4K1-OS-16a-03

フロンティア法によるDAGの非巡回縮約の列挙

Enumerating Acyclic Contractions of a Directed Acyclic Graph Using Frontier-Based Search

中畑 裕 *1 Yu Nakahata　鈴木 浩史 *2 Hirofumi Suzuki　石畠 正和 *2 Masakazu Ishihata　堀山 貴史 *3 Takashi Horiyama

*1奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

*2北海道大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Hokkaido University

*3埼玉大学 大学院理工学研究科
Graduate School of Science and Engineering, Saitama University

A directed acyclic graph (DAG) is a fundamental structure in computer science. In particular, contracting a DAG into a smaller DAG is important for applications such as the instruction-set extension problem. In this paper, we propose an algorithm to enumerate all the edge sets such that contracting all its edges makes the input graph another DAG. Our algorithm is based on frontier-based search, which is a framework of algorithms to construct a zero-suppressed binary decision diagram (ZDD) representing the desired family of edge sets.

1. はじめに

有向非巡回グラフ (Directed acyclic graph, DAG) は計算機科学における基本的な構造であり，プログラムの解析，バージョン管理，確率推論など様々な分野で利用されている．これらの応用において，DAG をより小さな DAG に縮約するという問題は重要である．例として命令セット拡張問題 [Galuzzi 11] を挙げる．命令セット拡張問題では，入力としてプログラムを表す DAG が与えられる *1．頂点は命令を，辺は命令間のデータの入出力関係を表す．DAG を縮約するとき，同じ頂点にまとめられる命令の集合を 1 つの拡張命令とみなすことができる．拡張命令間のスケジューリングが行えるために，縮約後のグラフは DAG でなければならないという制約がある．DAG の辺部分集合であって，それを縮約した結果が DAG になるようなものを非巡回縮約と呼ぶことにする（詳しくは 2.1 節で定義する）．例えば，図 1 左の DAG において辺 e_1 を縮約すると図中央のように DAG となるので $\{e_1\}$ は非巡回縮約である．一方，e_1 に加えて e_4 を縮約すると図右のように有向閉路が発生するため，$\{e_1, e_4\}$ は非巡回縮約でない．このように，DAG からどのような辺部分集合を選んだとき非巡回縮約となるかは自明でないため，応用上の様々な制約（各拡張命令に含まれる命令の個数，命令 A と命令 B は同じ拡張命令に入れてはならないなど）を満たす非巡回縮約を効率的に見つけることは困難であると考えられる．また，実用上は最適解を 1 つ見つけるだけではなく，条件を満たす解を複数列挙することで解を比較検討しよりよい解を選ぶことができる．

グラフが与えられたとき，指定した条件を満たす辺の選び方を効率的に列挙するための枠組みとして，フロンティア法 [Kawahara 17] が提案されている．フロンティア法は，指定した条件を満たす辺の選び方の集合を表すゼロサプレス型二分決定グラフ (Zero-suppressed binary decision diagram, ZDD) [Minato 93] を効率的に構築するアルゴリズムの枠組みである．ZDD は集合族を圧縮して表現するデータ構造であり，グラフの辺の選び方の集合，すなわち辺の集合族を効率よく表現できる．フロンティア法の計算時間は列挙される辺の選び方の個数ではなく出力の ZDD のサイズに依存する．ZDD のサイズは集合族を陽に書き表すよりも指数的に小さくなりうるため，フロンティア法を用いれば解を 1 つ 1 つ列挙するアプローチよりも指数的に高速に解を列挙できる場合がある．さらに，ZDD は集合族に対する様々な操作を提供している．例えば，2 つの ZDD が与えられたとき，それらが表す集合族の積集合や和集合を表す ZDD を構築するアルゴリズムが知られている．これらの操作を用いて効率的な解の絞り込みが可能となる．

連絡先: 中畑裕，奈良先端科学技術大学院大学情報科学研究科，奈良県生駒市高山町 8916-5，0743-72-5368，nakahata.yu.nm2@is.naist.jp

*1 プログラムのループはアンロールされているものとする．これにより入力グラフを DAG で表すことができる．

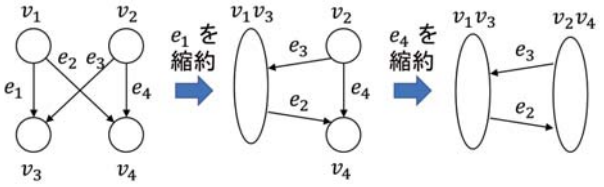

図 1: 有向非巡回グラフ (DAG) とその縮約の例.

