令和 2年度　卒業論文

凸多面体の辺展開図における
自己重複確認アルゴリズムの高速化

令和 3年 2月

九州工業大学　情報工学部
システム創成情報工学科

17236024

塩田　拓海

指導教員：斎藤　寿樹

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景・目的

凸多面体（凹みが無く，各面が多角形である多面体）の辺展開図の研究は，1525 年 Albrecht Dürer（1471–1528，Nürnberg，ドイツ）が，“Underweysung der messung mit dem zirckel un richt scheyt” [1] （邦訳『測定教本』）を出版したことに起源するとされる [2]．この本の中には，私たちが「展開図」や「辺展開」と呼んでいるものが描かれている（以降，辺や面に切り込みを入れて平坦に開いた多角形を「展開図」，辺に沿って切り込みを入れることを「辺展開」，辺に沿って切った展開図を「辺展開図」と区別して表記）．しかし，どの辺展開図を見ても面どうしが重ならない多角形，つまり自己重複を持たないものとなっている．Dürer が意図的に自己重複を持たないように辺展開図を描いたという証拠は無いが，この「Dürer の描画」を通じて次の未解決問題が得られた．

未解決問題 1.1 (凸多面体の辺展開（ [2]，未解決問題 21.1 を参照）). すべての凸多面体は単純で重ならない多角形に辺展開することが出来るだろうか．つまり全ての凸多面体は辺展開図をもつだろうか．

この問題は，「Dürer の描画」以降，未だ未解決問題である．この問題を解決すべく，凸多面体に制約条件が付けられた研究が進められてきた．しかし，最も私たちに馴染みの深い正多面体（プラトンの立体，全 5 種）の辺展開図に関しても，自己重複を持たないかはごく最近まで分かっておらず，2011 年になって始めてどの辺展開図も自己重複を持たないということが分かった [3]．また，半正多面体（アルキメデスの立体，全 13 種）に関しては，2015 年に 5 種類の多面体が重なりを持たないということが分かったが [4]，残りの 8 種類のうち 5 種類に関しては自己重複は存在するが，その個数は分かっておらず [5]，3 種類に関しては自己重複の有無すらも分かっていない (表 1.1)．これは，自己重複の有無を 1 つ 1 つの辺展開図に対して，各面の (x, y) 座標の計算，どの 2 組の面を見ても自己重複を持たないかを判定するという手法を取っているからだ．つまり，辺展開図の数は面の数が増えるほど座標の計算の回数が膨大になり，1 つ 1 つの辺展開図に対して自己重複を確認していくことが非常に困難であるからだ．

本研究では，表 1.1 に示す「未解決」の箇所を解明すべく，凸多面体の重複を既存の方法より高速に判定することを目的とする．

表 1.1: 正多面体および半正多面体の辺展開図の個数および重複の有無（赤文字で示す凸多面体立体が本論文で扱う立体）

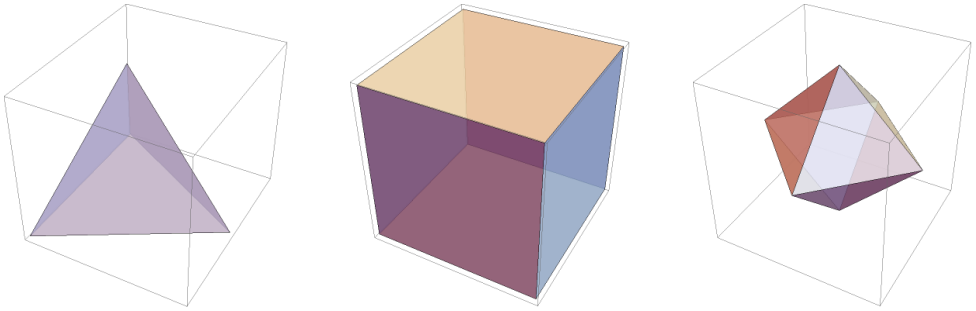
凸多面体	辺展開図の個数	重なりのある辺展開図の有無	重なりを持たない辺展開図の個数
正四面体	16	無	16
正六面体	384	無	384
正八面体	384	無	384
正十二面体	5,184,000	無 [3]	5,184,000
正二十面体	5,184,000	無 [3]	5,184,000
切頂四面体	6,000	無 [4]	6,000
切頂六面体	32,400,000	無 [4]	32,400,000
切頂八面体	101,154,816	無 [4]	101,154,816
切頂十二面体	4,982,259,375,000,000,000	有 [3]	未解決
切頂二十面体	375,291,866,372,898,816,000	有 [3]	未解決
立方八面体	331,776	無 [4]	331,776
二十・十二面体	208,971,104,256,000	未解決	未解決
斜方立方八面体	301,056,000,000	無 [4]	301,056,000,000
斜方二十・十二面体	201,550,864,919,150,779,950,956,544,000	有 [3]	未解決
斜方切頂立方八面体	12,418,325,780,889,600	未解決	未解決
斜方切頂二十・十二面体	21,789,262,703,685,125,511,464,767,107,171,876,864,000	有 [3]	未解決
変形立方体	89,904,012,853,248	未解決	未解決
変形十二面体	438,201,295,386,966,498,858,139,607,040,000,000	有 [6]	未解決

