修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学学研究科　情報・通信工学専攻　博士前期課程		
氏　　名	吉田　隆史	学籍番号	1231104
論 文 題 目	フロンティア法を用いた根無し木の列挙		

要　旨

ゼロサプレス型 BDD(ZDD:Zero-Suppressed BDD)は，集合族をコンパクトに表現するデータ構造であり，特に疎な集合族に対して効率よく解を表すことができる．ZDD は，膨大な集合族を効率よく圧縮できる場合が多く，集合データに対する様々な演算も効率よく行うことができるという特長を持っており，VLSI の設計，電力網や道路網などの社会インフラの制約充足問題，データマイニング，遺伝子データの解析など，幅広い分野に応用されている．

特に近年，ZDD を用いて様々な制約条件を満たすグラフ構造を列挙する技法であるフロンティア法が注目を集めている．フロンティア法は，Knuth によって提案された，グラフ上の 2 頂点間のパスを列挙するアルゴリズムである SIMPATH を一般化したものである．この技法は従来の手法に比べて大幅な高速化に成功している．

本研究では，フロンティア法を用いて非同型な根無し木の列挙を列挙するアルゴリズムを提案した．提案するアルゴリズムは，与えられた頂点数 n からなる根無し木を全列挙するものである．ただし同型なものは一度しか出力しない．根無し木の列挙については，中野と宇野によって，木 1 つあたりの計算時間が定数時間であるアルゴリズムが提案されている．同型な木を重複して出力することを許すならば，フロンティア法を用いて全域木を列挙する方法によって，簡単に実現することができる．提案手法では，中野と宇野のアルゴリズムで用いられている同型な木の重複を防ぐためのアイデアを利用することで，ZDD を用いた非同型な根無し木の全列挙を行う．根無し木の列挙にフロンティア法を用いることで，従来の解空間を探索して解を 1 つ 1 つ出力する方法とは異なり，ZDD を用いて全ての解をコンパクトに表現して出力する．また，求められた解は ZDD として索引化された形で表現されるので，特定の条件を満たす解だけを容易に抽出することも可能である．

平成25年度　修士論文

フロンティア法を用いた根無し木の列挙

学籍番号 1231104
吉田隆史
情報・通信工学専攻　情報数理工学コース
指導教員:武永康彦准教授
副指導教員:垂井淳准教授

目 次

1 はじめに

ゼロサプレス型 BDD(ZDD:Zero-Suppressed BDD)[1] は, 集合族をコンパクトに表現するデータ構造であり, 二分決定グラフ (BDD:Binary Decision Diagram)[2] と呼ばれるデータ構造の派生形である. BDD は実用的な多くの論理関数を効率よく表現することができ, 組み合わせ最適化問題の解を求める手法の 1 つとして多くの分野に用いられている. その派生形である ZDD は, 特に疎な集合族に対して BDD よりも効率よく解を表すことができる. ZDD は, 膨大な集合族を効率よく圧縮できる場合が多く, 集合データに対する様々な演算も効率よく行うことができるという特長を持っており, VLSI の設計, 電力網や道路網などの社会インフラの制約充足問題 [3], データマイニング [4], 遺伝子データの解析など, 幅広い分野に応用されている. 特に近年, ZDD を用いて様々な制約条件を満たすグラフ構造を列挙する技法であるフロンティア法 [5] が注目を集めている. フロンティア法は, Knuth によって提案された, グラフ上の 2 頂点間のパスを列挙するアルゴリズムである SIMPATH[6] を一般化したものである. この技法は従来の手法に比べて大幅な高速化に成功している.

本研究では, フロンティア法を用いて非同型な根無し木の列挙を列挙するアルゴリズムを提案した. 提案するアルゴリズムは, 与えられた頂点数 n からなる根無し木を全列挙するものである. ただし同型なものは一度しか出力しない. 同型な木を重複して出力することを許すならば, フロンティア法を用いて全域木を列挙する方法によって, 簡単に実現することができる. 提案手法では, 根無し木と 1 対 1 対応する標準形を定義し, フロンティア法を用いて標準形である木を列挙することにより，非同型な根無し木の全列挙を行う. 提案手法は, 根無し木の列挙にフロンティア法を用いることで, 従来の解空間を探索して解を 1 つ 1 つ出力する方法とは異なり, ZDD を用いて全ての解をコンパクトに表現して出力する. また, 求められた解は ZDD として索引化された形で表現されるので, 特定の条件を満たす解だけを容易に抽出することも可能である.

本稿では, 2 章で ZDD とフロンティア法の定義, 3 章で提案する根無し木の列挙アルゴリズムについて述べ, 4 章で本研究で残された課題について述べる.

2 準備

2.1 ZDD

ZDD は, 集合族を表現する有向非巡回グラフである. ZDD には開始頂点と呼ばれる特殊頂点があり, ある集合が集合族に含まれるかは否かは, 開始頂点から順に頂点を辿ることによって決定する. 各頂点は変数でラベル付けされており, 2 本の出力辺を持つ. その変数が集合に含まれるか否かに応じて, 出力辺の一方を決定的に選択する. 変数が集合に含まれることを意味する出力辺を 1 枝と呼び, 含まれないことを意味する出力辺を 0 枝と呼ぶ. ZDD は出力辺を持たないただ 2 つの終端頂点 $\top$ と $\bot$ を持ち, $\top$ に到達することは当該集合が集合族に含まれることを意味し, $\bot$ への到達は含まれないことを意味する.

図 2.1 は集合族 $Z = \{\emptyset, \{x_1, x_2\}, \{x_2, x_3\}\}$ を表す ZDD である. 変数順序は x_1, x_2, x_3 である. 開始頂点 x_1 から $\top$ までの経路のうち, 図 2.2 に示す x_1 をラベルに持つ頂点からの 0 枝,x_2 をラベルに持つ頂点からの 1 枝, x_3 をラベルに持つ頂点からの 1 枝と辿る経路は $\{x_2, x_3\}$ という集合を表す. 冗長な頂点の削除によって取り除かれた頂点については注意

が必要である. 例えば, 図 2.1 の表す集合族には, 集合 $\{x_1, x_2, x_3\}$ は含まれない. 図 2.3 に示す経路より, x_1 をラベルに持つ頂点から 1 枝を辿り, x_2 をラベルに持つ頂点から 1 枝を辿った後の頂点は, x_3 をラベルに持つ頂点ではなく $\top$ である. これは, x_3 をラベルに持つ頂点に冗長な頂点の削除が適用されたと見なすことができる. つまり, 集合に x_3 が含まれるならば $\bot$ に辿りつくべきで, そのような集合は集合族に含まれないことを意味する.