本稿では，DAG が与えられたとき，その非巡回縮約をすべて列挙するアルゴリズムを提案する．提案手法はフロンティア法に基づき，非巡回縮約の集合を表す ZDD を構築する．計算機実験により提案手法の性能を評価する．

2. 準備

2.1 諸定義

正の整数 k に対し $[k] = \{1, 2, \ldots, k\}$ とする．また $[0] = \emptyset$ とする．集合 S に対し，S のべき集合を $\mathfrak{P}(S)$ と書く．$\mathcal{T} \subseteq \mathfrak{P}(S)$ が $\bigcup_{T \in \mathcal{T}} T = S$ かつ $\forall T_1, T_2 \in \mathcal{T}, T_1 \neq T_2 \Rightarrow T_1 \cap T_2 = \emptyset$ を満たすとき $\mathcal{T}$ を S の分割という．$\mathcal{T}$ が S の分割であるとき，各 $T \in \mathcal{T}$ を $\mathcal{T}$ のセルという．

本稿では有向グラフ $G = \langle V, E \rangle$ を考える．$V =$

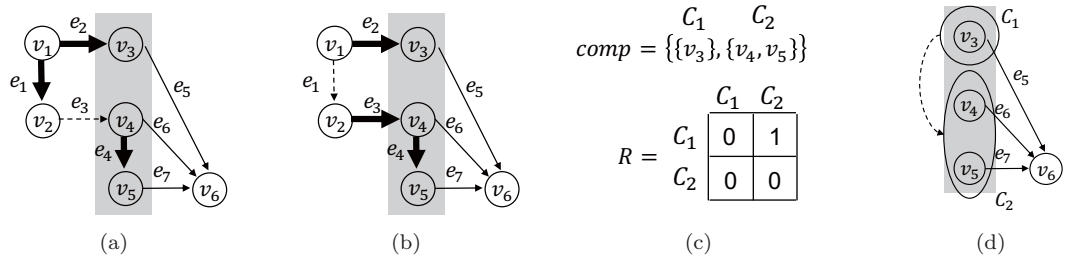

図 2: (a), (b) はフロンティア法において，辺 e_1 から e_5 までを処理し終えた直後の途中過程．太線は縮約することにした辺を，点線は縮約せず残すことにした辺を，実線は未処理の辺を表す．網掛けの領域の内部に存在する頂点の集合がフロンティアである．(a),(b) に対応する configuration はともに (c) となる．(a), (b) の太線の辺を縮約するとともに (d) のようになる．

$\{v_1, v_2, \ldots, v_n\}$ は頂点集合，$E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ は辺集合とする．辺 $e = (a, b)$ に対し，a を e の始点，b を e の終点という．頂点 v と辺 $e = (a, b)$ に対し，$v \in \{a, b\}$ ならば $v \in e$ と書く．$u, v \in V$ に対し，$\mathcal{P}(G, u, v)$ を G における u から v への有向パスの集合とする．G が有向閉路を持たないとき，G を有向非巡回グラフ (DAG) という．$E' \subseteq E$ に対し，G の E' による誘導部分グラフを $G[E'] = \langle \mathrm{dom}(E'), E' \rangle$ で定義する．ただし $E'' \subseteq E$ に対し $\mathrm{dom}(E'') = \{v \in V \mid \exists e \in E'', v \in e\}$ と定める．添字が i 以下，i 未満，i 以上，i より大きい辺の集合をそれぞれ $E^{\leq i}$, $E^{<i}$, $E^{\geq i}$, $E^{>i}$ と書く．G の辺 e による縮約 G/e とは，G において e を削除し，e の始点と終点を 1 つの頂点に併合したグラフのことをいう．$E' = \{e'_1, e'_2, \ldots, e'_{|E'|}\} \subseteq E$ に対し，G の E' による縮約を $G/E' = (\ldots((G/e'_1)/e'_2)/\ldots)/e'_{|E'|}$ で定める．G/E' が DAG であるとき，E' を G の非巡回縮約という．$E'' \subseteq E$ に対して $G' = \langle V', E' \rangle = G/E''$ とするとき，$u, v \in V$ が V' 上で同じ頂点に併合されているならば，u と v は連結であるという．また，V の V' 上での連結関係による同値類のことを連結成分という．