1.2 本論文の成果

本論文では，凸多面体のうち既に重ならないと判定されている正多面体の一部（正四面体，正六面体，正八面体）や半正多面体の一部（頂切四面体および立方八面体）(表 1.1 で赤文字で記した多面体，図 1.1) を対象の図形とした．そして，計算の時間が嵩む原因である，各面の (x, y) 座標の計算の回数およびどの 2 組の面を見ても自己重複を持たないかを判定する回数を削減することで，自己重複の確認の時間を最大で 7.9%短縮することができ，高速化に成功した．

1.3 本論文の構成

本論文の構成は以下の通りである．第 2 章では本論文で使用する諸概念を説明する．第 3 章では自身が考案した高速化した自己重複の確認アルゴリズムを説明する．第 4 章では実験条件および，考案したアルゴリズムが既存研究と比べ，どれほど高速に判定することが出来るようになったかについて述べる．最後に第 5 章で本論文のまとめおよび今後の課題について述べる．



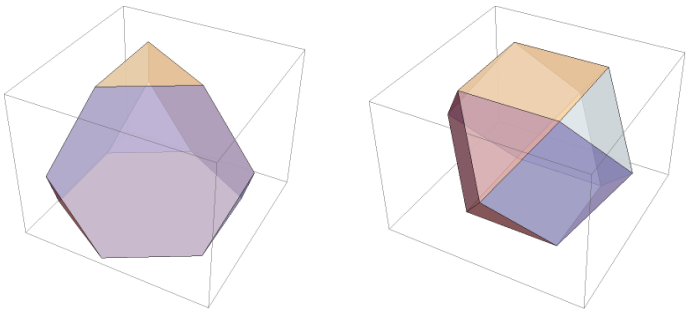


図 1.1: 本論文で扱う凸多面体

第2章　準備

2.1　グラフ

頂点の集合を V，辺の集合 $E \subseteq V \times V$ で構成されたものをグラフ $G = (V, E)$ と表記する．連続する 2 つの頂点の間に辺があるような頂点の列を**道（パス）**という．グラフの任意の 2 頂点 v_i, v_j $(v_i, v_j \in V)$ 間に頂点と頂点を繋ぐ道が存在するならばグラフは**連結**であるといい，始点と終点が同じであるならばグラフは**閉路**であるという．また，連結で閉路を持たないグラフを**木**といい，グラフ $S(V, E_T)$ が木で $E_T \subseteq E$ であるならば，$S(V, E_T)$ のことをグラフ $G(V, E)$ の**全域木**という．

2.2　凸多面体の辺展開

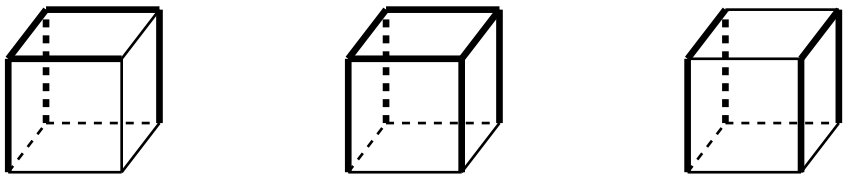
まず正六面体を例に説明する．図 2.1 に示す太線が引かれた辺に沿って辺を切るとき，正六面体は辺展開することが出来ないことがわかる．以降，辺を切ることを**カット**，カットされた辺のことを**カット線**と呼ぶ．正六面体のように凸多面体の面の数が少ないうちは，どのカット線を選択したら，どのような辺展開図となるかということは容易に想像することが出来る．しかし凸多面体の面の数が増えると，どのカット線を選択するかを与えられただけでは，どのような辺展開図となるかを想像するのは非常に難しくなる．そこでグラフの概念を取り入れ，凸多面体の頂点の集合をグラフの頂点集合 V，凸多面体の辺の集合をグラフの辺集合 E としグラフ問題とすることで抽象的に解くことが可能となる．ここで凸多面体 Q を切り開いた時のカット線の集合を C とするとき，次の定理が知られている．