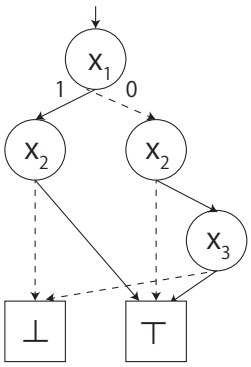

図 2.1: $Z = \{\emptyset, \{x_1, x_2\}, \{x_2, x_3\}\}$ を表す ZDD

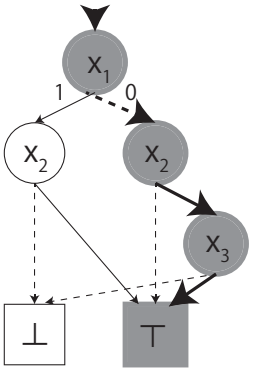

図 2.2: $\{x_2, x_3\}$ を表す経路

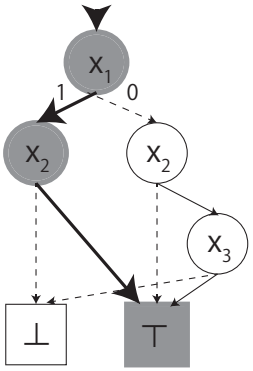

図 2.3: $\{x_1, x_2\}$ を表す経路

ZDD は, 冗長な頂点の削除と等価な頂点の共有という 2 つの簡約化規則を持つ. 冗長な頂点の削除は, 図 2.4 のように 1 枝が $\bot$ を直接指している場合, この節点を取り除く規則である. 等価な頂点の共有は, 図 2.5 のようにラベル, 0 枝の指す頂点, 1 枝の指す頂点の全てが同じであるような頂点を一つにまとめる規則である. 簡約化規則を可能な限り適用した ZDD を既約な ZDD と呼ぶ. 単に ZDD と言うときは, 既約なものを意味する.

変数の集合 $\{x_0, ..., x_{n-1}\}$ は, 置換 $\pi : \{0, ..., n-1\} \mapsto \{0, ..., n-1\}$ によって順序付けされる. ここで $\pi^{-1}(i) < \pi^{-1}(j)$ ならば, 開始頂点から任意の経路において終端頂点に辿っても, x_i は x_j より先に現れない. この π を変数順序と呼び, 以下では $\pi^{-1}(0), \pi^{-1}(1), ..., \pi^{-1}(n-1)$ の順に変数を並べて表現する. 変数順序を 1 つ決めたとき, 集合族を表す既約な ZDD は一意に定まる.

ZDD の特長として, ZDD によって多くの実用的な集合族が現実的な節点数で表現することができる, 集合族に対する和集合や差集合といった集合演算を ZDD のサイズにほぼ比例する時間で行うことができる, 変数順序を 1 つ決めると集合族を表す ZDD は一意に定まる, などの点が挙げられる. また, ZDD は特に疎な集合族に対して効率よく表現や操作が可能である.

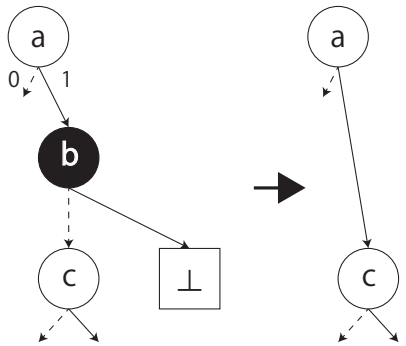

図 2.4: 冗長な頂点の削除

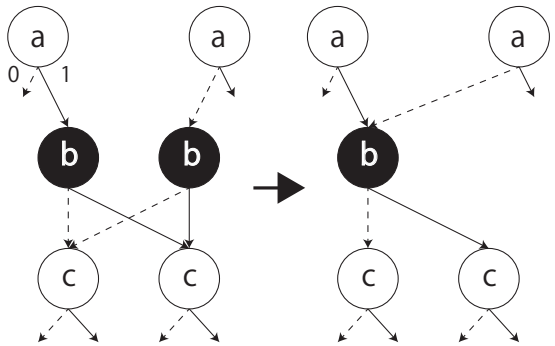

図 2.5: 等価な頂点の共有

2.2 フロンティア法

2.2.1 Simpath のアルゴリズム

Simpath のアルゴリズムは, グラフ上の 2 点間のパスを列挙する高速なアルゴリズムである. グラフ $G = (V, E)$ と G 上の 2 頂点 s, t が入力として与えられたとき, s から t へのすべてのパスを列挙する問題を考える. このアルゴリズムは, ちょうど s から t へのパスとなるような辺の組み合わせを全列挙した集合を表す ZDD を作るものである.

ここで, グラフ G の各辺 $e \in E$ は, ZDD が表す集合族における各集合の要素となる. s から t への各パス P は ZDD 上の開始頂点から $\top$ までのあるパスに対応し, P に含まれる辺は 1 枝を通り, 含まれない辺は 0 枝を通るか頂点のラベルとして現れない.

グラフ $G = (V, E)$ において, $i \in \{1, 2, \ldots, l-1\}$ に対し $(v_i, v_{i+1}) \in E$ であるとき, 頂点の列 $v_1 v_2 \cdots v_l$ (ただし $i \neq j$ となる i, j に対して, $v_i \neq v_j$), を v_1 から v_l へのパスという. また, パス $v_1 v_2 \cdots v_l$ に対し, 辺の集合 $\{(v_i, v_{i+1}) \mid i \in \{1, \ldots, l-1\}\}$ をパスと同一視する. 互いに異なる点素なパスの集合をパスマッチングという.

Simpath のアルゴリズム [2] の主なアイディアは, パスマッチングの集合 $\mathcal{P}$ を管理することである. 具体的には, まず $\mathcal{P}$ を空集合として初期化する. 辺に順序が与えられており, 各辺で次の操作が行われる. $\mathcal{P}$ の各パスマッチング P に辺 e を追加したパスマッチング P' 及び e を追加しないパスマッチングを $\mathcal{P}$ に新たに格納する. 全ての辺への処理が終了したとき, s から t へのパスのみで構成されるパスマッチングが出力される. 単純にこれを実行すると, パスマッチングの数は非常に多いため, 膨大な計算時間および領域が必要になる. Simpath では, すべてのパスマッチングを管理するのではなく, 今後行う操作が同じであるパスマッチングを同一視し, 管理すべきパスマッチングの数を減らしている. $E' \subset E$ を操作が終了した辺の集合, $P, P' \subseteq E'$ をパスマッチングとする. このとき, 任意の $E_P \subset E \setminus E'$ に対し, $P \cup E_P$ が s から t へのパスであるとき, またそのときに限り, $P' \cup E_P$ が s から t へのパスであるならば, P と P' を同値なパスマッチングという.

配列 $mate$ を用いて, 同値であることが明らかなパスマッチングを一つにまとめて取り扱う. $mate$ は要素数 $|V|$ で, 配列の添字は頂点に対応する. また, $mate$ の各要素の値はパスマッチング P の状態によって, 以下のような値をとる.

$$mate[i] = \begin{cases} j & P \text{に頂点 } i \text{ から } j \text{ へのパスが存在} \\ i & \text{頂点 } i \text{ を端点に持つ辺が使われていない} \\ 0 & \text{頂点 } i \text{ は } P \text{ のあるパスの中点} \end{cases}$$

Simpath では,mate の各状態を非既約な ZDD で管理する． Simpath の疑似コードを Algorithm 1 に示す.

