現順で，ZDD の最下層から積み上げていく順序付けという点では従来手法と同じであるが，従来手法と異なるのは，ZDD を生成する過程でその初期変数順序付けを行う際に，

- どのノードから探索を開始するか
- 探索中に上流ノードが複数存在する場合, どちらを先に辿るか

を，それぞれ独立した確率変数 X，条件付き確率テーブル (CPT)，ノードに入ってくる親ノードの数である入次数，ノードから出て行く子ノードの数である出次数など, これらの BN データ内の各ノードが持つデータを参照した上で，実験者が指定している点である．

ここで, 先程図 3 に示した BN と同じ BN を例に挙げ, CPT の項目数に注目し，降順で初期変数順序付けを行った具体的な例を図 4 に示す．この例では，図 3 で示した例と比較するため, 図 3 で示した例と同様に BN ノード C から探索を行う．その際にどの BN ノードを辿るかについては，CPT の項目の数を昇順に比較しながら進む点で従来手法とは異なっている．まず,BN ノード C から探索を開始し, ノード C に隣接するノード A,D,F の内,CPT の項目数の最も多い D へ進む. 最終的に E,D,B,G,F,A,C の順になる．前節での順序付け方法と比較すると，ZDD の生成される順序が異なっていることがわかる．

表 1: 探索開始点を従来手法と変化させた実験結果

		BN から生成された ZDD のサイズ		
		従来手法	今回提案する手法	
BN name	BN nodes	データ出現順	探索開始点	データ出現順
alarm	37	14986 (1.00)	N36v1	3801 (0.25)
pathfinder(pf1)	109	1489 (1.00)	N81v2	1168 (0.78)
pathfinder(pf23)	135	979 (1.00)	N31v0	979 (1.00)
Pigs	441	14921 (1.00)	N96v1	2725 (0.18)
water	32	13968 (1.00)	N26v1	15792 (1.13)

表 2: 探索順を従来手法と変化させた実験結果

	BN から生成された ZDD のサイズ						
	今回提案する手法						
BN name	データ出現順	CPT 降順	CPT 昇順	fin 降順	fin 昇順	fout 降順	fout 昇順
alarm	3801 (1.00)	6292 (1.64)	18298 (4.88)	6292 (1.64)	17260 (4.54)	6292 (1.64)	17260 (4.54)
pathfinder(pf1)	1168 (1.00)	1168 (1.00)	1489 (1.27)	1327 (1.14)	1168 (1.00)	1327 (1.14)	1489 (1.27)
pathfinder(pf23)	979 (1.00)	979 (1.00)	979 (1.00)	979 (1.00)	979 (1.00)	979 (1.00)	979 (1.00)
Pigs	2725 (1.00)	1928 (0.71)	14921 (5.48)	13273 (4.87)	3896 (1.43)	1928 (0.71)	13273 (4.87)
water	15792 (1.00)	8693 (0.55)	49157 (3.11)	8717 (0.55)	49170 (3.11)	13968 (0.88)	13968 (0.88)

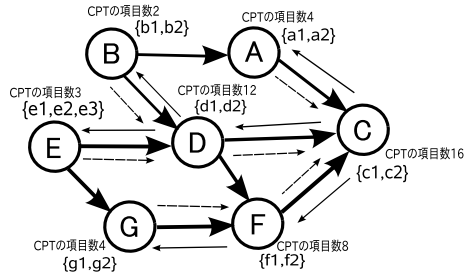


図 4: 新たに提案する手法による例

5. 実験結果と考察

本実験において使用した PC は Pentium4，3.0GHz, SuSE Linux 10.3，主記憶 512Mbyte で ZDD の最大節点数は 1,000 万個としている．

代表的な BN データに対して,BN を表現する ZDD の初期変数順序付けについて，様々な順序付けを行い，ZDD のノード数がどれだけ変化するのかを調べた．その結果を表 1 と表 2 に示す．本実験では, 従来の順序付け方法で一番 ZDD が大きくなった BN ノードに注目し，その ZDD のサイズがいかに変化するかを調査した．

表 1 中で BN name は対象とした BN データ名，BN nodes はノード数，探索開始点は今回提案する手法で一番最初に ZDD を生成する BN ノードを表し, データ出現順は，BN データの出現順で探索を行う従来の順序付け方法を表している．探索開始点には従来の順序付け方法で ZDD のサイズが最大になるノードを選択する．ZDD のサイズの後に付く () の中身は，従来手法に比べて ZDD のサイズが何倍になったかを表している．

表 2 中で CPT は CPT の項目数，fin,fout はそれぞれ入次数，出次数に注目し，降順と昇順で参照して順序付けを行っていることを表している．() の中身は，今回提案する手法で BN データの出現順で探索を行う従来の順序付け方法の ZDD のサイズを 1.00 とし，その手法と比較して，分岐を辿る方法を変化させた場合，ZDD のサイズが何倍になるかを表している．実験にかかる時間は ZDD のサイズにほぼ比例する．

表 1 に見られるように一番効果の現れた Pigs のデータでは，0.18 倍まで ZDD のサイズを小さくできた．また，表 2 に見られるように，探索方法を変化させることにより，最大で従来の 0.71 倍まで ZDD のサイズを小さくできた．pathfinder(pf23) のデータに対してはどの手法でも ZDD のサイズが全く変化しなかった．また，CPT,fin,fout どれかが常に優位という訳ではないようであった．

このように，MLF 式から生成された ZDD の初期変数順序付け方法に関して，どんなデータベースに対しても一様な縮約効果が得られるという手法は，今回の実験では発見できなかった．

深さ優先でどちらの枝を先に辿るかということを変化させただけで，ZDD のサイズが数倍変化する例題があることがわかった．一部の BN データに対してこの改善法で効果があるということは分かったので，今後はどのような BN に対して順序付けの影響が現れるのかを調べ，良い順序付け法の開発を目指したい．

最後にご指導頂いた Thomas Zeugmann 教授に感謝いたします．

参考文献

[1] S. B. Akers, Binary decision Diagrams, IEEE Trans. Comput., C-27, 6 (1978), 509-516.

[2] 湊真一, VSOP: ゼロサプレス型 BDD に基づく「重み付き積和集合」 計算プログラム, IEICE Technical Report COMP2005-13(2005-5)

[3] S. Minato,K. Satoh, and T.Sato, ”Compiling Bayesian Networks by Symbolic Probability Calculation Based on Zero-suppressed BDDs”, In Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI-2005),pp. 2550-2555, Aug. 2005.

[4] C. Y. Lee, Representation of switching circuits by binary-decision programs, Bell Sys. Tech. Jour., 38 (1959), 985-999.