定理 2.1 (辺展開図の基本的な性質（ [7], 定理 2.1.1）)．カット線の集合 C は，Q 上のすべてを頂点をつなぐ全域木である．

この定理より，凸多面体の辺展開図の数え上げを行うためには，凸多面体における全域木の数を求めればよい．

2.3　辺展開図の自己重複および自己重複の判定方法

いくつかの凸多面体は，特定の方法で辺展開すると，自己重複を持つということが知られている [5]．その例を図 2.2 に示す．(a) では正三角形と正十角形が，(b) では正五角形と正六角形が，(c) では正方形と正三角形がそれぞれ重なっている．辺展



(a) 全ての頂点を通らない **(b)** 閉路を持つ **(c)** ひとつながりではない

図 2.1: 辺展開が出来ないカットの例

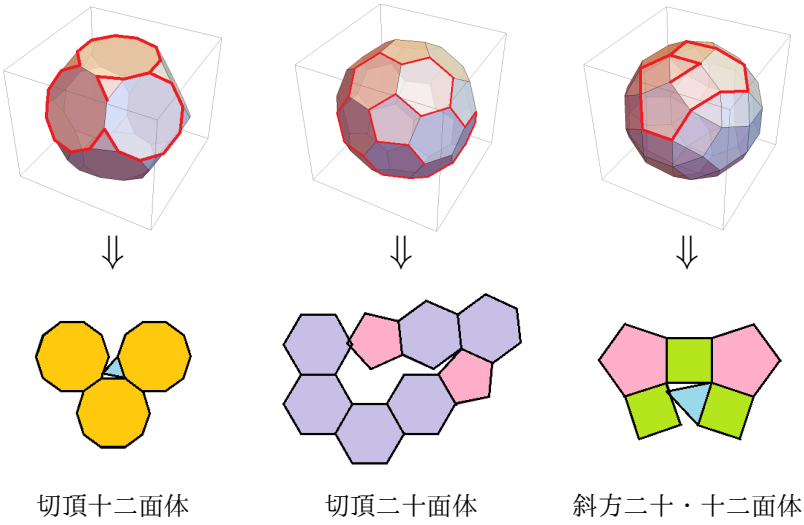


図 2.2: 自己重複を持つ凸多面体の例（各凸多面体を赤色の辺に沿ってカットすると，それぞれ下に示す展開図となる）

開図を描画すれば重複を確認できるが，研究の背景でも述べた通り，辺展開図の数が多く全ての辺展開図を目視で確認していくことは不可能である．そこで，命題 2.1 を考える．

命題 2.1 (面と外接円との関係性)**.** ある 2 つの面が与えられたとき，それぞれの面の外接円が重なりを持たないならば，面どうしは重なりを持たない．

この命題の対偶「ある 2 つの面が与えられたとき，面どうしが重なりを持つならば，それぞれの面の外接円が重なりを持つ」は真であることは明かである．ゆえに，この命題は正しい．なお，この命題の裏「ある 2 つの面が与えられたとき，それぞれの面の外接円が重なりを持つならば，面どうしは重なりを持つ」は常に成り立つとは限らない．

なお円どうしが重なるかに関しては，図 2.3 のように円 A の半径を d_A，円 B の半径を d_B，円 A と円 B の中点間距離を d とし，式 (2.1) を計算することで判定することが出来る（本論文では円どうしが接する状態は，重なるものとして考える）．また，この判定式はアルゴリズムとすると **Algorithm 1** のように書くことが出来る．

$$\begin{cases} \text{重ならない} & (d_A + d_B < d) \\ \text{重なる} & (otherwise) \end{cases} \tag{2.1}$$

