巨大で疎な組合せ集合を表すための三分索引化ZDD

青木 洋士 *1

戸田 貴久 *2

湊 真一 *1 *2

***1 北海道大学 情報科学研究科**

***2 JST ERATO湊離散構造処理系プロジェクト**

目次

- ZDD
  - 研究背景
- 三分探索木
- 提案手法
  - 三分索引化ZDD
  - 既存のZDDと三分索引化ZDDの相互変換
- 実験結果
- まとめ

ZDD (Zero-suppressed Binary Decision Diagram)

集合

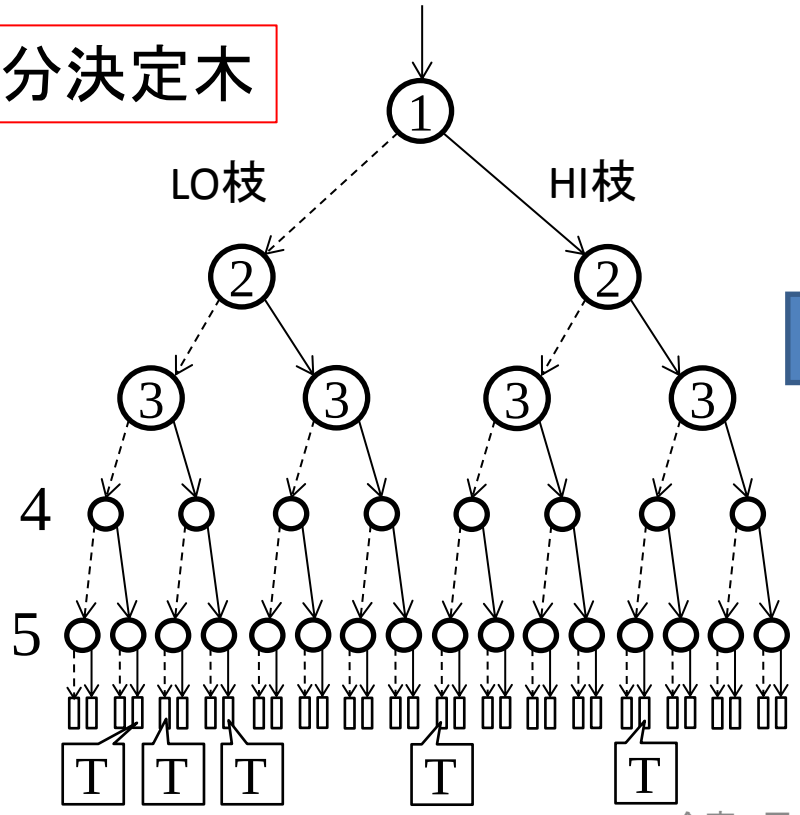
組合せ

{ {1, 2, 5}, {3, 4, 5}, {1}, {3}, {4, 5} }

アイテム

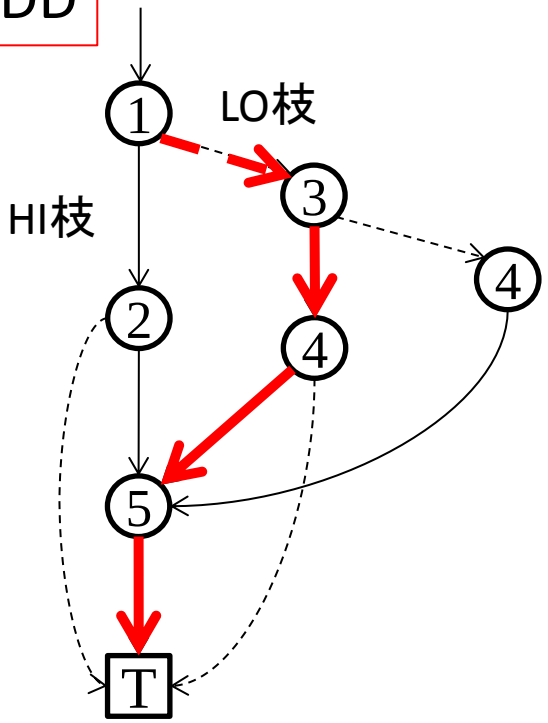
- アイテム集合マイニングに利用
- **アイテムの組合せ集合**を表現
- 二分決定木の圧縮形式