Algorithm 1 Simpath アルゴリズム

1: **for each**　辺 $e \in E$ **do**
2:　**for each**　ZDD の頂点 (mate の状態 M) **do**
3:　　M に e を加えた状態 M' を作り,M の 1 枝を M' に接続する.
4:　　M に e を加えない状態 M'' を作り,M の 0 枝を M'' に接続する.
5:　**end for**
6: **end for**

ステップ 3 では, 辺 $e = (u, v)$ を追加するとき, 追加した後の状態もパスマッチングであるかを判定しなければならない． 具体的には, $mate[u] = v$　$(mate[v] = u)$, または $mate[u] = 0$, または $mate[v] = 0$ のとき, e は追加できない. それ以外の u と v の $mate$ の値を取るとき, e をパスマッチングに追加し, $mate$ を次のように更新する. $mate[u] = w$ かつ $mate[v] = x$ のとき, $mate[x] = w, mate[w] = x$ とする． このとき, $v \neq x$ ならば $mate[v] = 0$, $u \neq w$ ならば $mate[u] = 0$ とする. 例えば図 2.6 の左図の状態に辺 $(2, 5)$ を追加するとき, $mate$ は図 2.6 の右図に示すように更新される. 実線の辺はパスマッチングに含まれる辺, 点線の辺は含まれない辺を表す.

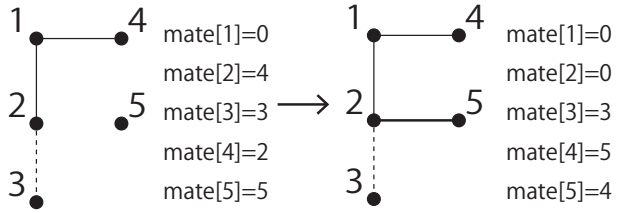

図 2.6: 辺 (2,5) の追加に伴う mate 値の変化

頂点 $v(v \neq t$ かつ $v \neq s)$ に接続するすべての辺の処理が終わったとき, v の次数が 1 ではいけない. つまり,$mate[v] = v$ または $mate[v] = 0$ でない mate の状態は,s から t へのパスにならないので破棄する.

さらに多くの同値なパスマッチングの共有を行うため, フロンティア上の頂点の $mate$ のみを管理する. フロンティアとは, 処理済の辺 E' および未処理の辺 $E \setminus E'$ の両方と接続している頂点の集合である. 処理済のどの辺をどのように使っているかは関係なく, フロンティア上の頂点の $mate$ の値が等しい二つのパスマッチングは同値であるので, フロンティア上の $mate$ のみを管理すればよい. これによって, 多くのパスマッチングの共有を行うことができ, 計算時間及び領域を大幅に削減することができる. 図 2.7 は辺 $e_1, ..., e_{11}$ まで処理済みの 2 つのパスマッチングに対応した状態で, 次に辺 e_{12} を処理する 2 つの異なる状態である. 太線の辺はパスマッチングに含まれる辺, 点線の辺はパスマッチングに含まれない辺, 細線の辺は未処理の辺を表す. 図 2.7 の (a) と (b) は, パスマッチングに含まれる辺は異なるが, フロンティア上の mate の値が一致しており,2 つのパスマッチングは同値である.

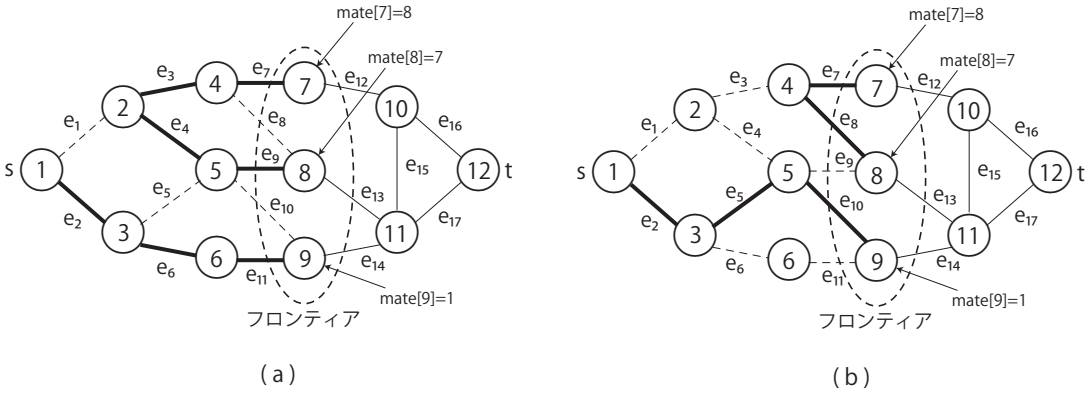

図 2.7: mate とフロンティア

2.2.2 フロンティア法としての一般化

Simpath アルゴリズムは,s-t パスの列挙に限らず, 様々な列挙問題に応用することができる．ある性質を持つ部分グラフの集合を表現する ZDD を構築する手続きにおいて, グラフを辿りながら ZDD をトップダウンで幅優先で構築し, フロンティア上の情報を基にノードの等価性を判定し, 共有を行う手法を総称してフロンティア法と呼ぶ.

ZDD の構築手順は基本的には s-t パスの場合と同じである．構築手順を疑似コードの形で Algorithm 2 に示す．Algorithm 2 の (a),(b),(c) を, 対象に応じて設計する.

Algorithm 2 ZDD 構築アルゴリズム

```
1段目に根ノードを作成
for i = 1 to m do
  for すでに作成済みの i 段目の各ノード n について do
    for x = 0, 1 do
      終端条件の判定 (a)
      新しいノード n′ を作成 (i + 1 段目とする)
      n′ の情報を更新 (b)
      if n′ と等価なノード n″ がすでに存在 (c) then
        n′ ← n″
      end if
      n の x 枝の先を n′ とする
    end for
  end for
end for
```

フロンティア法の例として, 全域木の集合を表現する ZDD を構築する手法について示す．ノードの等価性判定に必要な情報を configuration と呼ぶ．全域木の問題では, は configuration として part 配列を用いる．part 配列には, 各頂点が属する連結成分の番号を記憶させる．連結成分の番号は $1, \ldots, n$ の値を取る．最初はすべての頂点は孤立しているので,$j = 1, \ldots, n$ について,$part[v_j] = j$ と設定する.

G の部分グラフ G' が全域木になるには, 以下の (i),(ii) を満たしていればよい．(i)G' はサイクルを持たない．(ii)G' は 2 つ以上の連結成分を持たない.

(a) の終端条件の判定では, 各ノードの x 枝の先を作成する最中に,(i),(ii) をそれぞれ判定し, いずれかを満たさないと判定した場合は,$\perp$ に接続する．最終段のノード ($i = m$) の処理で,(i),(ii) を満たすと判定した場合, 全域木になることが確定するので, ノードの x 枝の先を $\top$ に接続する．具体的には,$e = \{v, w\}$ を全域木の辺として採用する場合,v と w が同じ連結成分に含まれているなら,v と w を接続するとサイクルが生じるので,(i) の条件を満たされなくなることが判定できる．(ii) の条件判定については, フロンティア上の頂点 u が, フロンティアから外れる場合,u 以外のフロンティア上の任意の頂点 v について,$part[u] \neq part[v]$ であるならば,u が属する連結成分は切り離されるので,(ii) が満たされなくなることを判定できる.

