Vol.2013-AL-144 No.1
2013/5/17

大規模ハイパーグラフから ZDDの高速な構築アルゴリズム

戸田　貴久[1,a)]

概要：ハイパーグラフはグラフの一般化であり，幅広い種類の情報をモデル化できるので，計算機科学に多くの応用がある．ZDD はハイパーグラフを表現するための圧縮データ構造であり，ハイパーグラフを操作する実際的に効率の良いさまざまな演算が考案されている．これらの ZDD 演算を通して組合せ問題を計算する手法は，計算が困難な大規模問題に対して有効であることが報告されている．この手法において，ハイパーグラフを ZDD に効率的に圧縮する処理は重要である．本研究では，ハイパーグラフを ZDD に圧縮する高速アルゴリズムを提案し，計算量の解析を行う．さらに，様々な種類のデータセットを用いた比較実験により，既存手法よりも高速かつ省メモリに動作することを確認する．

キーワード：ハイパーグラフ, 二分決定グラフ, データ圧縮, 整列, データマイニング, 組合せ問題

Fast Construction of ZDDs from Large-scale Hypergraphs

Abstract: We present an algorithm to compress hypergraphs into the data structure ZDDs and analyze the computational complexity. Since a ZDD provides an approach to solve large-scale problems that are difficult to compute in a reasonable amount of time and space, it is important to compress hypergraphs efficiently. Our algorithm uses multikey Quicksort given by Bentley and Sedgewick. By conducting experiments with various datasets, we show that our algorithm is significantly faster and requires smaller memory than an existing method.

Keywords: hypergraph, binary decision diagram, compression, sort, data mining, combinatorial problem

1. はじめに

ハイパーグラフはグラフの一般化である．グラフでは二節点間の隣接関係が扱われるのに対して，ハイパーグラフでは複数節点間の関係が扱われる．したがって，ハイパーグラフは台集合 V と集合族（すなわち，V の部分集合の集まり）$\mathcal{E}$ との対 $(V, \mathcal{E})$ として定義される．ここで，$\mathcal{E}$ に属する集合は**ハイパーエッジ**と呼ばれる．ハイパーグラフは幅広い種類の情報を表現できるので，計算機科学に多くの応用がある．最近ではハイパーグラフとみなされうる大規模データが広く蓄積・流通されるようになり，大規模ハイパーグラフの効率的な処理はますます重要になってきている．

ゼロサプレス型二分決定グラフ(ZDD) はハイパーグラフを表現する圧縮データ構造である [1]．一般に圧縮効率はハイパーグラフに依存するが，ZDD は疎なハイパーグラフ[*1]に対して有効であるように設計されている．疎なハイパーグラフは取引記録（トランザクション・データ）など現実のデータとしてよく現れる．ZDD の利点は圧縮効率だけではない．ZDD の最大の特徴は，明示的に解凍することなしにハイパーグラフに対する様々な操作を実行できることである．圧縮がよく効いているときに ZDD 演算は威力を発揮するので，圧縮に基づかない通常の方法では計算が困難な大規模問題を現実的な時間内に処理できることがしばしばある．したがって，ZDD に基づく計算法は大規模ハイパーグラフを処理するための実際的に有効な手法の一つとして数えられている．

[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
[a)] toda.takahisa@gmail.com

[*1] 台集合よりもかなり小さいサイズのハイパーエッジばかりで構成されたハイパーグラフ

ZDD および類似のデータ構造 BDD はさまざまな組合せ問題に応用されている．関根，今井，谷 [2] は，#P 困難な問題として知られるグラフの Tutte 多項式の計算に対して BDD に基づく計算法を提案した．グラフに関する様々な数え上げ問題は Tutte 多項式の計算に帰着されうるので，この手法は多くの応用を持つ．例えば，ネットワーク信頼度の厳密計算へ応用された [3]．Coudert [4] は，多くのグラフ組合せ問題から帰着されうる三つの基本問題に対して ZDD 演算を組み合わせた解法を示すことで，グラフ組合せ問題を解く一般的な枠組みを提案した．Knuth は有名な教科書「The Art of Computer Programming」 [5] の練習問題で，極小ヒッティング集合の列挙を ZDD 演算の組合せで解く方法を与えた．極小ヒッティング集合の列挙は，単調論理関数の双対化と等価な問題であり，人工知能，論理，データマイニングなどをはじめとする幅広い研究分野に多くの応用を持つことで知られている（例えば [6], [7], [8]）．近年この問題を計算する高速なアルゴリズムが盛んに開発されているが，戸田 [9] は ZDD だけでなく BDD も組み合わせて計算する手法を考案し，多くのデータセットにおいて既存手法より高速に動作することを実験的に示した．