2.2 ゼロサプレス型二分決定グラフ (ZDD)

ゼロサプレス型二分決定グラフ (ZDD) [Minato 93] は，集合族を表現する DAG $Z = \langle N_Z, A_Z \rangle$ である．ここで，N_Z は ZDD ノードの集合，A_Z は有向枝の集合である *2．N_Z には終端ノード $\bot, \top$ がそれぞれ高々 1 つ存在する．終端ノード以外のノードは非終端ノードと呼ばれ，0 個以上存在する．どの非終端ノードも自身を始点とし他のノードを終点とする枝である 0-枝，1-枝を 1 本ずつ持ち，台集合の要素 1 つに対応するラベルを持つ．$x \in \{0, 1\}$ に対し，ノード α を始点とする x-枝の終点を α の x-子という．また，α のラベルを $l(\alpha)$ と書く．Z の非終端ノードの個数を Z のノード数という．

ZDD Z が表す集合族 $\mathcal{F}_Z$ は次のように定義される．$Z = \langle \{\bot\}, \emptyset \rangle$ のとき $\mathcal{F}_Z = \emptyset$, $Z = \langle \{\top\}, \emptyset \rangle$ のとき $\mathcal{F}_Z = \{\emptyset\}$ である．それ以外の場合，Z は $\bot$ と $\top$ をそれぞれ 1 つ，非終端ノードを 1 つ以上持つ．非終端ノードのうち入次数 0 のノードが高々 1 つ存在する．これを根ノードといい，r_Z と書く．正の整数 k に対し，Z の根ノードからノード α への有向パス $p = (n_1, a_1, n_2, a_2, \ldots, n_k, a_k, \alpha) \in \mathcal{P}(Z, r_Z, \alpha), n_1 = r_Z$ を考える．有向パス p に対し，集合 $S_p = \{l(n_i) \mid a_i \in A_{Z,1}, i \in [k]\}$ を対応付ける．ただし $A_{Z,1}$ は Z の 1-枝の集合とする．さらにノード α に対し集合族 $\mathcal{D}(\alpha) = \{S_p \mid p \in \mathcal{P}(Z, r_Z, \alpha)\}$ を対応付ける．このとき $\mathcal{F}_Z = \mathcal{D}(\top)$ である．同じ集合族を表す ZDD は複数存在しうるが，既約化規則 [Minato 93] を可能な限り適用した ZDD は一意に定まる．これを既約な ZDD という．

*2 混乱を避けるため，入力グラフの頂点と辺はそれぞれ「頂点」「辺」と呼び，ZDD の頂点と辺はそれぞれ「ノード」「枝」と呼ぶことにする．

3. アルゴリズム

本節では，グラフが与えられたとき，その非巡回縮約全体からなる族を表す ZDD を構築するアルゴリズムを提案する．提案手法はグラフの辺部分集合の族を表す ZDD を効率的に構築するアルゴリズムの枠組みであるフロンティア法に基づく．3.1 節ではフロンティア法の一般的な枠組みについて述べ，3.2, 3.3 節ではフロンティア法を我々の問題に適用するための提案手法の技法を述べる．

3.1 フロンティア法

提案手法が元とするフロンティア法の一般的な枠組みについて説明する．フロンティア法は，入力グラフ G に対して，所望の辺の集合族 $\mathcal{M} \subseteq \mathfrak{P}(E)$ を表す ZDD を効率的に構築するアルゴリズムの枠組みである．フロンティア法は ZDD を根ノードから順に幅優先的に構築していく．まず辺の順序を $e_1 < e_2 < \cdots < e_m$ と固定し，ラベル e_1 を持つノードを根ノードとして作成する．以降，各 $i = 1, 2, \ldots, m$ に対し第 i ステップの処理を行う．$\mathcal{N}_i$ をラベル e_i を持つノードの集合とすると，第 i ステップでは各 $\alpha \in \mathcal{N}_i$ と各 $x \in \{0, 1\}$ に対し α の x-子 α' を作成する．α' は終端ノードまたはラベル e_{i+1} を持つ非終端ノード（$i < m$ のとき）である．フロンティア法は，新たなノードを作成をするとき可能な限りノードの共有と枝刈りを行うことで，作成されるノードの個数を抑えながら ZDD の構築を行う．