ZDD の各ノードは part 配列を記憶している．ノード n の part 配列を $n.part$ と表記する．ノード n の 0 枝,1 枝がさすノードをそれぞれ n_0, n_1 とする．(b) では,$n_0.part$ と $n_1.part$ を $n.part$ から計算している.

n において, 辺を採用しない場合は,part 配列は変化しない．したがってすべての頂点 u について $n_0.part[u] \leftarrow n.part[u]$ を実行すればよい.

n において, 辺 $e = \{v, w\}$ を全域木の辺として採用する場合は,$n.part[v] = n.part[w]$ のとき,v と w はすでに同じ連結成分に属し, サイクルが生じるので,n の 1 枝の先を $\perp$ に接続する．$n.part[v] \neq n.part[w]$ のとき,v と w は異なる連結成分に属するがそれらが結合される．$a = min\{n.part[v], n.part[w]\}, b = max\{n.part[v], n.part[w]\}$ とすると, 各頂点 v について, 以下を実行する.

$$n_1.part[v] \leftarrow \begin{cases} a & \text{if } \ n.part[v] = b \\ n.part[v] & \text{otherwise.} \end{cases}$$

すなわち, 大きいほうの連結成分の番号の頂点を, すべて小さいほうの連結成分の番号に置き換える.

(c) の等価性判定は, 同じ段の 2 つのノード n, n' に割り当てられた part 配列について, フロンティア上のすべての頂点の part 配列値が一致するとき,n と n' は一致すると判定できる. 実際には値の比較のために, 連結成分の番号が小さい順に現れるように番号を付け替える必要がある.

3　フロンティア法を用いた木の列挙

3.1　概要

正整数 n が与えられたとき, 頂点数が n である根無し木を全て列挙する問題を考える. ただし, 同型なものは一度しか出力しないものとする. 本章では, フロンティア法の拡張により, 頂点数が n の非同型な全ての根無し木を表現する ZDD を構築するアルゴリズムを提案する.

3.2　フロンティア法の拡張

完全グラフ K_n に対してフロンティア法を適用する. 2 章で示した全域木を表現する ZDD を構築するアルゴリズムをそのまま使えば, 頂点数 n の根無し木を列挙することができる

が, 同型な木が重複して列挙されてしまう. そこで, 根無し木と 1 対 1 対応する標準形を定義し, 標準形である全域木を表現する ZDD を構築するように拡張することによって, 同型な木の重複を防ぐ.

3.2.1 標準形

標準形を与えるにあたり, 順序木間における大小関係を定義する. 頂点 v の子供の数を $childNum(v)$ で表す.

定義 3.1 (順序木間の大小関係)**.** 順序木 T_1 と T_2 について, 幅優先探索で現れる順に, T_1 の各頂点を $a_1, a_2, \ldots, a_s$, T_2 の各頂点を $b_1, b_2, \ldots, a_t$ としたとき,

1. $childNum(a_i) = childNum(b_i)$（ただし, $i = 1, 2, \ldots, j-1$）かつ $childNum(a_j) > childNum(b_j)$ ならば, T_1 は T_2 より大きいといい, $T_1 > T_2$ で表す.

2. $s = t$ かつ $childNum(a_i) = childNum(b_i)$（ただし, $i = 1, 2, \ldots, s$）ならば, T_1 と T_2 は等しいといい, $T_1 = T_2$ で表す.

木 $T = (V, E)$ において, 頂点 $u \in V$ が任意の頂点 $v \in V - \{u\}$ との距離の最大値が最も小さい値を持つとき, u は T のセンターであると呼ぶ. 例えば図 3.1 では, 黒い頂点がセンターである. 木の直径が偶数のとき, 根は一意に定まる. 奇数の場合はセンターが 2 個になる.

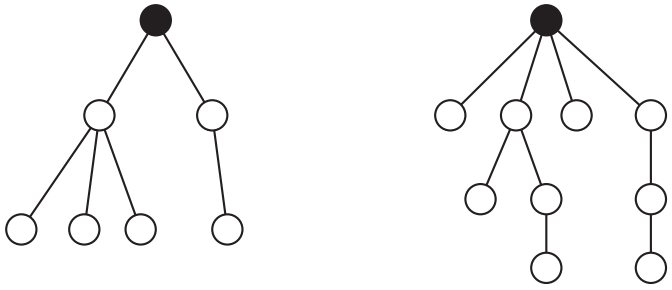

図 3.1: 根の与え方

根無し木 T に対して, T のセンターを根として割り当てることによって得られる根付き木を R とする. R に順序を与えた木のうち, 定義 3.1 の大小関係において最も大きい木を H とする. H を T の標準形であるという. 図 3.2 では, 根無し木 T に対応する順序木は 4 つ存在するが, $O_1 > O_2 > O_4 > O_3$ なので, O_1 が標準形 H となる.

任意の木 T に対し, 標準形 H は一意に定まるため, T の代わりに標準形 H を列挙することによって全ての根無し木を重複なく列挙することができる.

3.2.2 変数順序

K_n の各頂点をそれぞれ $v_1, v_2, \ldots, v_n$ でラベル付けをする. 各辺を変数として, 添字番号の小さい頂点に隣接する辺から順に処理していく. 例えば K_5 の場合, 図 3.3 に示す順に処理を行う.

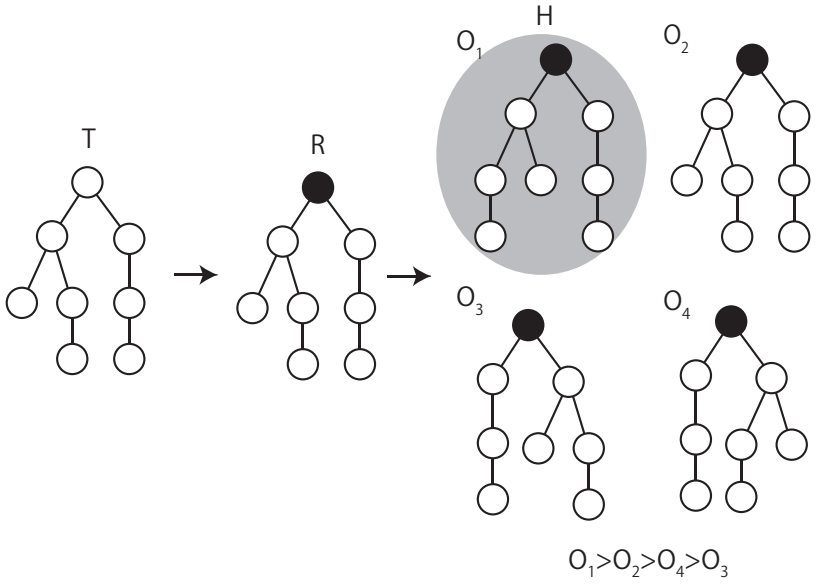

図 3.2: 根無し木 T と対応する標準形 H

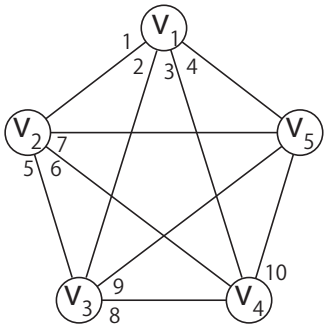

図 3.3: K_5 の変数順序

3.2.3　センターとなる頂点の固定

根無し木は, 木のセンターを根として扱うことによって, 根付き木と見なすことができる.2 章のアルゴリズムでは, 根無し木としては本来同型な木が, 重複して出力されている. 重複する木は, 根付き木と見なすと, 本来 1 つの根無し木に対して, センターが 1 つの場合は n 頂点のそれぞれを根とする n 通り,2 つの場合は n 頂点のうち 2 つの頂点を根とする ${}_nC_2$ 通りが存在する. 図 3.4 では, 頂点数が 4 の根無し木に対応して重複する木を表している.