近年，大規模データベースの解析に関する研究が盛んに行われている．湊と有村 [10] はそのようなデータベース解析問題に対して ZDD に基づく枠組みを提案した．データベースから頻出集合を列挙する問題は，頻出集合マイニングにおいて重要な役割を果たす [11]．湊，宇野，有村 [12] は，頻出集合を高速に列挙するアルゴリズム LCM [13] と連携させて，列挙結果を ZDD に圧縮して出力する手法 LCM over ZDDs を提案した．

本稿では，ハイパーグラフを ZDD に圧縮する高速アルゴリズムを提案する．既存の圧縮法は ZDD の和演算を繰り返してハイパーグラフを圧縮するものである（例えば，[9], [10], [14]）．ZDD の和演算は ZDD ライブラリで通常提供されているので実装が簡単な反面，入力によって性能が著しく低下することがある．また別の方法として，上述の LCM over ZDDs では，LCM によって逐次生成される頻出集合を受け取りながらボトムアップに ZDD を構築している．本手法はこの方法と基本的に同じ考え方に基づきボトムアップに ZDD を構築するが，本アルゴリズムではさらに**マルチキー・クイックソート**との連携により性能の向上をはかる．ここで，マルチキー・クイックソートは Bentley と Sedgewick により与えられた文字列の整列アルゴリズム [15] である．本稿では提案アルゴリズムの計算量を解析し，さらにさまざまなデータセットを用いた実験により，提案アルゴリズムは和演算に基づく圧縮法よりも高速かつ省メモリに動作することを示す．

以下の本文では，2 節で ZDD を導入する．3 節で提案アルゴリズムを解説し，計算量を解析する．4 節で実験結果を示す．5 節で本研究をまとめる．

2. ハイパーグラフを表現する圧縮データ構造

本稿ではハイパーグラフのための特別なデータ構造 ZDD を用いるので，本節で必要な概念と結果を導入する．混同の恐れがないときハイパーグラフと集合族を区別しない．

2.1 ゼロサプレス型二分決定グラフ

ゼロサプレス型二分決定グラフ (ZDD) はハイパーグラフのグラフ表現である．図 1 に ZDD 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 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 $\bot$ と $\top$ がある．

以下の二条件を満たすものを ZDD と呼ぶ．第一に，内部節点 u が v を指すとき，$\mathrm{V}(u) < \mathrm{V}(v)$ が満たされなければならない．第二に，以下のいづれの簡約規則も適用できないという意味で**既約**でなければならない．

(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 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)）．

ZDD は以下のように理解されうる．ZDD の根から $\top$ に至る道と ZDD の保持する集合とは，以下のように対応する．道上でラベル k の節点で HI 枝が選ばれるとき，k は集合に含まれ，そうでないときは含まれない．例えば，図 1 の道 ① ⇢ ② → ③ → $\top$ と ① → ② → $\top$ は $\{2,3\}$ と $\{1,2\}$ にそれぞれ対応する．ここで，後者の道に ③ が現れていないが，節点削除規則により，③ の HI 枝は $\bot$ を指すことが分かり，したがって 3 は集合に含まれない．

台集合 V 上の要素の順序が固定されているならば，V 上のハイパーグラフは ZDD として一意の形をもつことが知られている（例えば [1][5]）．効率性のため，各 ZDD 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 zdd_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．

2.2 既存の圧縮法

台集合 V 上の集合族 $\mathcal{E} := \{U_1, \ldots, U_m\}$ を ZDD に圧縮する既存アルゴリズムを解説する．各集合 U_i に対して，そ

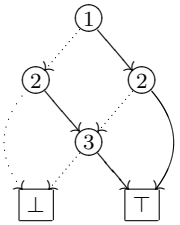
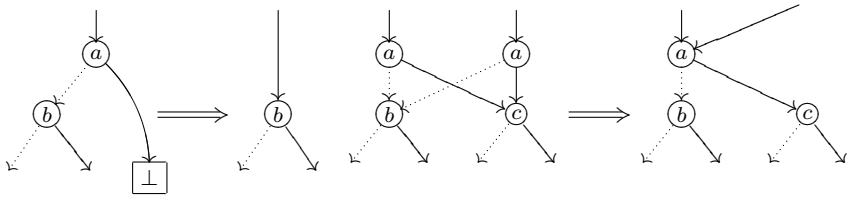