二分決定木



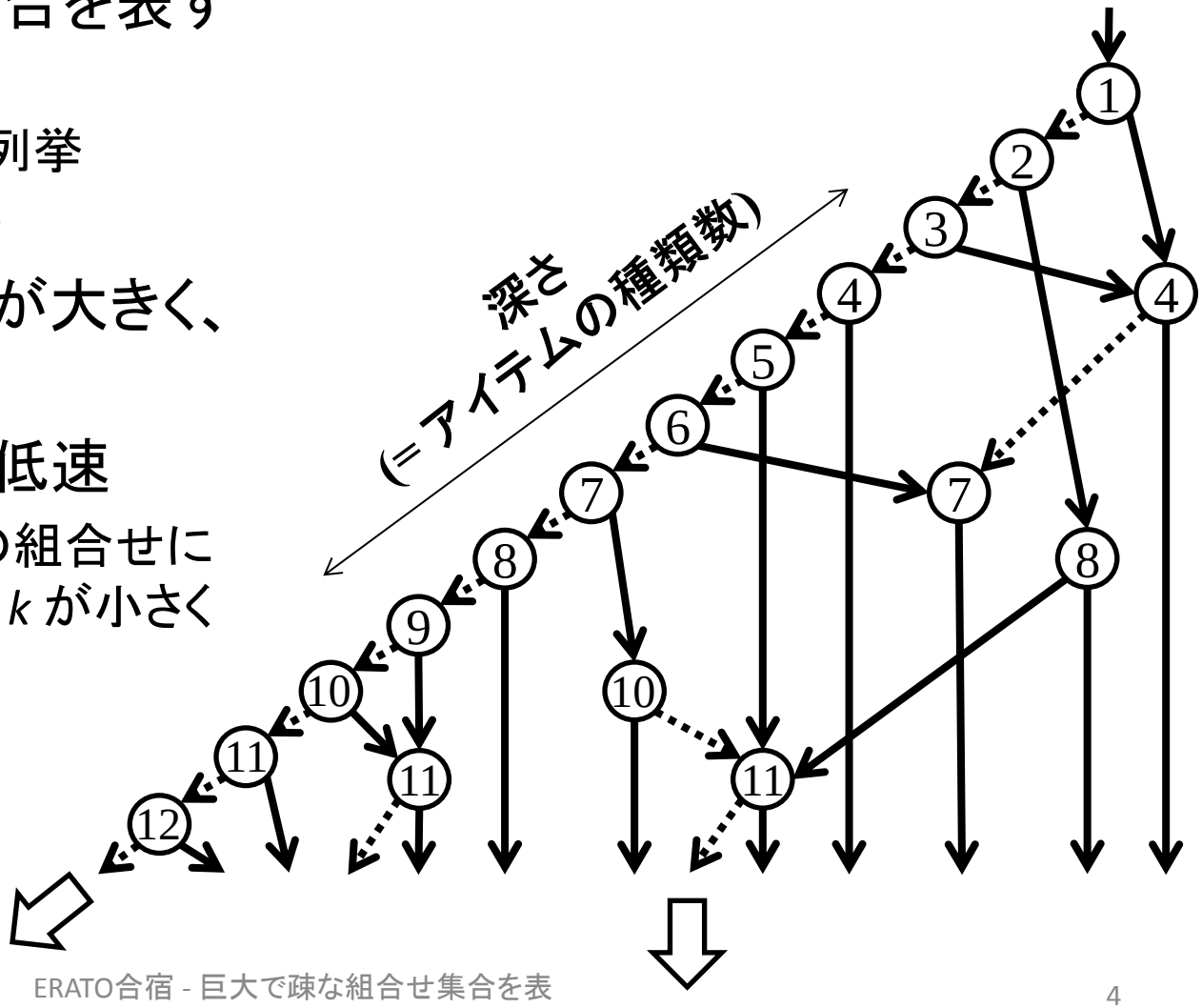
{3, 4, 5}

ZDD



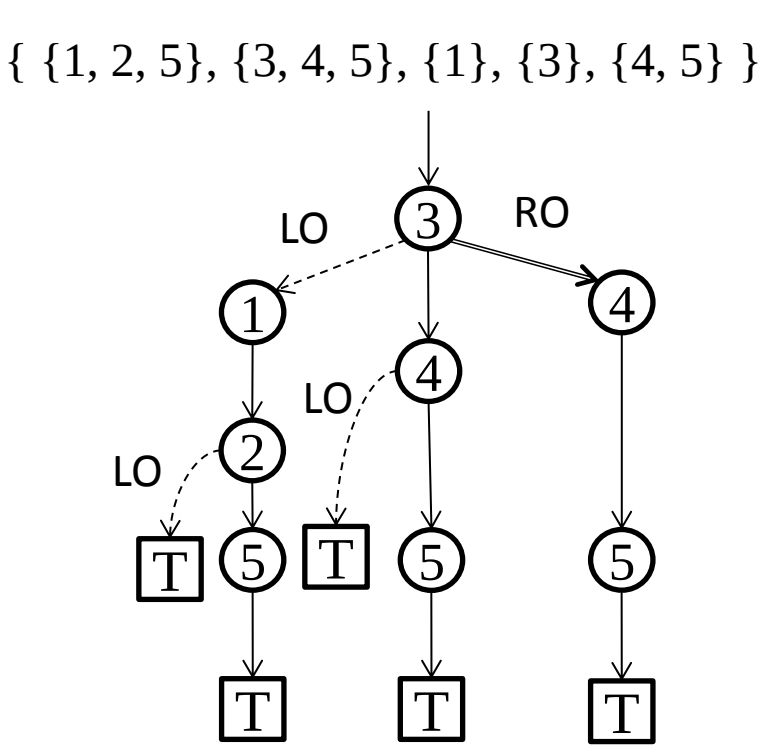
研究背景

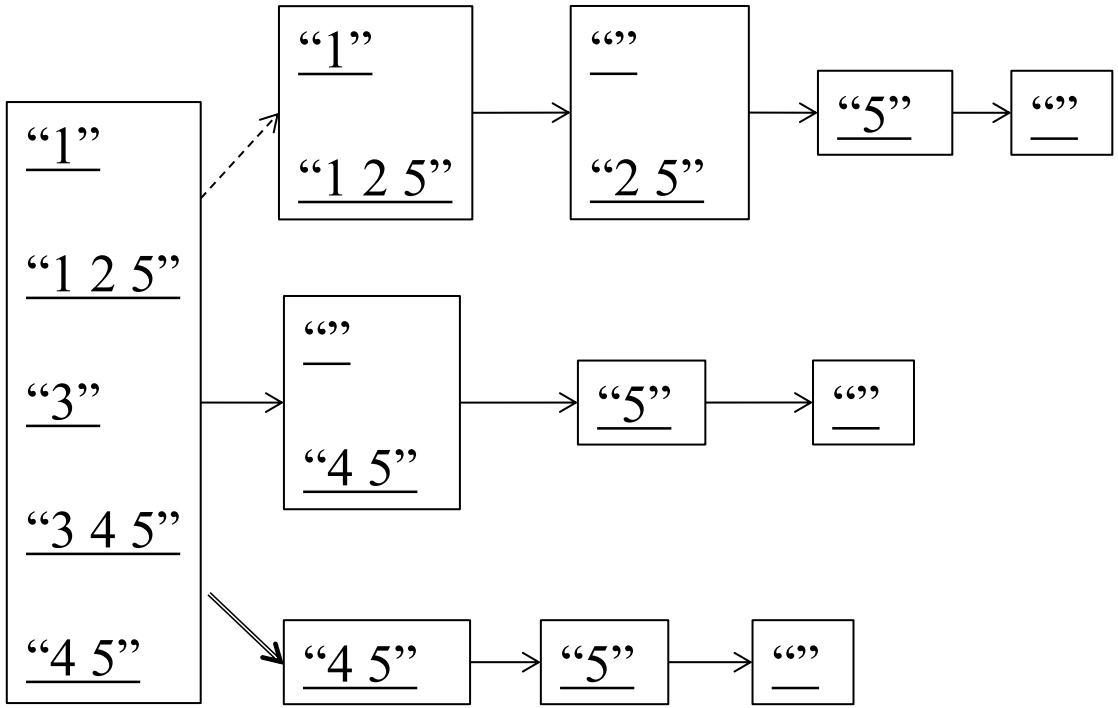
- 巨大で疎な組合せ集合を表すZDDが得られる応用
  - 頻出アイテム集合の列挙
  - 電力の配電網の列挙
- アイテムの種類数 σ が大きく、ZDDの深さが深くなる