このような重複を防ぐために, 木のセンターとなる頂点を予め指定する. センターが 1 つの場合は頂点 v_1,2 つの場合は v_1 及び v_2 がセンターとなる場合のみを有効な組み合わせとし, 残りはすべて $\perp$ へ接続する.

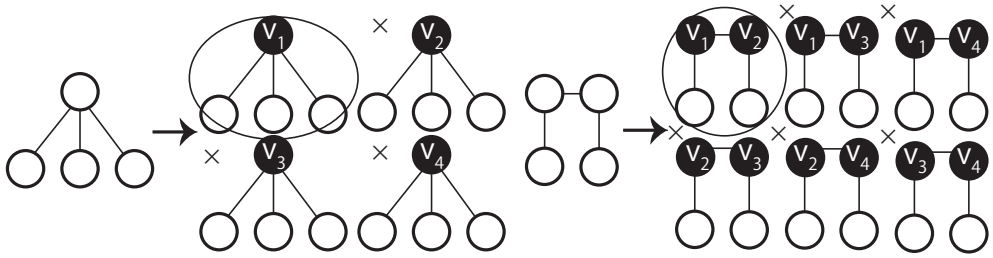

図 3.4: 頂点数が 4 の根無し木に対応して重複する木

3.2.4　同等な辺の処理

図 3.5 は,K_5 に対して辺 (v_1, v_5) までの処理が終わった状況である. 太線の辺はパスマッチングに含まれる辺, 点線の辺はパスマッチングに含まれない辺, 細線の辺は未処理の辺を表す. 辺 $(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5)$ のうち, パスマッチングにちょうど 3 本の辺を含む組み合わせは (a) から (d) まで 4 通り存在するが, これらは同等な処理であると見なすことができる. 同型なものを重複して列挙しないために,1 つだけを有効な組み合わせとして, 残りはすべて無効な組み合わせとして $\perp$ へ接続するようにしたい. これは, アルゴリズムの configuration 更新前の終端判定部分に, ある辺 (v_i, v_j)(ただし,$i+1 < j$) をパスマッチングに含めるとき, 辺 (v_i, v_{j-1}) がパスマッチングに含まれないならば $\perp$ へ接続する処理を追加することによって達成される. この処理によって,v_k に接続する $n-k$ 本の辺から l 本の辺をパスマッチングに含む ${}_{n-k}C_l$ 通りの組み合わせのうち, 辺 $(v_k, v_{k+1}), (v_k, v_{k+2}), \ldots, (v_k, v_{k+l})$ を含む組み合わせのみが有効なものとして存在し, 残りはすべて $\perp$ に接続されるようになる $(1 \leq k, l \leq n-1)$. 図 3.5 では,(a) が有効な組み合わせとして存在する.

3.2.5　configuration

木 $T = (V, E)$ において, 内部頂点 $v_0 \in V$ の k 個の子供 $v_1, v_2, \ldots, v_k$ を持つとき, $v_0 = v_1 v_2 \cdots v_k$ と表す. ただし, v_0 が根の場合は, ","を用いて $v_0 = v_1, v_2, \cdots, v_k$ で表す. また, $1 \leq p \leq k$ に対して, v_p の子孫をすべて含む部分木を T_p とする. v_0 の任意の 2 つの子供 v_s, v_t に対して, $T_s = T_t$ となるならば, 頂点 v_s と v_t は頂点 v_0 に対して同等であるといい, 同等な頂点を組にして $v_0 = v_1 v_2 \ldots (v_s v_t) \ldots v_k$ で表す. 例えば, 図 3.6 では, 頂

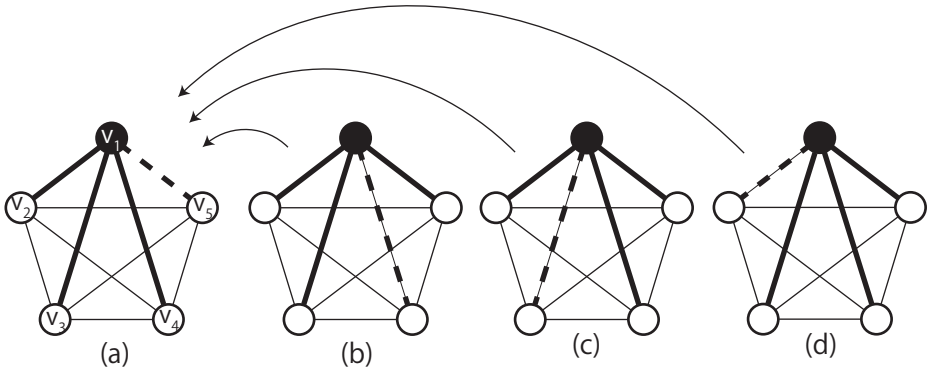

図 3.5: 1 つにまとめられる同等な辺の処理

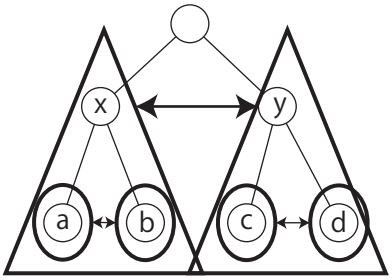

図 3.6: 同等な頂点の例

点 a と b は x に対して,c と d は y に対して,x と y が根に対して同等であるので, それぞれ $x = (ab), y = (cd)$, 根 $= (x, y)$ で表される.

グラフ $G = (V, E)$ の根に対して, 上記の表記を可能な限り展開すると, 葉の頂点だけからなる式で G を表現することができる. 展開した式から, フロンティア上に存在しない頂点を削除したものを G の式表現と呼ぶ. 図 3.6 に示すグラフの式表現は, $\{a, b, c, d\}$ がフロンティアとなるため, $((ab), (cd))$ で, 図 3.7 に示すグラフの式表現は $(bc), a$ である.