まずノードの共有について説明する．ノード α の x-子としてノード α' を作成するとき，同じラベルを持つノード α'' であって α' と等価なものが既に作成されていれば，α' の作成をやめ，α の x-子を α'' とする．ここでラベル e_j を持つノード β と β' が等価であるとは，任意の $D_1 \in \mathcal{D}(\beta), D_2 \in \mathcal{D}(\beta'), H \subseteq E^{\geq j}$ に対し $D_1 \cup H \in \mathcal{M} \Leftrightarrow D_2 \cup H \in \mathcal{M}$ となることをいう．ここで等価性を効率的に判定するために，各ノードについて configuration と呼ばれるデータを管理する．configuration ϕ は，同じラベルを持つ任意のノード β, β' に対し，$\phi(\beta) = \phi(\beta')$ ならば β と β' が等価であるように管理する．

次に，枝刈りについて説明する．$j \in [m]$ と $D \subseteq E^{<j}$ について，任意の $H \subseteq E^{\geq j}$ に対し $D \cup H \notin \mathcal{M}$ が成り立つとき，D は $\mathcal{M}$ に拡張不可能であるという．ノード α の x-子としてノード α' を作成するとき，どの $D \in \mathcal{D}(\alpha')$ も $\mathcal{M}$ に拡張不可能であるとわかれば，α' の作成をやめ，α の x-子を $\bot$ とする．この処理を枝刈りという．枝刈りが可能であるかの判定は各ノードの configuration を用いて行われる．

フロンティア法に基づくアルゴリズムは，$\mathcal{M}$ に応じて適切

Algorithm 1: MakeNewNode(α, i, x)

α を α' にコピーする．
$\alpha' \leftarrow$ Init(α', i)
$\alpha' \leftarrow$ Update(α', i, x)
if IsBottom$(\alpha', i) =$ `true` **then return** $\bot$
if $i = m$ **then return** $\top$
$\alpha' \leftarrow$ Forget(α', i)
return α'

に configuration と MakeNewNode 関数を定義することで設計できる．MakeNewNode 関数は引数としてラベル e_i を持つ非終端ノード α, 整数 i, $x \in \{0, 1\}$ を受け取り，α の x-子となるべきノードを返す関数である．

いくつかの $\mathcal{M}$ に対し，configuration としてフロンティアと呼ばれる頂点部分集合に関する情報を管理すれば十分であることが知られている．$i \in [m-1]$ に対し，第 i フロンティアとは第 i ステップにおける処理済みの辺と未処理の辺の両方が接続している頂点の集合，すなわち $F_i = \{v \in V \mid \exists e' \in E^{<i}, v \in e'\} \cap \{u \in V \mid \exists e'' \in E^{\geq i}, u \in e''\}$ である．また，$F_0 = F_m = \emptyset$ と定義する．なお，フロンティア法によって構築された ZDD は既約とは限らない．既約な ZDD を得るには，フロンティア法によって構築された ZDD に既約化のアルゴリズム [Knuth 11, 7.1.4 節, Algorithm R] を適用すればよい．