- **メンバシップクエリ**が低速
  - クエリサイズ (クエリの組合せに含まれるアイテム数) k が小さくても $O(\sigma)$



三分探索木 (1/2)

- 文字列集合を表現するデータ構造
  - 本研究では組合せ集合を表現

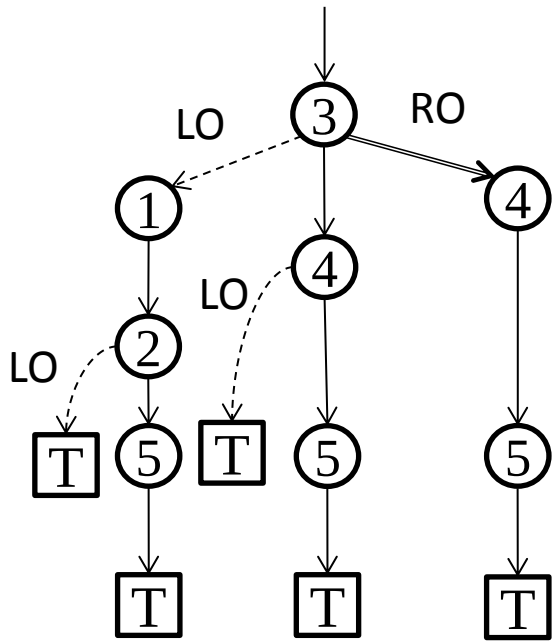


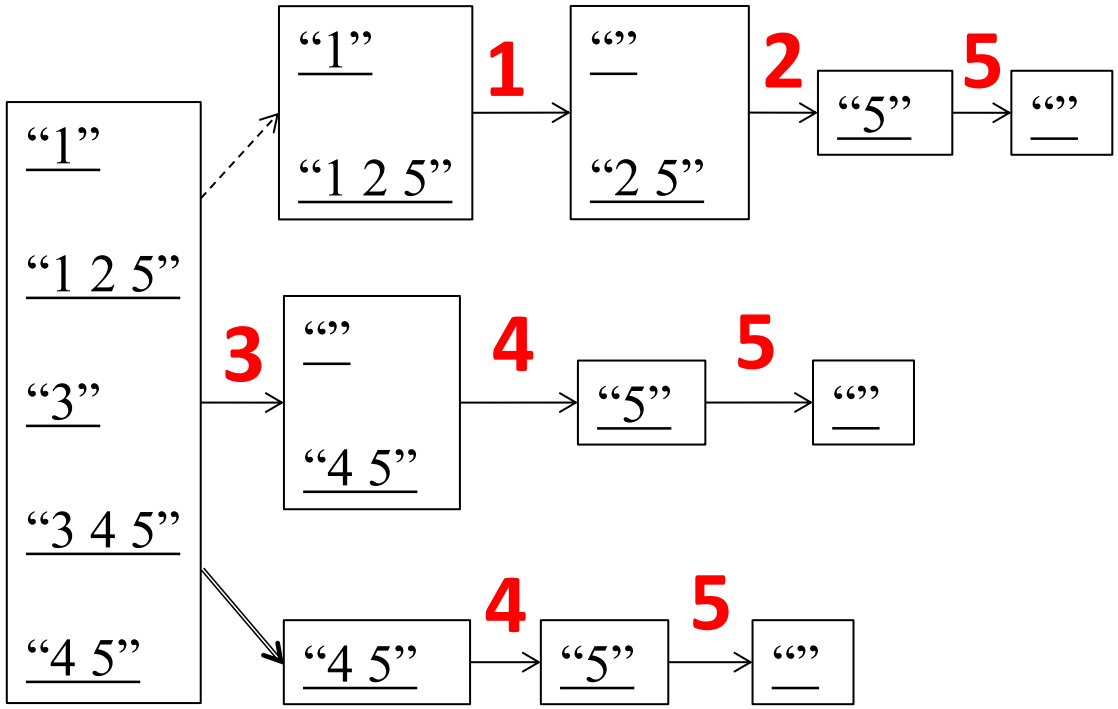


三分探索木 (2/2)