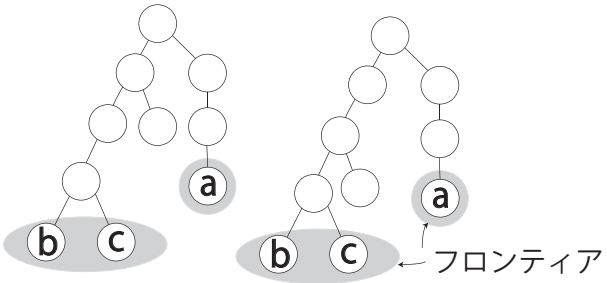

図 3.7: 等しい式表現によって共有が行われる例

式表現は標準形を満たすために必要な頂点間の関係を保持している. 式表現 $(\ldots, u, \ldots, v \ldots)$ において, 残りの操作で追加される部分木を考えると, 頂点 v の子供の数は, 同一括弧内で v より左に現れる頂点 u の子供の数より大きくなってはいけない. すなわち $childNum(u) \geq childNum(v)$ となければならない. 図 3.6 と図 3.7 の式表現が表す, 標準形を満たすために条件式は以下の式で表せる.

$$
\begin{aligned}
(bc), a \quad &\rightarrow \quad childNum(b) \geq childNum(c) \\
((ab), (cd)) \quad &\rightarrow \quad \Bigg(\quad childNum(a) \geq childNum(b) \\
&\qquad \wedge \; childNum(c) \geq childNum(d) \\
&\qquad \wedge \; \Big(\quad childNum(a) > childNum(c) \\
&\qquad\qquad \vee \; \big(childNum(a) = childNum(c) \wedge childNum(b) \geq childNum(e)\big)\Big)\Bigg)
\end{aligned}
$$

提案手法では,configuration として式表現を記憶する. mate 配列や part 配列では, ある 1 つの途中状態に対して, フロンティア上の各頂点が値を持っていたが, 本手法では 1 つの途中状態に対して 1 つの式表現を持つ.

図 3.7 の両図は, 構成されるグラフの形は異なるが, 式表現はどちらも $(bc), a$ となり等しいので, 等価なノードとして ZDD 上で共有が行われる.

補題 3.1. 式表現を configuration として記憶したフロンティア法において, 等価なノードの共有は正しく行われる.

証明. Configuration に基づいて共有が行われたとき, 共有した節点までのパスは複数存在する. そのうちの任意のパスが表すグラフが正しい解となるならば, 他のパスが表すグラフが正しい解となることを言えばよい.

M_P を通る任意のパス P が表すグラフが正しい解となると仮定する. ZDD の開始節点を M_0, 共有が行われた節点を M_P として, P を M_P 前後で分割する. 分割後のパスをそれぞれ P_1, P_2 とする. P_1 が表すグラフが標準形を満たすための条件式を NE とする. P を図 3.8 の左図に示す. 仮定より $P = P_1 + P_2$ が表すグラフは正しい解であるので, P_2 が表すグラフは条件式 NE を満たす.

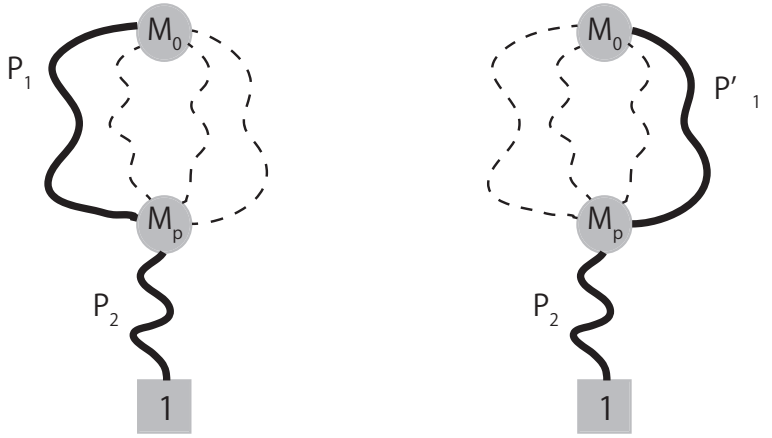

図 3.8: 任意のパス P とその他のパス P'

P 以外のパスを P' とし, P' の M_0 から M_P までの部分パスを P'_1 とする. P' を図 3.8 の右図に示す. P_1 と P'_1 は, M_0 から M_P への辿り方が異なるので, 図 3.9 に示すように構成

されるグラフも異なるが, M_P 時点での式表現は等しい. 従って, P_1' が表すグラフが標準形を満たすための条件式は, P_1 が表すグラフと同様に, NE である.

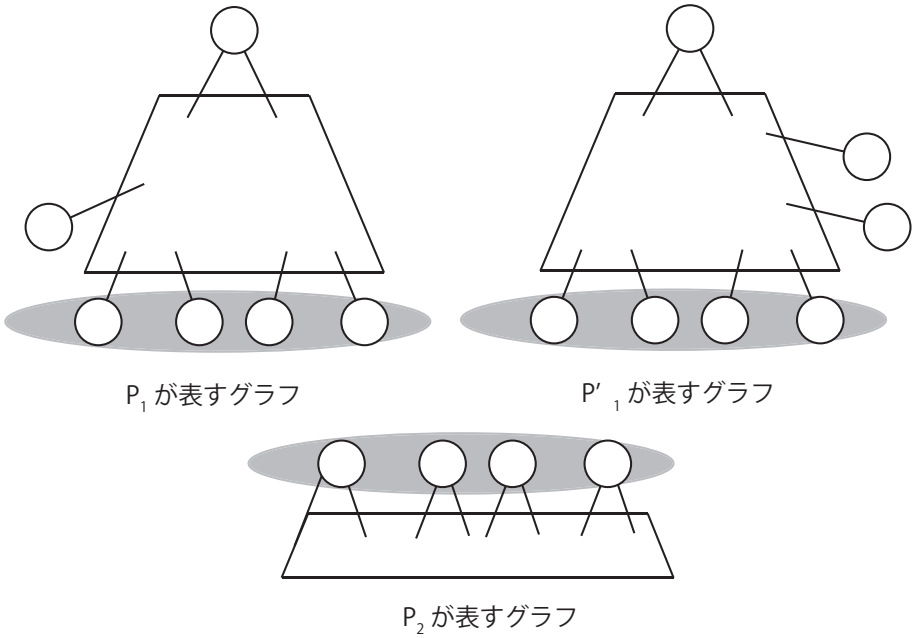

図 3.9: P_1, P_1', P_2 が表すグラフ

P_2 が表すグラフは条件式 NE を満たすので, $P' = P_1' + P_2$ が表すグラフは正しい解となる. 以上より, 題意は示された. □

3.3 アルゴリズム

提案手法のアルゴリズムは, フロンティア法を用いているので, 構造は Algorithm 2 と同じである. ただし, (a),(b),(c) の中身が 2.2.2 節とは異なるので, 以下で個別に述べる.

(a) の終端条件の判定では, 各ノードの $x \in \{0, 1\}$ 枝の先を作成する際に,(i) グラフがサイクルを持たないこと,(ii) 同等な辺の処理をまとめる規則において有効な組み合わせであること,(iii) グラフが標準形であること, (iv)v_1 または v_1, v_2 がセンター頂点から外れないことを判定し, いずれかを満たさないと判定した場合は,⊥ に接続する.

(i) の条件判定については, 具体的には,$e = (v_i, v_j)$ を根無し木の辺として採用する場合, 式表現に v_j が含まれているなら,v_i と v_j を接続するとサイクルが生じるので,(i) の条件が満たされなくなることが判定できる.

