2E5-OS-09b-4

二分決定グラフに基づく大規模ハイパーグラフの双対化とその応用

Dualizing Large-scale Hypergraphs with Binary Decision Diagrams and Its Applications

戸田貴久[*1*2] 湊真一[*2*1]
Takahisa Toda Shin-ichi Minato

[*1]科学技術振興機構 ERATO 湊離散構造処理系プロジェクト
ERATO MINATO Discrete Structure Manipulation System Project, Japan Science and Technology Agency

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

Dualizing hypergraphs is a fundamental computation on set families, and hence there are many applications in data mining, logic, artificial intelligence, etc. We present a new efficient algorithm using the compressed data structures BDDs and ZDDs, and we analyze the time complexity. By conducting computational experiments with various problem instances, we show that our algorithm is highly competitive with existing algorithms.

1. はじめに

ハイパーグラフ双対化は, 集合の集まりに関する基本的な計算問題であり, データマイニング, 論理, 人工知能をはじめとする幅広い研究分野に応用があることが知られている. 古くから理論と応用の両面から活発に研究されている. 詳細はサーベイ論文 [Eiter 02] [Eiter 08] を参照されたい.

はじめに, 本研究で扱う問題を定義するために必要な概念を導入する. **ハイパーグラフ**とは台集合上の集合の集まり（集合族）であり, 台集合 V と集合族 $\mathcal{E}$ の対 $(V,\mathcal{E})$ として与えられる. $\mathcal{E}$ の元は V の部分集合であり, **ハイパーエッジ**と呼ばれる. $\mathcal{E}$ の**ヒッティング集合**とは, V の部分集合であり, すべてのハイパーエッジに”当たる”ものである：すなわち, すべての $U \in \mathcal{E}$ に対して $U \cap T \neq \emptyset$ を満たす集合 T である. ヒッティング集合を包含する集合は明らかにヒッティング集合であるが, ヒッティング集合の部分集合はヒッティング集合とは限らない. ヒッティング集合が他のヒッティング集合を包含しないとき, **極小ヒッティング集合**と呼ぶ. $(V,\mathcal{E})$ の**双対ハイパーグラフ**[*1]は, 台集合が V であり, $\mathcal{E}$ の（すべての）極小ヒッティング集合をハイパーエッジとしてもつハイパーグラフである. **ハイパーグラフ双対化**とは, ハイパーグラフが与えられるとき, その極小ヒッティング集合をすべて列挙することにより双対ハイパーグラフを計算する問題である.

例えば, ハイパーグラフ $(V,\mathcal{E})$ が $V = \{1,\ldots,10\}$, $\mathcal{E} = \{\{9\},\{7,8\},\{2,4,7\},\{9,10\}\}$ のとき, 極小ヒッティング集合は $T_1 = \{7,9\}, T_2 = \{2,8,9\}, T_3 = \{4,8,9\}$ の 3 つであり, したがって双対ハイパーグラフは $(V,\{T_1,T_2,T_3\})$ である.

ハイパーグラフ双対化は**論理関数の双対化**と密接な関係がある. 論理関数の双対化とは, 論理関数 f の論理積形が与えられるとき, その**双対論理関数** f^d の主論理積形を計算する問題である. ここで双対論理関数とは $f^d(x_1,\ldots,x_n) := \overline{f}(\overline{x_1},\ldots,\overline{x_n})$, すなわち f^d が入力ベクトル $(v_1,\ldots,v_n)$ で 1 を返すときは, f が 0 と 1 を反転させた入力ベクトル $(v_1 \oplus 1,\ldots,v_n \oplus 1)$ で 0 を返すときとして定義される. 論理関数の双対化は計算困難な問題として知られているので, 一般の論理関数ではなく単調論理関数[*2]に制限した問題などがよく研究されている（例えば [茨木 94]）. 一般に, ハイパーグラフ双対化と単調論理関数の双対化は等価な問題であることが知られている（例えば [Eiter 02] の定理 2）. 実際, 単調論理関数は否定記号を用いない論理積形で記述可能な論理関数に対応するので, 上の例のハイパーグラフ $(V,\mathcal{E})$ を以下のように与えられる単調論理関数として記述できる.