- 組合せを辞書順に並べ、中央値の順番の組合せで集合を前後に分割する三分木

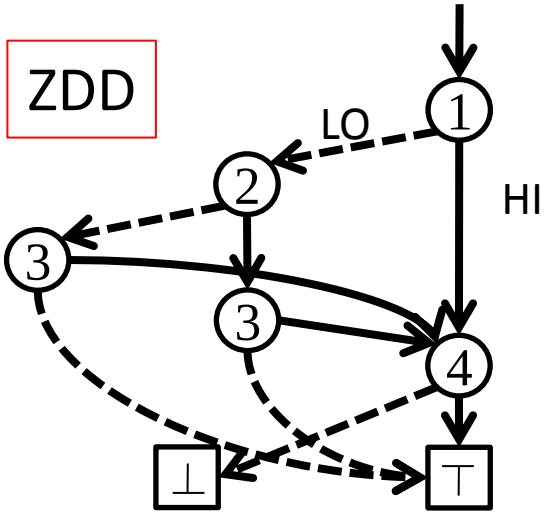
{ {1, 2, 5}, {3, 4, 5}, {1}, {3}, {4, 5} }



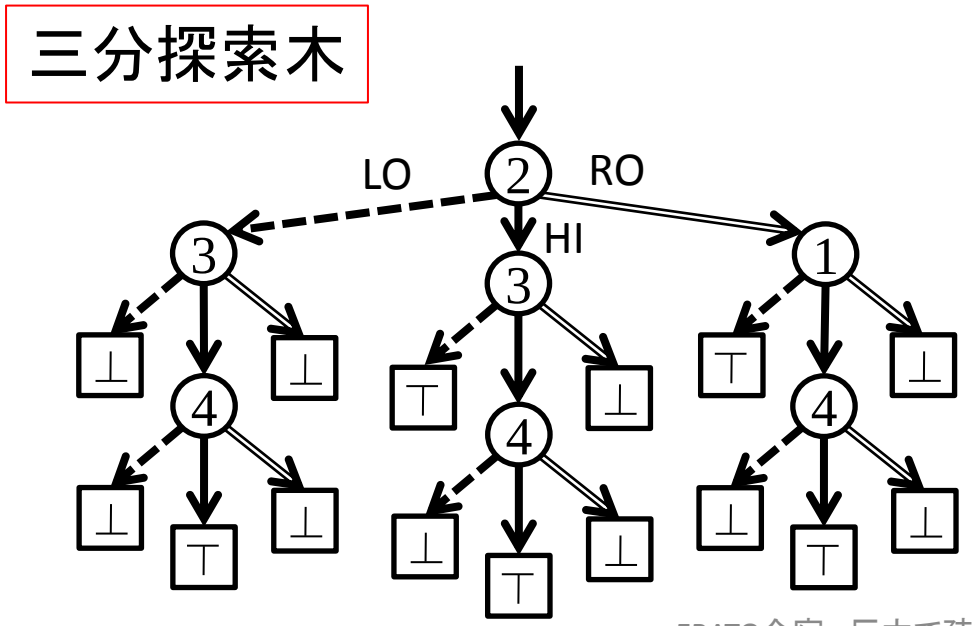


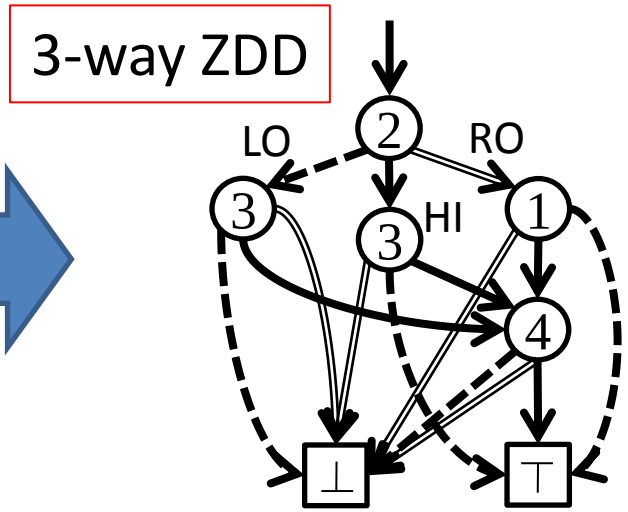
提案手法: 三分索引化ZDD (Three-way Indexing ZDD, 3-way ZDD)