図 1 集合族 $\{\{2,3\},\{1,3\},\{1,2\}\}$ を表す ZDD



(a) 節点削除　(b) 節点共有

図 2 ZDD の簡約規則

れだけからなる集合族 $\{U_i\}$ を表す ZDD を構築する．それらの ZDD を和演算で足し合わせる．ここで，和演算は $Z(\mathcal{F})$ と $Z(\mathcal{G})$ を受け取り $Z(\mathcal{F}\cup\mathcal{G})$ を計算する．ただし，$Z(\mathcal{S})$ は集合族 $\mathcal{S}$ を表す ZDD とする．[9] において議論されているように，この構成法は $O(|\mathcal{E}|\cdot|V|)$ 時間を要する．

和演算は ZDD ライブラリにおいて提供されるので，上述の圧縮法を実装するのは容易であるが，その反面いくつかの欠点がある．

- 実際的な効率性は集合を加える順序に依存する．
- たとえ集合のサイズが台集合よりかなり小さい場合でも，最悪計算時間は台集合のサイズに依存する．
- 出力 ZDD に現れない ZDD 節点が計算途中に多数生成されることがある．

3. アルゴリズム

本節では，まず最初に ZDD はハイパーエッジの整列過程（を図式化したもの）とみなされることを観察し，その後で提案アルゴリズムを解説する．

図 3 に同型対応の例を示す．正整数からなる台集合 V 上のハイパーグラフ $(V,\mathcal{E})$ が与えられるとする．ここで各ハイパーエッジ $U\in\mathcal{E}$ は U の元を昇順に並べた配列として表現されている．したがって，U の d 番目の**値**というとき，U の d 番目に大きな数を意味する．簡単のため，すべてのハイパーエッジは正の無限大 $+\infty$ を持つこととする．このとき，以下の手続きに従い，配列上の位置 d における最小の値 v に関して，$\mathcal{E}$ を”等しい部分“$\mathcal{E}_=$ と“大きい部分”$\mathcal{E}_>$ に再帰的に分割しながら，辞書式順にすべてのハイパーエッジを整列させる．ここで，この手続きを最初に実行するときには $d=1$ が渡されるものとする．

function SORT($\mathcal{E}, d$)
  if $|\mathcal{E}|\le 1$ **then**
    return;
  end if
  $v\leftarrow\mathcal{E}$ に属する集合の d 番目の値のうちで最小の数;
  if $v=+\infty$ **then**
    return;
  end if
  $\mathcal{E}_=\leftarrow d$ 番目の値が v のハイパーエッジ全体;
  $\mathcal{E}_>\leftarrow\mathcal{E}\setminus\mathcal{E}_=$;
  sort($\mathcal{E}_=, d+1$); sort($\mathcal{E}_>, d$);
end function