$$x_9 \wedge (x_7 \vee x_8) \wedge (x_2 \vee x_4 \vee x_7) \wedge (x_9 \vee x_{10})$$

この双対はド・モルガンの法則により以下の論理和形

$$x_9 \vee (x_7 \wedge x_8) \vee (x_2 \wedge x_4 \wedge x_7) \vee (x_9 \wedge x_{10})$$

で表される. 計算により, 以下の主論理積形が求まる.

$$(x_7 \vee x_9) \wedge (x_2 \vee x_8 \vee x_9) \wedge (x_4 \vee x_8 \vee x_9)$$

これは $(V,\mathcal{E})$ の双対ハイパーグラフと対応している.

単調論理関数の双対化（したがって, ハイパーグラフ双対化）の計算量は大きな注目を集めてきたが, 1996 年に Fredman と Khachiyan により提案されたアルゴリズムを用いて $N^{o(\log N)}$ 時間で計算できることが分かり大きな進展がみられた（例えば [Eiter 08]）. ただし, N は入力サイズと出力サイズの和を表す. 出力サイズは入力サイズに関して指数的に増加することがありうるので, 出力サイズも考慮されていることに注意されたい. これは現時点で理論的に最速な結果である. **出力多項式時間**（すなわち, 入力サイズと出力サイズに関して多項式時間）のアルゴリズムが存在するかは未解決のままである.

一方, 2000 年代に入って実際的に効率の良いアルゴリズムが盛んに開発されている. 特に 2013 年, 村上と宇野は探索に基づく二つのアルゴリズムを提案し, 主要な既存アルゴリズムよりも性能が良いことを実験的に示した. 詳細に関しては [Murakami 13a] およびその中に含まれる参考文献を参照されたい.

連絡先: 戸田貴久 科学技術振興機構 ERATO 湊離散構造処理系プロジェクト 〒 060-0814 札幌市北区北 14 条西 9 丁目 Tel: 011-728-8280 Fax: 011-728-8277 E-mail: toda@erato.ist.hokudai.ac.jp

*1 **横断ハイパーグラフ**とも呼ばれる. 論理関数の双対化と密接な関係があるため双対ハイパーグラフと呼ばれるが, 双対は別の意味で定義されることがあるので混同されないよう注意されたい.

*2 論理関数 f が**単調**とは, 任意の入力ベクトル u,v に対して $u \leq v$ ならば $f(u) \leq f(v)$ を満たすものとして定義される.

本研究では, ハイパーグラフ双対化のための新しいアルゴリズムを提案する. 本アルゴリズムでは, 効率的な圧縮技法として広く認められているデータ構造 BDD と ZDD を用いる. これらのデータ構造の特徴は, データを圧縮して表現できること, 様々な基本操作を効率的に実行できることなどである. 圧縮データ構造に基づくハイパーグラフ双対化の計算法は, Coudert によって示唆されている [Coudert 97] が, 具体的に与えたのはおそらく Knuth が最初である. ただ, 教科書 [Knuth 11] の多くの練習問題の中に埋もれているので広く知られていないように思われる. Knuth アルゴリズムは ZDD だけを用いるのに対して, 提案アルゴリズムは BDD と ZDD の両方を用いるので, 提案アルゴリズムは Knuth アルゴリズムの変種とみなされる. 圧縮データ構造に基づく計算法の性能評価はこれまで実施されていないようなので, 本稿では提案アルゴリズム, Knuth アルゴリズム, 村上・宇野アルゴリズムの比較実験を行う. 既存研究で使われてきた問題例だけでなく, しきい値関数の論理積形に対応する問題例などを作成し実験に用いた.

以下の本文では, 2 節で圧縮データ構造 BDD と ZDD を導入する. 3 節で提案アルゴリズムを解説し, 計算量の解析も行う. 4 節で実験結果を示す. 5 節で本研究をまとめる.

2. 集合族のための圧縮データ構造

本節では集合族のための圧縮データ構造 BDD と ZDD について解説する. 詳細は [Knuth 11] の §7.1.4 を参照されたい.