- LO枝, HI枝に加えRO枝を加えたZDD
- 三分探索木の部分グラフを共有したもの

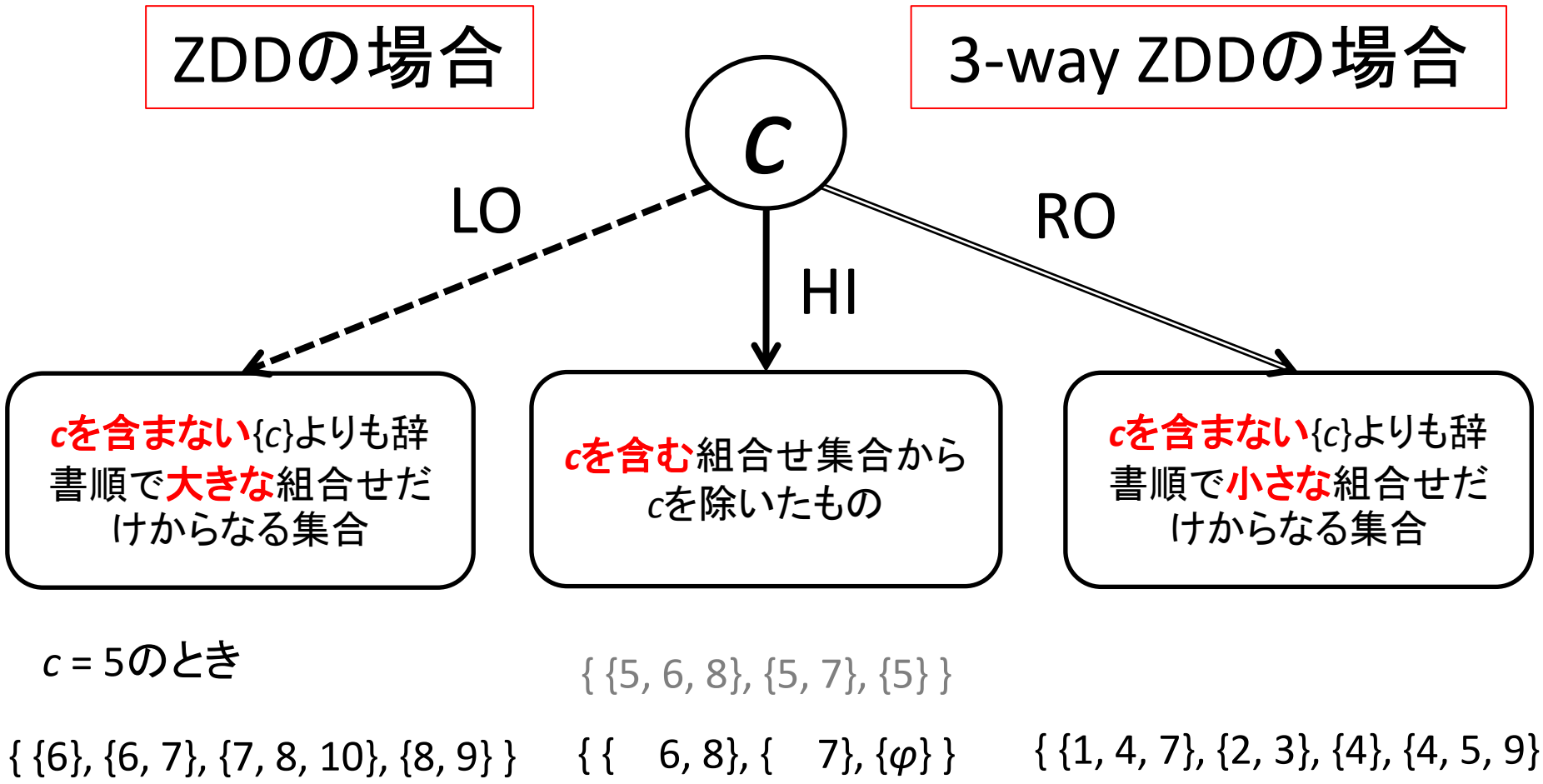


{ φ, {1, 4}, {2}, {2, 3, 4}, {3, 4} }





節点ラベルの出現順序


ZDDの場合
3-way ZDDの場合
c
LO
HI
RO
cを含まない{c}よりも辞書順で大きな組合せだけからなる集合
cを含む組合せ集合からcを除いたもの
cを含まない{c}よりも辞書順で小さな組合せだけからなる集合
c = 5のとき
{ {5, 6, 8}, {5, 7}, {5} }
{ {6}, {6, 7}, {7, 8, 10}, {8, 9} }
{ { 6, 8}, { 7}, {φ} }
{ {1, 4, 7}, {2, 3}, {4}, {4, 5, 9} }

3-way ZDDの特徴

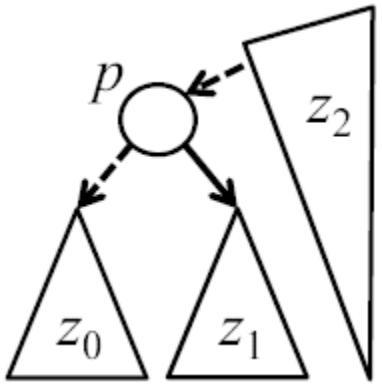
- ZDDと比べた利点
  - 最悪のパスの深さが浅くなる
  - パスの深さに依存するメンバシップクエリが高速
    - ZDD: $O(\sigma)$
    - 3-way ZDD: $\lfloor \lg n \rfloor + k$
- ZDDと比べた欠点
  - 動的にZDDを新しく構築する演算の実現が困難
    - 中央値を再計算する必要があるため

従来のZDDから3-way ZDDへの変換

各再帰呼び出しでZDDを3つの部分に分割

Algorithm 1 従来の ZDD から 3-way ZDD への変換

```
function Z2T(z)
    if z = ⊤ then
        return ⊤;
    end if
    if z = ⊥ then
        return ⊥;
    end if
    p ← FIND_PIVOT (z);
    z0 ← LO (p); z1 ← HI (p); z2 ← z \ p;
    t0 ← Z2T (z0); t1 ← Z2T (z1); t2 ← Z2T (z2);
    t ← TDD_UNIQUE (V (p) , t0, t1, t2);
    return t;
end function
```