(ii) の条件判定については, 同じ v_i からの辺の中で, $j > i + 1$ かつ v_{j-1} が式表現に含まれていないなら,4.2.2 節で述べた無効な組み合わせにあたるので,(ii) の条件を満たさないことが判定できる.

(iii) の条件判定については, 辺を採用することによって, グラフが標準形でなくなってしまう可能性があるため, 式表現を用いて判定する. 具体的には, Algorithm 3 に従って判定を行う.

処理中の辺 $e = (v_i, v_j)$ を追加するとき ($i < j$), 5 行目の条件を満たし, 6 から 33 行目の処理に入る. 辺を追加しないとき, 式表現は明らかに標準形である. 6 行目からの処理では, v_i から出る辺を初めて使う場合は v_i, 2 本目以降の場合は v_{j-1} の位置から始め, 式を左

Algorithm 3 標準形であるかを判定するアルゴリズム

1: $leftPar \leftarrow 0$.
2: $rightPar \leftarrow 0$.
3: $parSub \leftarrow 1$.
4: $visited \leftarrow FALSE$.
5: **if** 処理中の辺 (v_i, v_j) を使う $(i < j)$ **then**
6: **if** v_i が式表現中に存在 **then**
7: $k \leftarrow v_j$が現れる位置.
8: **else**
9: $k \leftarrow v_j - 1$ が現れる位置.
10: **end if**
11: **while** $k > 0$ **do**
12: **if** k 番目の文字が‘(’ **then**
13: $leftPar \leftarrow leftPar + 1$.
14: **else if** k 番目の文字が‘)’ **then**
15: $rightPar \leftarrow rightPar + 1$.
16: **end if**
17: **if** $(leftPar - rightPar) = parSub$ **then**
18: **if** $visited = FALSE$ **then**
19: **if** $parSub \neq 1$ **then**
20: **if** $Check(rightCheckPos, leftCheckPos) = -1$ **then**
21: return FALSE;
22: **end if**
23: **end if**
24: $rightCheckPos \leftarrow k$.
25: $visited \leftarrow TRUE$.
26: **else if** $visited = TRUE$ **then**
27: $leftCheckPos \leftarrow k$.
28: $parSub \leftarrow parSub + 1$.
29: $visited \leftarrow FALSE$.
30: **end if**
31: **end if**
32: $k \leftarrow k - 1$.
33: **end while**
34: **end if**
35: return TRUE.

方向にたどっていく. 現れる括弧の数を数えていき, 開括弧の数が閉括弧の数よりも 1 個多くなる位置が 2 つ見つかるまで探し続ける. そのような位置が 2 つ見つかったとき, 式表現が標準形であるためには, 2 つ目に見つけた括弧に対応する部分木 $>$ 1 つ目の括弧に対応する部分木でなければならない. Check 関数を呼び出してこれを判定する. 条件を満たすならば, 2 つ目に見つけた括弧の位置から探索を再開し, 式の最初にたどり着くまで繰り返し行う. その際, 見つける位置の条件である開括弧と閉括弧の差を 1 ずつ増やしていくことによって, 次に大小関係を比較すべき場所が得られる. 式の最初までたどり着いたとき, グラフは標準形であることが判定できる.

Algorithm3 では, 葉の数のみが異なりうる 2 つの部分木の大小関係を調べる関数として Check, 与えられた開括弧に対応する閉括弧の場所を調べる関数として GetCorPar を用いる. Check のアルゴリズムを Algorithm 4 に, GetCorPar のアルゴリズムを Algorithm 5 に示す. Check は, 葉を子に持つノードの子の数を左のノードから比較していき, 大小関係を判定する. 左の部分木の方が大きければ 1, 等しければ 0, 右の方が大きければ-1 を返す. GetCorPar は, 与えられた開括弧の位置から式表現の右側へ, 1 文字ずつ開括弧と閉括弧の数をカウントしながら探索していく. 閉括弧の数が開括弧の数より大きくなったとき, 求める閉括弧が得られる.

Algorithm 4 葉の数が異なりうる 2 つの部分木の大小関係を調べる関数 Check

Input: $leftIndex, rightIndex$: 比較する部分の開始インデックス 2 つ.

1: $i \leftarrow leftIndex + 1$.
2: $j \leftarrow rightIndex + 1$.
3: **while** $i \neq GetCorPar(leftIndex)$ **do**
4: $leftVarNum \leftarrow 0$.
5: $rightVarNum \leftarrow 0$.
6: **while** i 番目の文字が‘)’でない **do**
7: **if** i 番目の文字が頂点 **then**
8: $leftVarNum \leftarrow leftVarNum + 1$.
9: **end if**
10: $i \leftarrow i + 1$.
11: **end while**
12: **while** j 番目の文字が‘)’でない **do**
13: **if** j 番目の文字が頂点 **then**
14: $rightVarNum \leftarrow rightVarNum + 1$.
15: **end if**
16: $j \leftarrow j + 1$.
17: **end while**
18: **if** $leftVarNum > rightVarNum$ **then**
19: return 1;
20: **else if** $leftVarNum < rightVarNum$ **then**
21: return -1;
22: **end if**
23: **end while**
24: return 0;

Algorithm 5 対応する閉括弧のインデックスを返す関数 GetCorPar

Input: $leftParIndex$: 開括弧のインデックス

1: $k \leftarrow leftParIndex + 1$.
2: $leftParNum \leftarrow 0$.
3: $rightParNum \leftarrow 0$.
4: **while** $k <$ 式表現の文字数 **do**
5: **if** k 番目の文字が‘(’ **then**
6: $leftParNum \leftarrow leftParNum + 1$.
7: **else if** k 番目の文字が‘)’ **then**
8: $rightParNum \leftarrow rightParNum + 1$.
9: **end if**
10: **if** $leftParNum < rightParNum$ **then**
11: return k;
12: **end if**
13: $k \leftarrow k + 1$.
14: **end while**
15: return -1;

(iv) の条件判定について, センターが 1 個の場合, v_1 がセンターから外れるのは, フロンティア上の何れか 2 つの頂点を端点とし, かつセンター頂点を中心点とするパスが存在しないときである. 図 3.10 では, 辺 (v_3, v_5) を採用しないとき, v_1 がセンターから外れてしまう. 現段階で既にセンターがずれており, 以降でどのように操作したとしても, ずれた中心を元に戻すことができないからである. 具体的には, v_1 から接続される辺を採用したときに生じた式表現中の “,”に注目することによって判定することができる. 辺 e を木の辺として採用しない場合, v_i が式表現に含まれており, かつ v_1 からの辺の採用によって生じた “,”が式表現に存在しないなら, v_1 がセンターから外れるので, (iv) の条件を満たさないことを判定できる. センターが 2 個の場合は, 辺 e を木の辺として採用する場合, “,”が式表現に存在しないならば, v_1 がセンターから外れてしまうので, (iv) を満たさないと判定できる.

また, 辺 $e = (v_i, v_n)$ を辺として採用したとき,(i)(ii)(iii)(iv) の条件を満たすと判定されたなら, 頂点数が n の根無し木が完成するので, ノードの x 枝の先を $\top$ に接続する.