3.2 configuration

提案手法は，ラベル e_i を持つ ZDD ノード α の configuration として，$\alpha.comp$ と $\alpha.R$ という 2 つの状態を管理する．$\alpha.comp$ は F_{i-1} の分割であり，処理済みの部分の縮約グラフにおけるフロンティア上の頂点間の連結関係を表す．すなわち，任意の $x, y \in F_{i-1}$, $D^{<i} \in \mathcal{D}(\alpha)$ に対し，x と y が $G[E^{<i}]/D^{<i}$ において連結であるとき，またそのときのみ x と y が $\alpha.comp$ の同じセルに属するように $\alpha.comp$ を管理する．一方 $\alpha.R$: $\alpha.comp \times \alpha.comp \to \{0, 1\}$ は，処理済みの部分の縮約グラフにおける連結成分間の推移関係を表す．すなわち，任意の $C_1, C_2 \in \alpha.comp$, $D^{<i} \in \mathcal{D}(\alpha)$ に対し，$\mathcal{P}(G[E^{<i}]/D^{<i}, C_1, C_2) \neq \emptyset$ であるとき $\alpha.R[C_1][C_2] = 1$，そうでないとき $\alpha.R[C_1][C_2] = 0$ となるように $\alpha.R$ を管理する．

図 2 (a), (b) にフロンティア法の途中過程（辺を e_1 から e_5 まで処理し終えた直後）の例を示す．図 2 (a), (b) において，太線は縮約することにした辺を，点線は縮約せず残すことにした辺を，実線は未処理の辺を表す．網掛けの領域の内部に存在する頂点の集合がフロンティアである．図 2 (a), (b) に対する configuration の値はともに図 2 (c) となり一致するため，これらの状態は共有される．実際，図 2 (a), (b) において太線の辺を縮約するとともに図 2 (d) のようになり，フロンティア上の状態が等価であることがわかる．

3.1 節で述べたように，フロンティア法ではまずラベル e_1 を持つ根ノードを作成する．根ノードを ρ とするとき，ρ の configuration は $\rho.comp$ を $\emptyset$，$\rho.R$ を空行列とする．

3.3 MakeNewNode 関数

提案手法の MakeNewNode 関数を Algorithm 1 に示す．MakeNewNode 関数はまず，ノード α を，その configuration を含めて α' にコピーする．Init 関数はフロンティアに新たに現れる頂点についての情報を configuration において初期化する．IsBottom 関数は枝刈り可能とわかれば `true` を，そうでなければ `false` を返す関数である．IsBottom 関数が `true` を返した場合，MakeNewNode 関数は $\bot$ を返す．最後の辺 e_m を処理し終えるまで枝刈りされなかったならば，非巡回縮約が完成したので MakeNewNode 関数は $\top$ を返す．Forget 関数はフロンティアから去る頂点についての情報を configuration から削除する．以下，提案手法に固有の処理を多く含む Update 関数と IsBottom 関数について説明する．

Algorithm 2: Update(α', i, x)

if $x = 1$ **then**
  // 辺 e_i を縮約
  $C(uv) = C(u) \cup C(v)$ とする．
  $\alpha'.comp \leftarrow (\alpha'.comp \setminus \{C(u)\} \setminus \{C(v)\}) \cup \{C(uv)\}$
  $\alpha'.R$ から $C(u)$ と $C(v)$ に対応する行・列を削除し，$C(uv)$ に対応する行・列を挿入する．
  foreach $C_1, C_2 \in \alpha'.comp \setminus \{C(uv)\}$ *such that* $\alpha'.R[C_1][C(uv)] = 1$ **and** $\alpha'.R[C(uv)][C_2] = 1$ **and** $\alpha'.R[C_1][C_2] = 0$ **do**
    $\alpha'.R[C_1][C_2] \leftarrow 1$
else
  // 辺 e_i を縮約せず残す
  foreach $C_1, C_2 \in \alpha'.comp$ *such that* $(C_1 = C(u)$ **or** $\alpha'.R[C_1][C(u)] = 1)$ **and** $(C_2 = C(v)$ **or** $\alpha'.R[C(v)][C_2] = 1)$ **and** $\alpha'.R[C_1][C_2] = 0$ **do**
    $\alpha'.R[C_1][C_2] \leftarrow 1$
return α'

Algorithm 3: IsBottom(α', i)

if $\exists C \in \alpha'.comp, \alpha'.R[C][C] = 1$ **then**
  return `true` // 有向閉路が発生している
if $\exists x, y \in F_{i-1} \cup F_i$, $(\alpha'.R[C(x)][C(y)] = 1$ **and** $r[i][y][x] = 1)$ **then**
  return `true` // 今後必ず有向閉路が発生する
return `false`