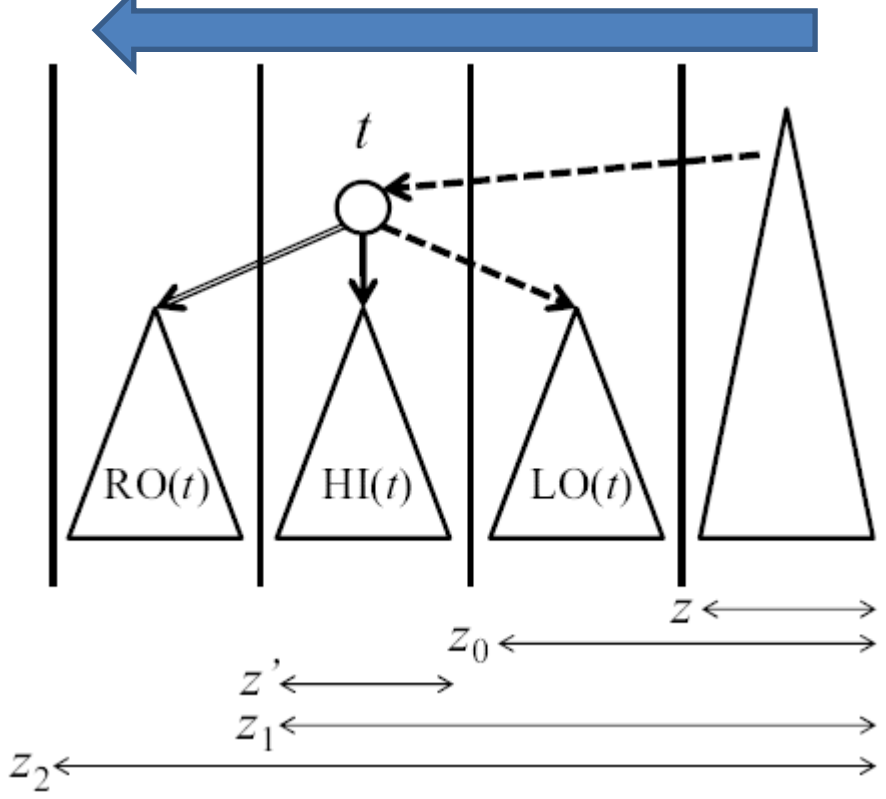
入力ZDD

3-way ZDDから従来のZDDへの変換

Algorithm 3 3-way ZDD から従来の ZDD への変換．最初は $z = \bot$ として呼び出さなければならない．

```
function T2Z(t, z)
    if t = ⊤ then
        return ⊤;
    end if
    if t = ⊥ then
        return ⊥;
    end if
    z0 ← T2Z(LO(t), z);
    z' ← T2Z(HI(t), ⊥);
    z1 ← ZDD_UNIQUE(V(t), z0, z');
    z2 ← T2Z(RO(t), z1);
    return z2;
end function
```