(b) の configuration の更新では,Algorithm 6 に従って更新を行う.

処理中の辺 $e = (v_i, v_j)$ を追加するとき $(i < j)$, e が v_i に接続する初めての辺ならば, 2 行目の条件を満たし, 3,4 行目の処理に入る. e が v_i に接続する 2 本目以降の辺ならば, 6 から 10 行目の処理に入る. 6 行目の条件判定では, v_1 に接続する辺についての処理とそれ以外の頂点に接続する辺の処理とを判別して, センターとなる頂点を固定するために必要な処理を行っている. 辺 e を追加しないとき, v_i に接続する辺が 1 本もないならば, 13 行目の条件を満たし, 14,15 行目の処理に入る. v_i に接続する辺が 1 本もないことを, “v_i”から “()”への置き換えによって表す.

(c) の等価性判定は, 同じ段の 2 つのノード n, n' に割り当てられた式表現について, 式が一致するとき,n と n' は等価であると判定できる.

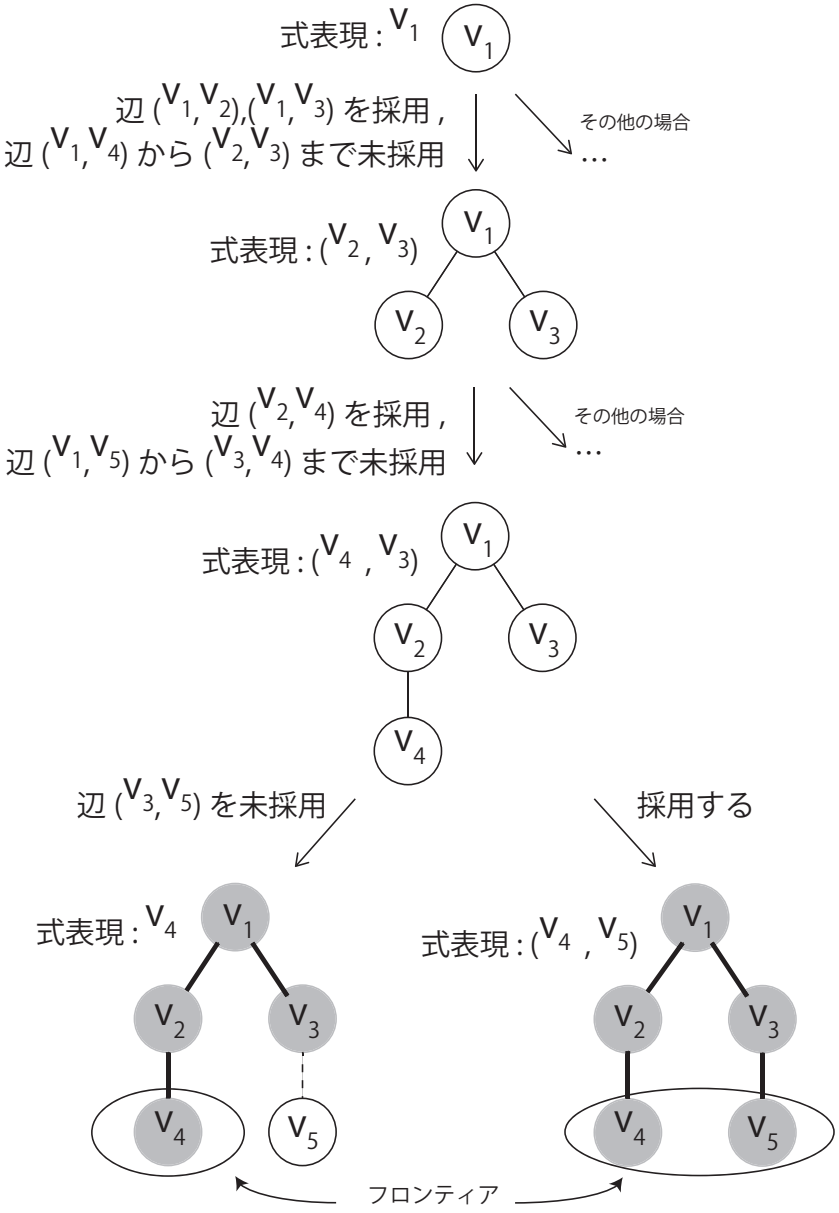

図 3.10: v_1 がセンターから外れてしまう場合

Algorithm 6 configuration の更新

if 処理中の辺 (v_i, v_j) を使う $(i < j)$ **then**
  if v_i が式表現中に存在 **then**
    v_i を削除.
    削除した場所に"(v_j)"を追加.
  else
    if $i = 1$ **then**
      v_{j-1} の後ろに"$, v_j$"を追加.
    else
      v_{j-1} の後ろに"v_j"を追加.
    end if
  end if
else
  if v_i が式表現中に存在 **then**
    v_i を削除.
    削除した場所に"()"を追加.
  end if
end if

4 おわりに

本研究により, フロンティア法を用いた非同型な根無し木を列挙するアルゴリズムを構築することができた. 提案したアルゴリズムを実装したプログラムが完成していないため, 早急に実装を行い, 評価を行いたい.

提案した手法では, センターを 1 つに固定する場合と, 2 つの場合とで 2 回実行する必要がある. 今後の課題としては, 1 回の実行で両方の場合を網羅できるようにすることが挙げられる.

フロンティア法を用いたグラフの列挙では, *mate* 配列や *part* 配列のように, configuration としてフロンティア上の各頂点に値を記憶させることが多い. 一方本研究では, confguration を ZDD の 1 つの途中状態に対して 1 つの式を持たせている. フロンティア上の各頂点に値を記憶させる方式の configuration を考えて, 計算量の比較を行うことも今後の課題として挙げられる.

また, ZDD の特長を生かして, 特定の条件を満たす解の抽出などを行えるのではないかと考えている.

謝辞

本研究を進めるにあたり, ご指導を頂いた指導教員の武永康彦准教授に感謝致します. また, 日常の議論を通じて多くの知識や示唆を頂いた武永研究室の皆様に感謝します.

参考文献

[1] S. Minato, “Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems”, In Proc. of 30th ACM/IEEE DAC, pp.272-277(1993).

[2] R. E. Bryant,“Graph-based algorithms for Boolean function manipulation,” IEEE Transactions on Computers, Vol. 35 pp.677-691(1986).

[3] 井上武 他, フロンティア法による電力網構成制御, オペレーションズ・リサーチ, Vol. 57, No. 11, pp.610-615(2012).

[4] E. Loekito, J. Bailey, “Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams”, KDD ’06 Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.307-316(2006)

[5] 湊真一, BDD/ZDD を用いたグラフ列挙索引化技法, オペレーションズ・リサーチ, Vol. 57, No. 11, pp.597-603(2012).

[6] D. Knuth, The Art of Computer Programming, Volume 4A, Addison Wesley(2011).

[7] S. Nakano, T. Uno, “A Simple Constant Time Enumeration Algorithm for Free Trees”, IPSJ SIG Technical Report, pp.9-16(2003).