2.1 BDD

BDD (Binary Decision Diagram) は論理関数のグラフ表現である. BDD の利点は, 多くの有用な論理関数を小さなサイズに圧縮できること, 論理関数に対するさまざまな操作を BDD 上で効率的に実行できることなどである. BDD は, 一方で, 集合族のデータ構造としても見られる. なぜなら, 論理関数 $f(x_1,\ldots,x_n)$ はその解集合 $\{v\in\{0,1\}^n\colon f(v)=1\}$ に対応し, 解 $(v_1,\ldots,v_n)\in\{0,1\}^n$ は添数集合 $\{i\colon v_i=1\}$ に対応するからである.

図 1 に BDD の例を示す. 最上段の節点は**根**と呼ばれる. 内部節点は $\mathrm{V},\mathrm{LO},\mathrm{HI}$ の三つのデータからなる. V には変数の添数, LO と HI には他の節点へのポインタが格納される. LO と HI の指す節点は, それぞれ LO **節点**, HI **節点**と呼ばれる. LO 節点への有向枝は LO **枝**と呼ばれ, 点線矢印で表される. 同様に, HI 節点への有向枝は HI **枝**と呼ばれ, 実線矢印で表される. 二つの終端節点 $\bot,\top$ がある. BDD と ZDD の終端節点を区別するため, 前者を $\bot_{\mathrm{BDD}},\top_{\mathrm{BDD}}$, 後者を $\bot_{\mathrm{ZDD}},\top_{\mathrm{ZDD}}$ と表すことがある.

以下の二条件を満たすものを BDD と呼ぶ. 第一に, 内部節点 u が v を差すとき, $\mathrm{V}(u)<\mathrm{V}(v)$ が満たされなければならない. 第二に, BDD は**既約**でなければならない : すなわち, 以下のいずれの簡約規則も適用できない.

1. 節点 u の二つの枝が同一の節点 v を指すならば, u を指す全ての枝を v を指すように変え, u を削除する（図 2(a)）.
2. もし節点 u と v を根とする部分グラフ同士が等価ならば, 部分グラフを共有する（図 2(b)）.

BDD を集合族の表現とみなすとき, BDD は以下のように理解されうる. BDD の根から終端節点 $\top$ に至る道は BDD が保持する集合を表す. 例えば, 図 1 において道 ① → ② ⇢ ③ ⇢ $\top$ と ① → ② → $\bot$ は, BDD は $\{1\}$ を保持するが, $\{1,2\}$ も $\{1,2,3\}$ も保持しないことを意味する. ここで, 後者の道には節点 ③ が明示的に現れていないが, 節点削除規則によれば二つの枝はともに $\bot$ を指していることに注意されたい. $\bot_{\mathrm{BDD}}$ だけからなるグラフは BDD であり, 集合族として見るとき（$\top$ に至る道がないので）$\emptyset$ に対応する. 一方, $\top_{\mathrm{BDD}}$ は, 同一の子を指す節点の列が削除されているので, 冪集合 2^V（ただし V は台集合とする）に対応する.

任意の論理関数 $f(x_1,\ldots,x_n)$ は BDD として一意の表現を持つ (例えば [Knuth 11]) ので, BDD を集合族の表現とみなすとき, 添数集合 $\{1,\ldots,n\}$ 上の集合族と BDD とは一対一に対応する. 効率性のため, 各 BDD 節点 p はそれが保持する三つ組 $(\mathrm{V}(p),\mathrm{LO}(p),\mathrm{HI}(p))$ に対して一意になるようにハッシュ表で管理されている. すなわち, 関数 BDD_UNIQUE は, 添数, LO 節点, HI 節点の組 (k,l,h) を受け取り, 対応する節点がハッシュ表に登録されているときその節点を返す. 登録されていないとき, $\mathrm{V}(p)=k,\mathrm{LO}(p)=l,\mathrm{HI}(p)=h$ を満たす節点 p を作成してハッシュ表に登録して p を返す. これにより等価なグラフを表す異なる節点が作成されなくなる上, グラフの等価性判定を定数時間のうちに行えるようになる.

BDD 上の任意の節点に対して, それを根とする部分グラフは BDD である. 節点を BDD と呼ぶときは, その節点を根とする BDD を意味する. BDD 上の節点を**部分** *BDD* と呼ぶ.