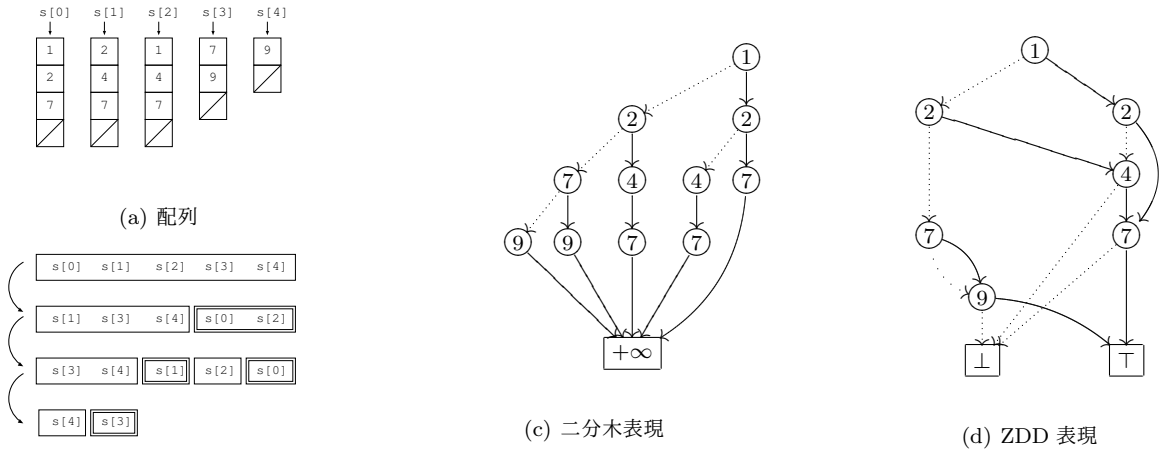
この整列過程を順序付きの二分木で表すとき，各節点は分割の値 v を保持し，節点の左の子と右の子はそれぞれ再帰呼び出し sort($\mathcal{E}_>, d$) と sort($\mathcal{E}_=, d+1$) に対応するものとして図式化される．この二分木は，左の子と右の子をそれぞれ LO 節点と HI 節点とみなすとき，ZDD の順序に関する条件を満たし，さらに，節点削除規則に関して既約な形である（事実，$v<+\infty$ ならば $\mathcal{E}_=\neq\emptyset$ であるから）．すべての再帰呼び出しが木の節点に正確に対応していることに注意されたい．この二分木の等価な部分グラフ同士を共有させることで，対応する ZDD ができあがる．この対応関係が全単射であることは容易に観察されるだろう．

この整列アルゴリズムは Bentley と Sedgewick により与えられた**マルチキー・クイックソート** [15] の三分割法と密接な関係がある．彼らのアルゴリズムは上述のものと同様に $\mathcal{E}$ を再帰的に分割しながら整列するが，以下に示す二つの相違点がある．第一に，分割値 v は最小値でなくても良く，真の中央値から乱数までさまざまな選び方がある．第二に，$\mathcal{E}$ は三つのグループ $\mathcal{E}_<$，$\mathcal{E}_=$，$\mathcal{E}_>$ に分割される．ただし，$\mathcal{E}_<$ は d 番目の値が v より小さいハイパーエッジからなり，$\mathcal{E}_=$ と $\mathcal{E}_>$ も同様に定義される．効率的に分割を行う方法は重要である．マルチキー・クイックソートの性能は理論的によく解析されている．ここでは [15] から以下の二つの定理を引用する．c は定数を表す．中央値を計算するアルゴリズムで，最悪時に $c=3$ を実現するものと平均で $c=3/2$ を実現するものがそれぞれ [16] と [17] で与えられている．

定理 1. もしマルチキー・クイックソートが cn 回の比較で計算される中央値に関して分割を行うならば，n 個の長さ k のベクトルは高々 $cn(\log n+k)$ 回の比較で整列される．

定理 2. もしマルチキー・クイックソートがランダムに選ばれる $2t+1$ 個の要素の中央値に関して分割を行うならば，n 個の長さ k のベクトルは高々 $2nH_n(H_{2t+2}-H_{t+1}+O(kn))$ の平均比較回数で整列される．ただし，H_n は調和数を表し，$H_n=\sum_{1\le i\le n}1/i$ で与えられる．

マルチキー・クイックソートを用いてハイパーエッジを



(a) 配列　(b) 整列過程（右から左に配置）　(c) 二分木表現　(d) ZDD 表現

図 3　配列として表された五つの集合 $s[0], \ldots, s[4]$ の整列過程と対応する ZDD 表現

整列させるとき比較回数は台集合のサイズに依存しないので，疎なハイパーグラフに対して効果的である．しかしながら，マルチキー・クイックソートは三分割法に基づくので三分探索木と同型であり，従ってマルチキー・クイックソートから直接 ZDD を生成できそうにない．

そこで，提案アルゴリズムは以下の二段階の方法で ZDD を構築する．はじめに，集合族をマルチキー・クイックソートにより整列させる．その後，二分割法に基づき，ZDD を構築する．アルゴリズム 1 に構築部分のアルゴリズムを示す．

Algorithm 1 集合族 $\mathcal{E}$ を ZDD に圧縮する．

```
function ZCOMP(E, d)
    if E = ∅ then
        return ⊥;
    end if
    v ← E に属する集合の d 番目の値のうちで最小の数;
    if v = +∞ then
        return ⊤;
    end if
    E_= ← d 番目の値が v のハイパーエッジ全体;
    E_> ← E \ E_=;
    hi ← zcomp(E_=, d + 1); lo ← zcomp(E_>, d);
    return zdd_unique(v, lo, hi);
end function
```