Update 関数

Update 関数はノード α' と整数 $i \in [m]$, $x \in \{0, 1\}$ を引数として受け取る．この関数は辺 e_i を縮約する（$x = 1$ のとき）か縮約せず残す（$x = 0$ のとき）かに応じて α' の configuration を更新し，更新後の α' を返す．e_i を縮約する場合と縮約せず残す場合に分けて Update 関数の処理の概要を説明する．以後 $e_i = (u, v)$ とする．また，$w \in F_{i-1} \cup F_i$ に対し，$C(w)$ は $\alpha'.comp$ において w を含むセルとする．

e_i を縮約する場合（$x = 1$ のとき），u の属する連結成分と v の属する連結成分が併合されるため，$\alpha'.comp$ の更新を行う．具体的には，$\alpha'.comp$ から $C(u)$ と $C(v)$ を削除し，$C(uv) = C(u) \cup C(v)$ を挿入する．また，$\alpha'.R$ においても，$C(u), C(v)$ に対応する行・列を削除し，$C(uv)$ に対応する行・列を挿入する．$C(uv)$ に対応する行の値 $R[C(uv)][C], C \in \alpha'.comp$ は，$R[C(u)][C] = 1$ または $R[C(v)][C] = 1$ のとき $R[C(uv)][C] = 1$, そうでないとき $R[C(uv)][C] = 0$ と計算できる（列についても同様である）．さらに，$C(u)$ と $C(v)$ が併合されたことにより，今まで縮約グラフにおいて有向パスが存在しなかった連結成分対に対して，新たに有向パスが発生する場合がある．そのような連結成分対に対し，$\alpha'.R$ の値を更新する．

e_i を縮約せず残す場合（$x = 0$ のとき），$\alpha'.comp$ を更新する必要はないが，$\alpha'.R$ は更新される場合がある．例えば，縮約グラフにおいて $C(u)$ から $C(v)$ への e_i による有向パスが発生す

表 1: グリッドグラフと Bayesian Network に対する実験結果.

Graph Name	n	m	Time	Memory	# node 1	# node 2	# solution
grid6×6	36	60	0.29	5	69258	10240	4.36×10^{13}
grid7×7	49	84	3.20	33	628256	49152	9.18×10^{18}
grid8×8	64	112	40.20	277	5781450	229376	1.41×10^{25}
grid9×9	81	144	477.56	2621	53867892	1048576	1.59×10^{32}
grid10×10	100	180	5666.67	25511	507298474	4718592	1.32×10^{40}
HAILFINDER	55	66	< 0.01	3	450	288	5.19×10^{17}
HEPAR II	69	123	23.17	173	1350205	174447	3.78×10^{22}
WIN95PTS	75	112	21.54	414	1456555	69343	7.67×10^{25}
PATHFINDER	108	195	0.09	22	8822	1908	1.49×10^{49}

る．より一般には，$C_1, C_2 \in \alpha'.comp$ に対して，C_1 から $C(u)$ への有向パスが存在し，かつ $C(v)$ から C_2 への有向パスが存在すれば $C_1 \dashrightarrow C(u) \dashrightarrow C(v) \dashrightarrow C_2$ なる有向パスが発生する．このようなすべての C_1, C_2 に対し $\alpha'.R[C_1][C_2] \leftarrow 1$ と更新する．Update 関数の擬似コードを Algorithm 2 に示す．

IsBottom 関数

IsBottom 関数は引数としてノード α' と整数 $i \in [m]$ を受け取り，どの $D^{\leq i} \in \mathcal{D}(\alpha')$ も非巡回縮約に拡張不可能であるとわかれば `true` を，そうでなければ `false` を返す．$D^{\leq i}$ が非巡回縮約に拡張不可能であるとわかるのは，まず，縮約グラフに有向閉路が発生した場合である．さらに，ある状況においては「現在有向閉路が発生していなくとも，今後必ず発生する」と断定することができ，その時点で枝刈りが可能である．枝刈り可能な状況を示すのが次の補題である（紙面の都合上，証明は省略する）．