論理関数 f の BDD を $B(f)$ で表す. $B(f)$ の**サイズ**とは, $B(f)$ における終端節点を含む節点の総数であり, $|B(f)|$ で表す. BDD 演算 $\mathrm{AND}(B(f),B(g)):=B(f\wedge g)$ は $|B(f)|\cdot|B(g)|$ に比例する時間内に計算できる. BDD を集合族の表現として見るとき, AND 演算は共通部分の計算に対応する.

2.2 ZDD

台集合のサイズに比べて小さいサイズの集合ばかりからなる集合族を BDD で表すとき, HI 枝が $\bot$ を指す節点がたくさん現れるようになる. *ZDD (Zero-suppressed binary Decision Diagram)* はこのような疎な集合族に特化した BDD の進化形である [Minato 93]. 具体的には, ZDD は BDD の順序に関する条件を満たし, 以下の簡約規則に関して既約であるものとして定義される.

1. 節点 u の HI 枝が $\bot$ を指すならば, u を指す全ての枝を u の LO 節点を指すように変え, u を削除する（図 3）.
2. もし節点 u と v を根とする部分グラフ同士が等価ならば, 部分グラフを共有する（図 2(b)）.

共有規則は BDD と同じだが, 削除規則が異なる（したがって, BDD の節点削除規則を満たす必要はない）.

図 4 に ZDD の例を示す. ZDD の根から $\top$ に至る道が ZDD の保持する集合に対応する. 例えば, 図 4 の道 ① → $\top$ と ① ⇢ ② ⇢ ③ ⇢ $\top$ は $\{1\}$ と $\emptyset$ にそれぞれ対応する. 図 4 の ZDD は図 1 の BDD と同じ集合族 $\{\emptyset,\{1\},\{2\},\{3\}\}$ を表す. $\bot_{\mathrm{ZDD}}$ だけからなるグラフは ZDD であり, $\emptyset$ を表す. $\top_{\mathrm{ZDD}}$ もまた ZDD であり, HI 枝が $\bot_{\mathrm{ZDD}}$ を指す節点の列が削除されているので $\{\emptyset\}$ に対応する. よって, $\top_{\mathrm{ZDD}}$ は $\top_{\mathrm{BDD}}$ と異なる集合族を表す.

BDD の場合と同様に, 添数集合 $\{1,\ldots,n\}$ 上で集合族と ZDD とは一対一に対応する [Minato 93]. ZDD 節点もハッシュ表で管理されており, 関数 ZDD_UNIQUE を通して操作される. また, ZDD 上の任意の節点（を根とする部分グラフ）は ZDD であり, **部分** *ZDD* と呼ぶ. 集合族 $\mathcal{F}$ の ZDD を $Z(\mathcal{F})$ で表す. $Z(\mathcal{F})$ の**サイズ**は終端節点を含む節点の総数を意味し, $|Z(\mathcal{F})|$ で表す. ZDD 演算 $\mathrm{DIFF}(Z(\mathcal{U}),Z(\mathcal{V})):=Z(\mathcal{U}\setminus\mathcal{V})$ は $|Z(\mathcal{U})|\cdot|Z(\mathcal{V})|$ に比例する時間内に計算できる.

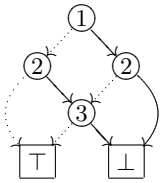
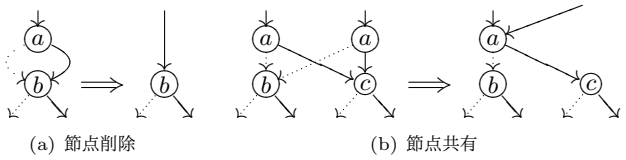


図 1: BDD



(a) 節点削除　(b) 節点共有

図 2: BDD の簡約規則

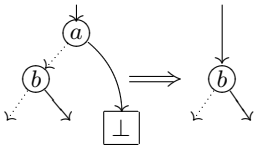


図 3: ZDD の節点削除

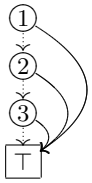


図 4: ZDD

3. アルゴリズム

アルゴリズムは以下の四つの部分からなる. $\mathcal{S}(x)$ は BDD あるいは ZDD x に対応する集合族を表す. 集合族の台集合は $\{1,\ldots,n\}$ であり, 各集合は昇順に整列されていると仮定する.