定理 3. 入力集合族 $\mathcal{E}$ は辞書式順に整列された配列として与えられるとする．アルゴリズム 1 は $O(N \log^2(\sum_{U\in\mathcal{E}} |U|/N))$ 時間で動作するように実装可能である．ここで，N は出力 ZDD の等価な部分グラフを共有しないときに得られる二分木の節点数を表す．必要な記憶量は出力 ZDD の節点数に比例する．

証明. $\mathcal{E}$ は整列済みの配列として与えられるので，$\mathcal{E}$ を $\mathcal{E}_=$ と $\mathcal{E}_>$ に分割するためには，d 番目の値が v に等しい最後のハイパーエッジを発見すれば十分である．このために，できるだけ多くのハイパーエッジを飛び越しながら探索すれば効率的に発見できる．以下の手続きを考察されたい．

(1) U を最初のハイパーエッジとする．
(2) d 番目の値が v である間，U から順に $2^0, 2^1, \ldots$ 番目のハイパーエッジを探索する．
(3) もし最後のハイパーエッジが発見されないならば，U を現在のハイパーエッジに更新して手続き (2) に戻る．もし発見されるならばそれを返す．

手続き (2) は高々 $\log|\mathcal{E}_=|$ 回の試行で終了するので，手続き全体で $O(\log^2|\mathcal{E}_=|)$ 時間を要する．d 番目の値のうち v に等しいものはこの後の再帰呼び出しで検査されることはないので，計算に現れるすべてのサイズ $|\mathcal{E}_=|$ の和はハイパーエッジのサイズの総和に等しい．したがって，Jensen の不等式から計算時間は $O(N \log^2(\sum_{U\in\mathcal{E}} |U|/N))$ である．

アルゴリズム 1 は出力 ZDD をボトムアップに構築するので，zdd_unique によって得られる ZDD 節点はすべて出力 ZDD に現れる．また，再帰呼び出しの深さの最大値は出力 ZDD における道の長さの最大値に対応するので，再帰呼び出しのために必要な記憶量は出力 ZDD サイズによっておさえられる．□

4. 実験

実装と環境. 提案アルゴリズムと和演算に基づく既存の圧縮法を C 言語で実装した．さらに，マルチキー・クイックソートと既存手法の組合せ，すなわち，ソートした後に既存手法で ZDD を圧縮する方法も実装した．これらのプログラムでは，湊により開発された BDD パッケージ (BDD Package Sapporo-Edition-1.0) を用いた．マルチキー・クイックソートの実装（ただし，高速化の調整が施されていないもの）を [18] 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 である．すべてのプログラムを gcc4.3.4 版でコンパイルした．

問題例. 以下の問題例を Hypergraph Dualization

Repository [19] から入手した．

- BMS-WebView-2 (bms(n))：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す．パラメタ n はしきい値を表す．
- Uniform random (rand(n))：台集合のサイズが 50 で，ハイパーエッジの数 1,024,000 で，さらに各節点が確率 $n/10$ でハイパーエッジに含まれるように生成されたハイパーグラフを表す．

また，問題例 T40I10D100K，retail，accidents を Frequent Itemset Mining Dataset Repository から入手した．さらに，以下の問題例を作成した．

- しきい値関数 (TH(n))：30 変数のしきい値関数，すなわち，30 変数のうち n 変数以上が 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（ハイパーエッジの数は $\binom{30}{n-1}$）

すべての問題例は以下の形式のデータファイルとして与えられるものとする．各行がハイパーエッジに対応し，項目はすべて正整数とする．さらに，同一行の項目は昇順に整列されており，空白で区切られている．

アルゴリズムの比較． 提案アルゴリズム (zcomp)，和演算に基づく既存手法 (naive)，マルチキー・クイックソートと既存手法の組合せ (sort+naive) を比較した．ここで，zcomp はマルチキー・クイックソートとアルゴリズム 1 の組合せを表す．結果を図 4 と表 1 に示す．本実験において，入力データファイルの行はランダムに並べられている．