3-way ZDDを全走査し、ボトムアップでZDDを構築

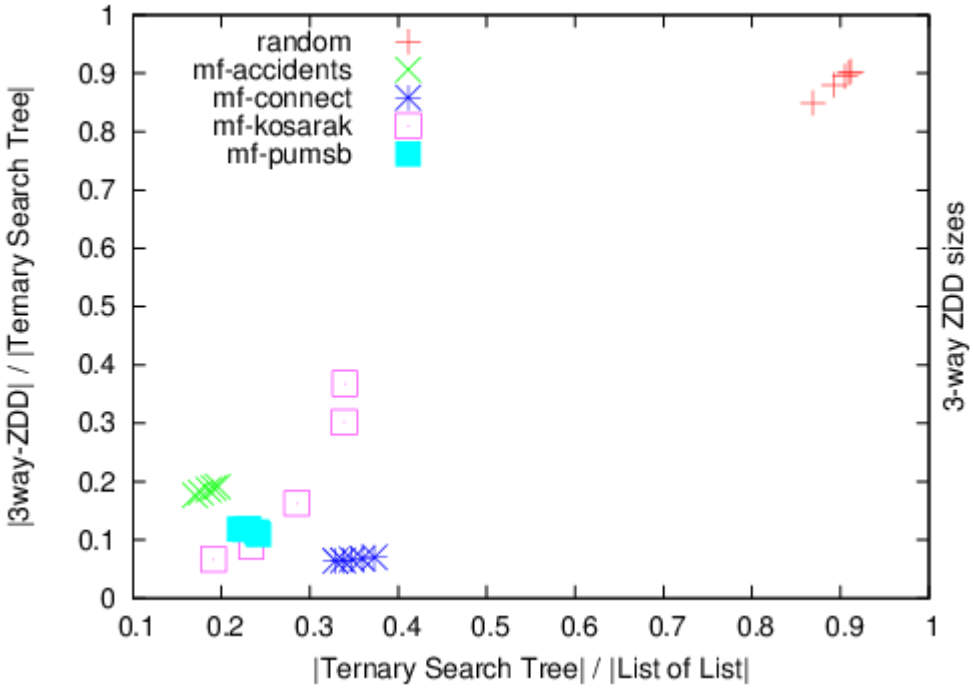


入力3-way ZDD

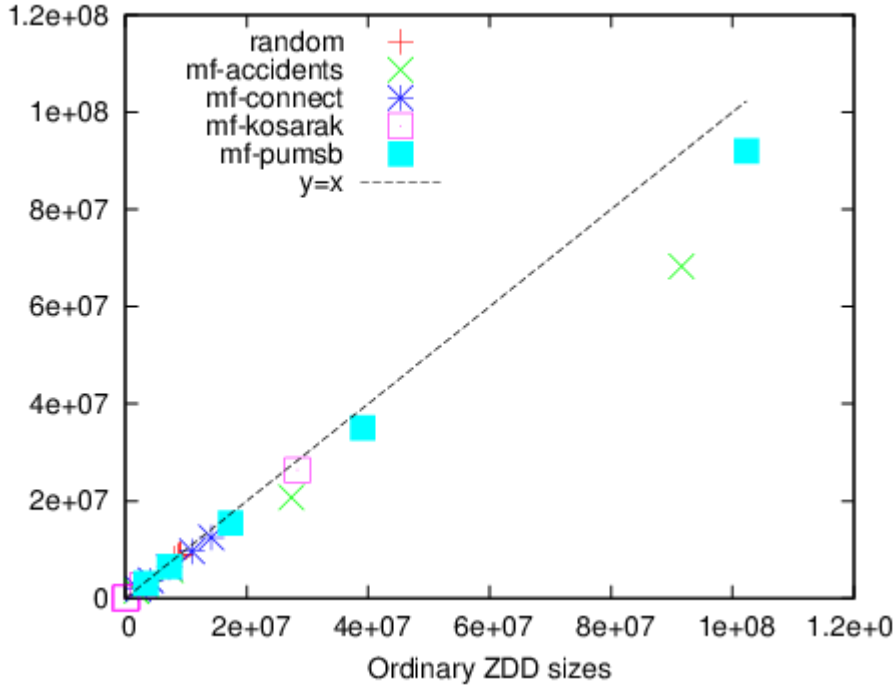
実験

- 実験1: データ構造のサイズの比較
  - 組合せ集合を表すアイテムの**List of List**
  - 組合せ集合を表す**三分決定木 (Ternary Search Tree)**
  - 組合せ集合を表す**3-way ZDD**
  - 組合せ集合を表す**ZDD**
- 実験2: メンバシップクエリの処理時間の比較
  - ZDDと3-way ZDD
- 実験3: 変換アルゴリズムをナイーブ法と比較
  - ナイーブ手法 : ”ZDD/3-way ZDD”をアイテムの”List of List”に展開してから、”3-way ZDD/ZDD”に変換する手法

実験1: データ構造のサイズの比較



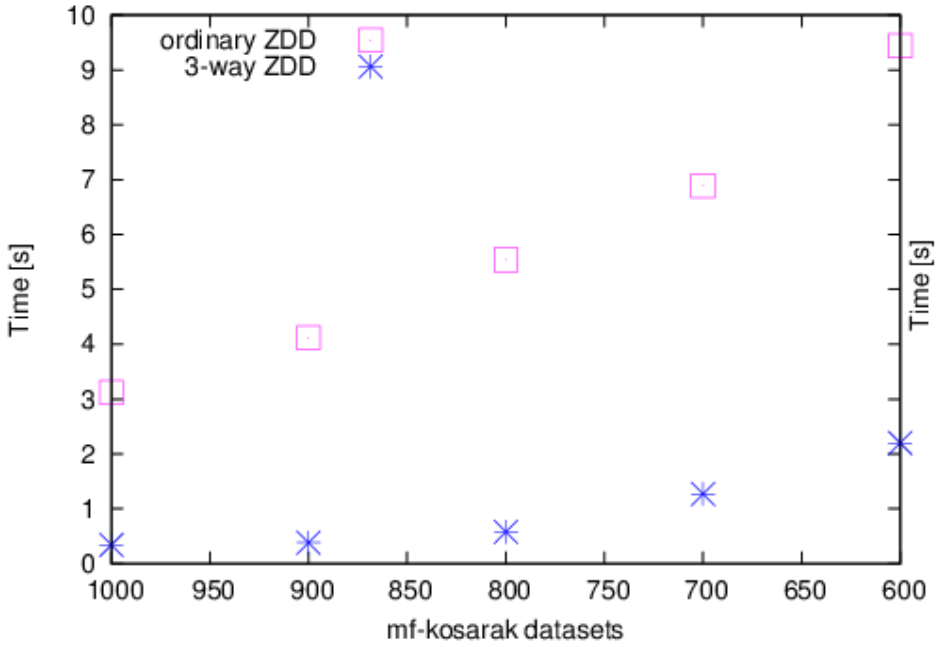
横軸: 三分決定木による接頭辞の圧縮効率
縦軸: 3-way ZDDによる接尾辞の圧縮効果