1. 入力の集合族 $\mathcal{E}$ を圧縮して, 対応する ZDD p を求める.
2. $\mathcal{S}(p)$ のヒッティング集合全体からなる BDD q を計算する.
3. $\mathcal{S}(q)$ における極小集合全体からなる ZDD r を計算する.
4. ZDD r を解凍して, 対応する集合族 $\mathcal{E}^*$ を出力する.

ZDD を解凍するとき莫大な数の集合が生成されることがあるので, 実際には解凍まで行わないで ZDD を出力する方が良い. ただ, 既存アルゴリズムとの比較実験を同一の条件下で行うため, 本研究では解凍まで行う.

ZDD の圧縮には [Toda 13a] で提案されているアルゴリズムを採用する. アルゴリズムの詳細は紙面の都合上割愛する. ZDD の解凍は以下のようになされる. ZDD の根から終端節点までの道は ZDD が保持する集合に対応するので, すべての道をたどりながら対応する集合を出力すれば良い. 道の長さは台集合 V のサイズ以下なので, 解凍に要する時間は $O(|V|\cdot|\mathcal{E}^*|)$ である. ただし, $\mathcal{E}^*$ は出力される集合族を表す.

提案アルゴリズムの二番目の部分はアルゴリズム 1 の関数 HIT によって計算される. この計算の正当性を入力 ZDD p の構造に関する帰納法で示す. p が終端節点の場合はただちに確認できるので, p は内部節点として $pl := \mathrm{LO}(p), ph := \mathrm{HI}(p)$ と表す. まず $\mathcal{S}(p)$ は $\mathcal{S}(pl)$ と $\{\{\mathrm{V}(p)\}\cup U : U\in\mathcal{S}(ph)\}$ の直和である. よって, 集合 T が $\mathcal{S}(p)$ のヒッティング集合であるためには, (1) T が $\mathcal{S}(pl)$ のヒッティング集合であり, かつ (2) $\mathrm{V}(p)$ が T に属すか, T は $\mathcal{S}(ph)$ のヒッティング集合であることが必要十分である. 帰納法の仮定より, 条件 (1) を満たす集合は $\mathrm{HIT}(pl)$ によって列挙され, 条件 (2) を満たす集合は $\mathrm{BDD\_UNIQUE}(\mathrm{V}(p), \mathrm{HIT}(ph), \bot_{\mathrm{BDD}})$ によって列挙される. したがって, 両条件を満たす集合, すなわち $\mathcal{S}(p)$ のヒッティング集合は $\mathrm{HIT}(p)$ によって列挙される.

定理 1. アルゴリズム *1* は $O(|p|\cdot N(p)^2)$ 時間で計算できる. ここで $N(p) := \max\{|\mathrm{HIT}(p')| : p' \text{ は } p \text{ の部分 } ZDD\}$.

Proof. 部分 ZDD p' に対する出力 BDD $\mathrm{HIT}(p')$ を記憶するためにハッシュ表を用いる. これにより各 p' に対して $\mathrm{HIT}(p')$ の計算はちょうど一回だけしか行われない. 関数 BDD_UNIQUE は定数時間で計算される. $|t| < |hh|+2$ より, $\mathrm{AND}(hl,t)$ の計算時間は $O(|hl|\cdot|hh|)$, したがって $O(N(p)^2)$ である. ゆえに, $\mathrm{HIT}(p)$ の計算に要する時間は $O(|p|\cdot N(p)^2)$ である. □