- 提案手法は実行時間と最大メモリ使用量のいづれに関しても最も性能が良かった．特に，問題の規模が大きくなるにつれ差が顕著に現れた．
- しきい値関数において既存手法は整列した後で実行した方が（整列しない時より）性能が良かったが，他の問題例においてはほとんど差が見られなかった．既存手法に関しては整列を行うことの一定の効果が見られるが，それでもなお提案手法との間に大きな差があった．整列だけでは不十分であることが観察される．
- ランダムなハイパーグラフの実験結果では問題例のパラメータサイズに応じた変化が見られなかった．このことから，この問題例に関してはハイパーエッジのサイズは性能にあまり影響しないように思われる．

5. まとめ

本稿では，ハイパーグラフを ZDD として高速に圧縮するアルゴリズムを提案した．本アルゴリズムはマルチキー・クイックソートを用いてハイパーエッジ全体を整列し，その後，ボトムアップに ZDD を構築する．構築部分の計算量を解析した．さらに，さまざまな問題例を用いた実験を行い，本アルゴリズムは和演算に基づく既存手法よりもかなり高速かつ省メモリに動作することを示した．ZDD を用いる計算の枠組みは大規模ハイパーグラフを効率的に処理するためのもっとも有効な手段の一つであるが，この枠組みに基づく計算が提案アルゴリズムによりさらに高速化・省メモリ化されうることが期待される．

今後の課題を以下にまとめる．データベース解析への応用では，重複するレコードを含むデータベースを重複度とともに圧縮する必要がある．このために既存研究では ZDD や BDD を拡張したデータ構造が用いられており，和演算に基づく圧縮が行われている（ [10], [14]）．これらの拡張データ構造に関しても効率的な圧縮手法を導入する必要がある．

参考文献

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

[2] Sekine, K., Imai, H. and Tani, S.: Computing the Tutte polynomial of a graph of moderate size, *6th International Symposium on Algorithms and Computations*, Cairns, Australia, pp. 224–233 (1995).

[3] 今井　浩：組合せ数え上げでの近似解法と厳密解法，離散構造とアルゴリズム V（藤重　悟，編），近代科学社，pp. 1–50 (1998).

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

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

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

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

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

[9] Toda, T.: Hypergraph Transversal Computation with Binary Decision Diagrams, *12th International Symposium on Experimental Algorithms, accepted*, Rome, Italy (2013).

[10] Minato, S. and Arimura, H.: Efficient Method of Combinatorial Item Set Analysis Based on Zero-Suppressed BDDs, *IEEE/IEICE/IPSJ International Workshop on Challenges in Web Information Retrieval and Integration (WIRI-2005)*, Tokyo, Japan, pp. 3–10 (2005).

[11] Goethals, B.: Survey on frequent pattern mining, `http://win.ua.ac.be/~adrem/bibrem/pubs/fpm_survey.pdf`.

[12] Minato, S., Uno, T. and Arimura, H.: LCM over ZBDDS: fast generation of very large-scale frequent itemsets using a compact graph-based representation, *12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, Osaka, Japan, pp. 234–246 (2008).

[13] Uno, M., Kiyomi, H. and Arimura, H.: LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal

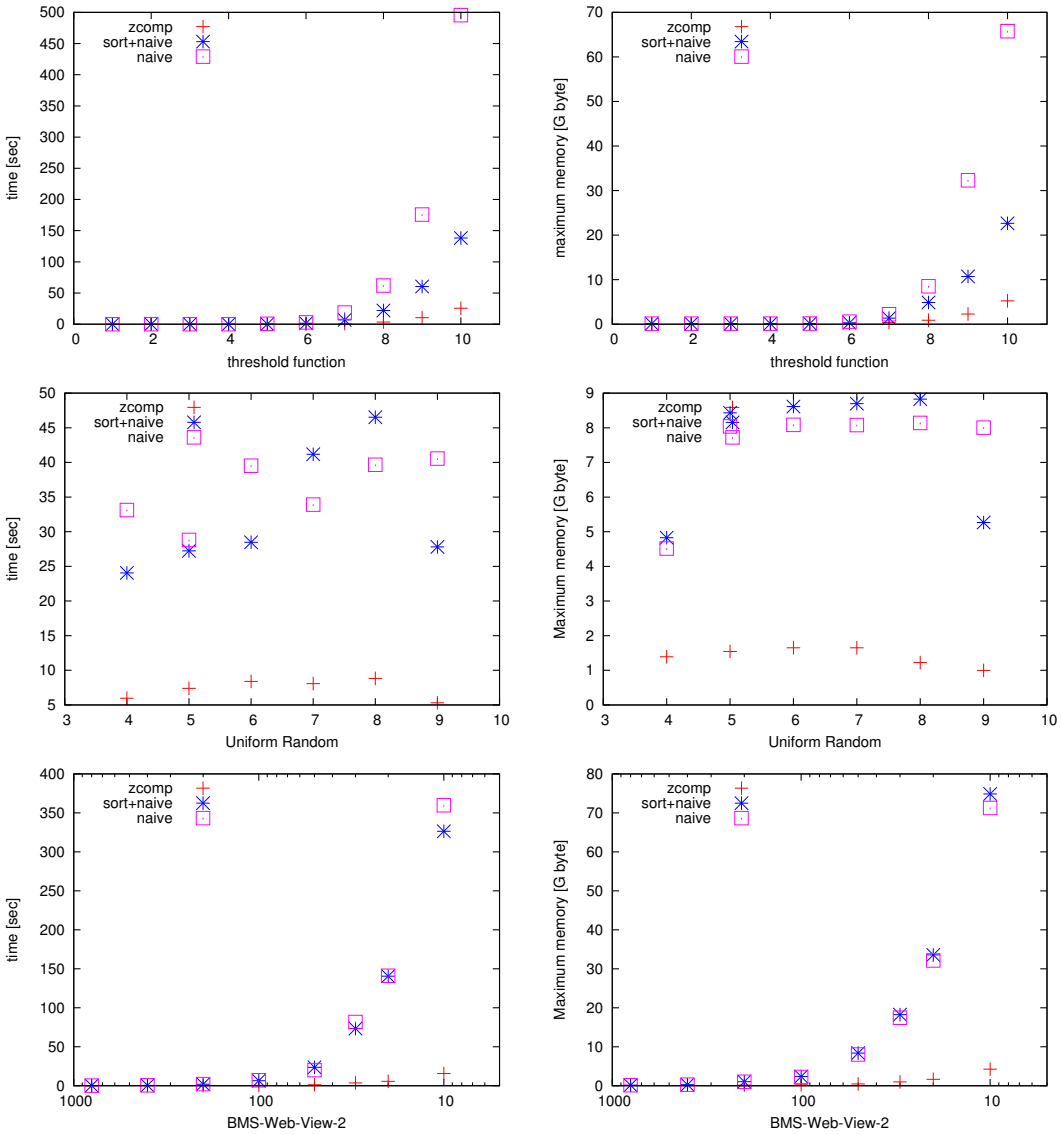


図 4 実行時間（左段）と最大メモリ使用量（右段）の比較（点の水平成分：問題例のパラメタ）

表 1 実行時間と最大メモリ使用量の比較

問題例	時間（秒）			メモリ（ギガバイト）		
	naive	sort+naive	zcomp	naive	sort+naive	zcomp
T40I10D100K	7.32	7.53	2.53	2.00	2.07	0.63
retail	15.59	18.85	0.34	4.18	4.42	0.16
accidents	19.36	18.43	3.84	4.19	4.39	1.19

Itemsets, *the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, Brighton, UK (2004).

[14] Salleb, A. and Vrain, C.: Estimation of the Density of Datasets with Decision Diagrams, *Foundations of Intelligent Systems* (Hacid, M.-S., Murray, N., Raś, Z. and Tsumoto, S., eds.), Lecture Notes in Computer Science, Vol. 3488, Springer Berlin Heidelberg, pp. 688–697 (2005).

[15] Bentley, J. and Sedgewick, R.: Fast Algorithms for Sorting and Searching Strings, *8th Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, Louisiana, USA, pp. 360–369 (1997).

[16] Schoenhage, A., Paterson, M. and Pippenger, N.: Finding the Median, *Journal of Computer and Systems Sciences*, Vol. 13, pp. 184–199 (1976).

[17] Floyd, R. and Rivest, R.: Expected Time Bounds for Selection, *Communications of the ACM*, Vol. 18, No. 3, pp. 165–172 (1975).

[18] Bentley, J. and Sedgewick, R.: Fast Algorithms for Sorting and Searching Strings, `http://www.cs.princeton.edu/~rs/strings/`.

[19] Murakami, K. and Uno, T.: Hypergraph Dualization Repository, `http://research.nii.ac.jp/~uno/dualization.html`.