補題 1 $D^{\leq i} \subseteq E^{\leq i}$ とする．ある 2 頂点 $x, y \in V$ が存在して $\mathcal{P}(G[E^{\leq i}]/D^{\leq i}, x', y') \neq \emptyset$ かつ $\mathcal{P}(G[E^{>i}], y, x) \neq \emptyset$ ならば，$D^{\leq i}$ は非巡回縮約に拡張不可能である．ただし，x', y' は $G[E^{\leq i}]/D^{\leq i}$ において x, y が縮約された頂点（あるいは自分自身）とする．

補題 1 に基づく枝刈りを実現するために，前処理として各 $i \in [m]$ に対し，$G[E^{>i}]$ における推移閉包 $r\colon [m] \times \mathrm{dom}(E^{>i}) \times \mathrm{dom}(E^{>i}) \rightarrow \{0, 1\}$ を求めておく．$i \in [m]$ と $x, y \in \mathrm{dom}(E^{>i})$ に対し，$\mathcal{P}(G[E^{>i}], x, y) \neq \emptyset$ のとき $r[i][x][y] = 1$，そうでないとき $r[i][x][y] = 0$ と定める．IsBottom 関数の擬似コードを Algorithm 3 に示す．

4. 実験

計算機実験により提案手法の性能を評価する．用いたグラフはグリッドグラフと Bayesian Network の 2 種類である．グリッドグラフは 6×6 から 10×10 の 5 つを用い，各辺の向きは，グリッドグラフを行列とみなしたとき，行番号または列番号の増える方向に向き付けた．Bayesian Network のグラフは `bnlearn` の Bayesian Network Repository から選んだ *3．用いた計算機は Intel Xeon Processor E7-8870 (2.4GHz) CPU と 2 TB のメモリを有し，OS は Oracle Linux 6.7 である．すべてのソースコードを C++ 言語と `TdZdd` ライブラリ *4 を用いて実装し，g++4.7.7 で `-O3` 最適化オプションつきでコンパイルした．

表 1 に実験結果を示す．表にはグラフの名前，頂点数，辺数，提案手法の計算時間（秒），消費メモリ（MB），構築された ZDD の既約化前のノード数，既約化後のノード数，得られた解（非巡回縮約）の個数を示している．グリッドグラフの結果を見ると，グラフのサイズが大きくなるほど，既約化後のノード数に対して既約化前のノード数が大きくなっている．そのため，既約化前のノード数をより小さくするような新たな枝刈りが設計できればより大きなグラフを扱えるようになる可能性がある．一方 Bayesian Network の結果では，グラフのサイズと計算時間等の相関があまり見られなかった．特に PATHFINDER は grid10×10 とほぼ同じ頂点数・辺数を有するにもかかわらず，grid10×10 に比べてかなり高速に ZDD を構築できており，解の圧縮効率もよい．このことは，入力グラフの構造によっては，グラフのサイズがある程度大きくなっても提案手法が効果的であることを示している．

*3 http://www.bnlearn.com/bnrepository/
*4 https://github.com/kunisura/TdZdd/

5. おわりに

本稿では，DAG からいくつかの辺を選んで縮約するとき，その結果が DAG となるような辺の選び方をフロンティア法により列挙する手法を提案した．計算機実験により，提案手法は DAG に含まれる 10^{40} 通り以上の辺の選び方の集合を表す ZDD を現実的な時間と消費メモリで構築できることを示した．今後の課題として，既約化前の ZDD のノード数を更に小さくするような，より効率的なアルゴリズムの設計が挙げられる．また，本手法のプログラム解析や確率推論への応用についても検討していきたい．

参考文献

[Galuzzi 11] Galuzzi, C. and Bertels, K.: The Instruction-Set Extension Problem: A Survey, *ACM Trans. Reconfigurable Technol. Syst.*, Vol. 4, No. 2, pp. 18:1–18:28 (2011)

[Kawahara 17] Kawahara, J., Inoue, T., Iwashita, H., and Minato, S.: Frontier-based search for enumerating all constrained subgraphs with compressed representation, *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 100, No. 9, pp. 1773–1784 (2017)

[Knuth 11] Knuth, D. E.: *The art of computer programming, Vol. 4A, Combinatorial algorithms, Part 1*, Addison-Wesley (2011)

[Minato 93] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, in *Proc. of the 30th ACM/IEEE design automation conference*, pp. 272–277 (1993)