アルゴリズムの三番目の部分では, アルゴリズム 1 の出力 BDD q から極小集合を抽出する. この計算を理解するために, このような極小集合は論理関数の観点からは f_q の主項に対応することを概説する. ここで f_q は q に対応する論理関数を表す. まず $\mathcal{S}(q)$ は, **上方に閉じている**, すなわち任意の $U\in\mathcal{S}(q)$ に対して $U\subseteq U'$ ならば $U'\in\mathcal{S}(q)$ を満たす. これから, f_q が単調であること, すなわち $u\le v$ ならば $f_q(u)\le f_q(v)$ を満たすことが導かれる. 以下の事実が知られている（例えば [Knuth 11] の 54–55 項）: 単調論理関数 h の極小解 S と h の主項 g とは $g=\wedge_{i\in S}x_i$ を通して一対一に対応する. ゆえに, $\mathcal{S}(q)$ の極小集合は f_q の主項とみなされる. [Knuth 11] の 256 項に記述されている, 単調論理関数のすべての主項を表現する ZDD の再帰式 (137) は, $\mathcal{S}(q)$ の極小集合全体を表現するものとしてもみなされるので, アルゴリズム 2 がただちに得られる.

Algorithm 1 ZDD p が与えられて, $\mathcal{S}(p)$ のヒッティング集合全体からなる BDD q を計算する.

```
function HIT(p)
    if p = ⊤_ZDD then
        return ⊥_BDD;
    end if
    if p = ⊥_ZDD then
        return ⊤_BDD;
    end if
    hl ← HIT(LO(p)); hh ← HIT(HI(p));
    t ← BDD_UNIQUE(V(p), hh, ⊤_BDD);
    q ← AND(hl, t);
    return q;
end function
```

Algorithm 2 BDD q で $\mathcal{S}(q)$ が上方に閉じたものが与えられて, $\mathcal{S}(q)$ における極小集合全体からなる ZDD r を計算する.

```
function MIN(q)
    if q = ⊥_BDD then
        return ⊥_ZDD;
    end if
    if q = ⊤_BDD then
        return ⊤_ZDD;
    end if
    mh ← MIN(HI(q)); ml ← MIN(LO(q));
    t ← DIFF(mh, ml);
    r ← ZDD_UNIQUE(V(q), ml, t);
    return r;
end function
```

アルゴリズム 2 によって計算される関数を MIN と表す. 以下は定理 1 と同様の方法で証明されうるので, 証明を省く.

定理 2. アルゴリズム *2* は $O(|q|\cdot L(q)^2)$ 時間で計算できる. ここで $L(q) := \max\{|\mathrm{MIN}(q')| : q' \text{ は } q \text{ の部分 } BDD\}$.

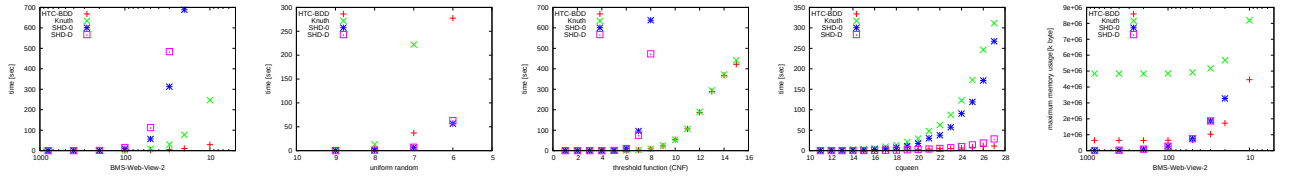


図 5: 実行時間の比較（点の水平成分は問題例のパラメータを表す.）

図 6: 最大メモリ使用量

4. 実験

実装と環境. 提案アルゴリズムと Knuth アルゴリズムを C 言語で実装した（HTC-BDD 1.0.0 版）. このプログラムは [Toda 13b] で公開されている. 本実験では, 湊により開発された BDD パッケージ（BDD Package SAPPORO-Edition-1.0）を利用した. 村上・宇野アルゴリズムの実装 SHD 3.1 版を [Murakami 13b] から入手した. 使用した計算機環境は, 2.67GHz Xeon®E7-8837, 1.5TB RAM, SUSE Linux Enterprise Server 11, gcc version 4.3.4 である.

問題例. bms(n) はデータマイニングの現実のデータセット”BMS-WebView-2” の極大頻出アイテム集合の補集合をハイパーエッジとするハイパーグラフを表し, rand(n) はランダムに生成されたハイパーグラフを表す（n はパラメータ）. これらは [Murakami 13a] の実験で用いられた. 一方, 今回以下の問題例を作成した：TH(n) は, n をしきい値とする 30 変数のしきい値関数, すなわち n 個以上の引数が 1 のとき 1 を返す論理関数の主論理積形に対応するハイパーグラフである. cqueen(n) は n クイーングラフの補グラフである.