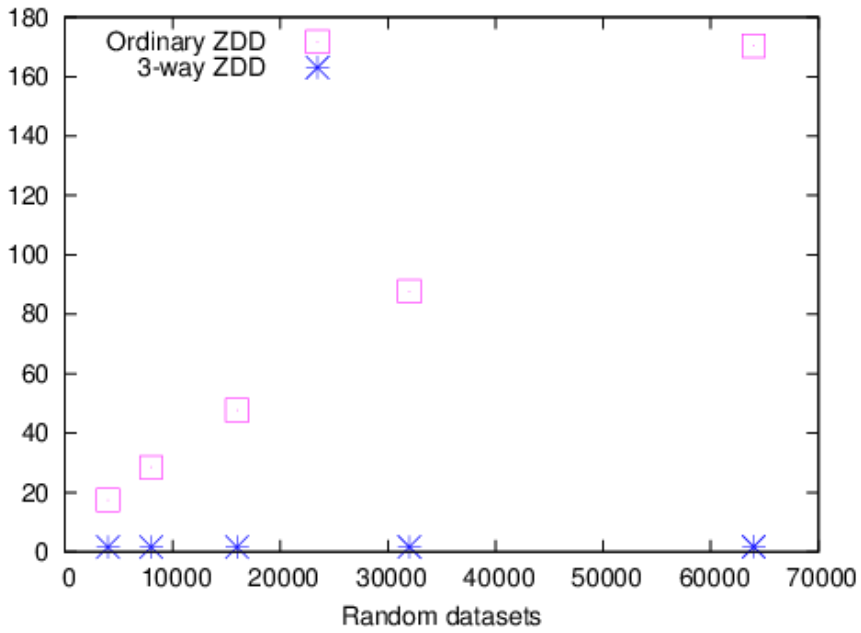


従来のZDDと3-way ZDDのサイズの比較

実験2: メンバシップクエリの比較

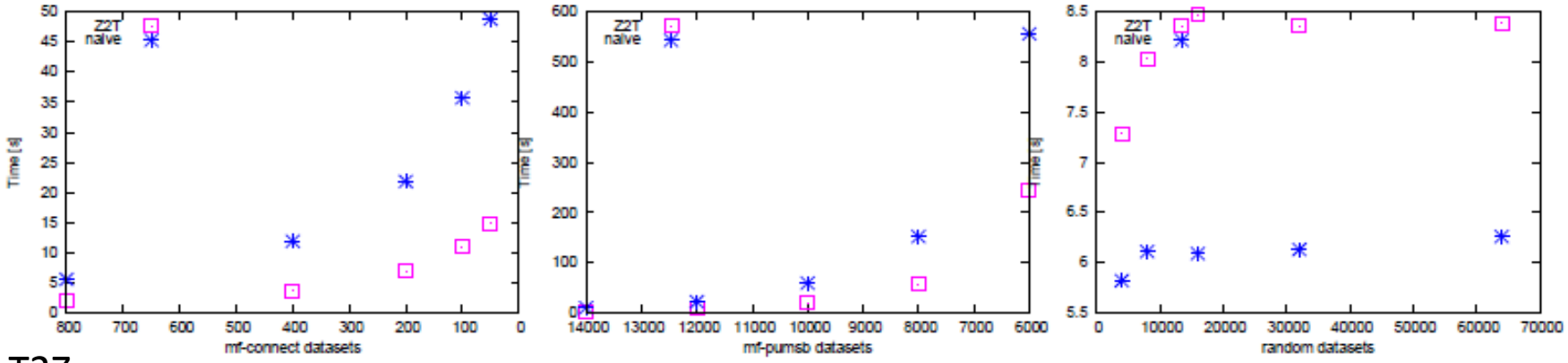
いずれのデータに対しても、3-way ZDDに対するメンバシップクエリが高速に動作



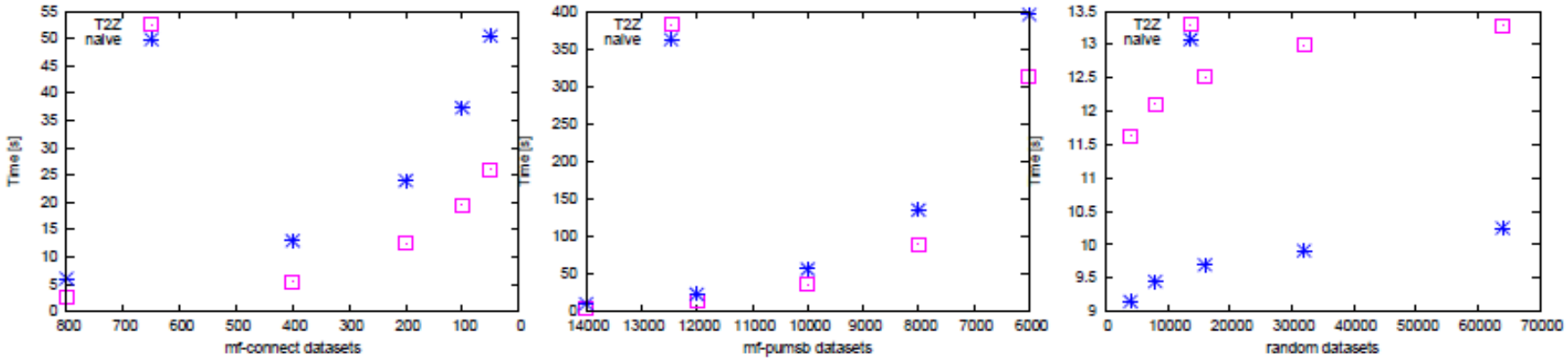


実験3: 変換アルゴリズムの比較

Z2T



T2Z



ランダムに構築したデータ以外では提案手法が高速に動作

まとめ

- 3-way ZDDを提案
  - ZDDと比べてメンバシップクエリが高速
  - 三分探索木と比べて節点数が削減
- ZDDと3-way ZDDの相互変換アルゴリズム
  - ナイーブ法と比べて高速
- 今後の課題
  - 中央値以外の効率の良い分岐の基準の考案