実行時間. 提案アルゴリズム, Knuth アルゴリズム, 村上・宇野アルゴリズム (SHD-0, SHD-D) の実行時間を比較した. 図 5 に実験結果を示す. 1000 秒の時間制限を設け, それを超過したときプログラムを強制的に停止した. そのような問題例は図にプロットされていない. 提案アルゴリズムは多くの問題例で最速であったが, rand(6) では村上・宇野アルゴリズムに比べ約 4 倍ほど遅かった（他の実験結果は [Toda 13c]）.

最大メモリ使用量. 提案アルゴリズムと Knuth アルゴリズムは多数の節点をハッシュ表で管理するので, メモリ使用量が急増する傾向がある. それに比べ, 村上・宇野アルゴリズムは非常に安定的であった. ただ, 問題例 bms(n) では提案アルゴリズムとの間に顕著な差は見られなかった（図 6）.

5. まとめ

本研究ではハイパーグラフ双対化の新しいアルゴリズムを提案し, Knuth アルゴリズム, 村上・宇野アルゴリズムと実行時間と最大メモリ使用量を比較した. 提案アルゴリズムは多くの問題例において最速であったが, 一部の問題例において村上・宇野アルゴリズムより遅いことがあった. 圧縮データ構造を用いる計算法は集合族を保持する必要があるので, 探索に基づく村上・宇野アルゴリズムにくらべて多くのメモリを必要とする. 今後の課題として, 本アルゴリズムの並列化, 本アルゴリズムが高速に動作する問題例の特徴の明確化, 一般の論理関数の双対化（例えば [Yamamoto 12]）への応用などがある.

参考文献

[Coudert 97] Coudert, O.: Solving Graph Optimization Problems with ZBDDs, in *the 1997 European Conference on Design and Test (EDTC97)*, pp. 224–228 (1997)

[Eiter 02] Eiter, T. and Gottlob, G.: Hypergraph Transversal Computation and Related Problems in Logic and AI, *LNAI*, Vol. 2424, pp. 549–564 (2002)

[Eiter 08] Eiter, T., Makino, K., and Gottlob, G.: Computational aspects of monotone dualization: A brief survey, *Discrete Applied Mathematics*, Vol. 156, pp. 2035–2049 (2008)

[Knuth 11] Knuth, D.: *The Art of Computer Programming Volume 4a*, Addison-Wesley Professional (2011)

[Minato 93] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, in *30th ACM/IEEE Design Automation Conference (DAC-93)*, pp. 272–277 (1993)

[Murakami 13a] Murakami, K. and Uno, T.: Efficient Algorithms for Dualizing Large-Scale Hypergraphs, in *Meeting on Algorithm Engineering & Experiments (ALENEX13)*, pp. 1–13 (2013)

[Murakami 13b] Murakami, K. and Uno, T.: Hypergraph Dualization Repository, `http://research.nii.ac.jp/~uno/dualization.html` (2013), accessed on 19 January

[Toda 13a] Toda, T.: Fast construction of ZDDs from large-scale hypergraphs, Technical Report A-13-62, Division of Computer Science, Hokkaido University (2013)

[Toda 13b] Toda, T.: HTC-BDD: Hypergraph Transversal Computation with Binary Decision Diagrams, `http://kuma-san.net/htcbdd.html` (2013), accessed on 28 March

[Toda 13c] Toda, T.: Hypergraph Transversal Computation with Binary Decision Diagrams, in *12th International Symposium on Experimental Algorithms (SEA13)* (2013), accepted

[Yamamoto 12] Yamamoto Y., Iwanuma K., and Inoue, K.: Non-monotine dualization via Monotone Dualization, in *22nd International Conference on Inductive Logic Programming (ILP2012)* (2012)

[茨木 94] 茨木俊秀：正論理関数の同定問題とその複雑さ, 室田一雄（編）, 離散構造とアルゴリズム III, pp. 1–29, 近代科学社 (1994)