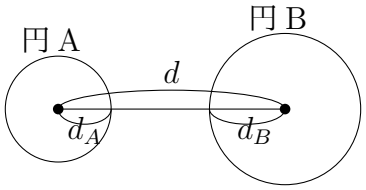


図 2.3: 面の重なる条件

Algorithm 1 円どうしの重なりの確認

Input: 円 A の中心座標 (x_A, y_A)，半径 d_A，円 B の中心座標 (x_B, y_B)，半径 d_B

1: $d = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
2: **if** $d_A + d_B < d$ **then**
3:　　**print** 重ならない
4: **else**
5:　　**print** 重なる
6: **end if**

図 2.4 に示す辺展開図を例に自己重複を持つかどうかを判定していく．自己重複が生じる可能性があるのは，図 2.2 の例のように面どうしが同じ辺や頂点を持たない場合である．以降，凸多面体を辺展開したとき面と面が同じ辺もしくは同じ頂点を 1 つ以上持つ場合を**隣接する面**，それ以外の場合を**隣接しない面**とし，面番号 A と面番号 B の面の組みを (A,B) と呼ぶ．例えば，図 2.4 の例の場合，隣接しない面の組の集合は $\{(1,5),(1,6),(1,7),(1,8),(2,5),(2,6),(2,8),(3,7),(4,5),(4,6),(4,7),(4,8),(5,6),(5,7),(6,7),(7,8)\}$ の 16 個である．続いて，命題 2.1 を用いるために，図 2.5 に示すように辺展開図の各面に対する外接円を考える．この際，数値計算誤差が生じてしまうことを予め踏まえた上で，外接円の半径を $+\Delta$ 程度大きく設定する（Δ の大きさは各辺展開図の数値誤差に応じて変わる）．最後に，隣接しない面の集合に対して **Algorithm 1** を適用することで，図 2.4 の辺展開図は自己重複を持たないと判定することが出来る．

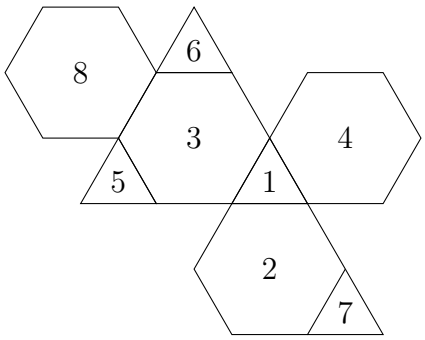


図 2.4: 切頂四面体の辺展開図の例

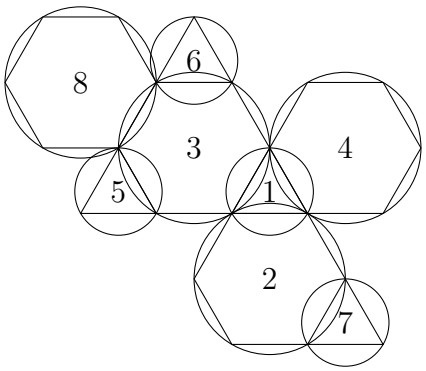


図 2.5: 展開図の各面に対する外接円

2.4 BDD・ZDD

BDD(Binary Decision Diagram：二分決定グラフ，[8]) は，論理関数を非巡回有向グラフ (DAG) で表現したデータ構造である．各節点は変数のラベルを持ち，内部節点は 0–枝と 1–枝を持っている．0–枝を通る道は枝の始点の節点変数を 0 に割り当て，1–枝を通る道は枝の始点の節点変数を 1 に割り当てる．また，終端節点 (節点のうち子をもたない節点) は，0 か 1 の値を持つ．根から終端節点までの道が論理関数の値と対応しており，終端節点が 0 ならば論理関数の値は 0，終端節点が 1 ならば論理関数の値は 1 となる.

ZDD(Zero-suppressed Binary Decision Diagrams：ゼロサプレス型二分決定グラフ，[9]) は，組合せ集合 (集合族) をコンパクトに表現することができ，疎なデータに対して，特に有効性を発揮する BDD である．各変数を集合の要素に対応させ，その要素を組合せに含めないのであれば 0–枝を辿り，その要素を組合せに含めるのであれば 1–枝を辿る．そして，根から 1–終端節点までのパスが 1 つの組合せに対応する. ZDD は BDD を次の簡約化規則 [10, p.20] に従い圧縮することで得られる．

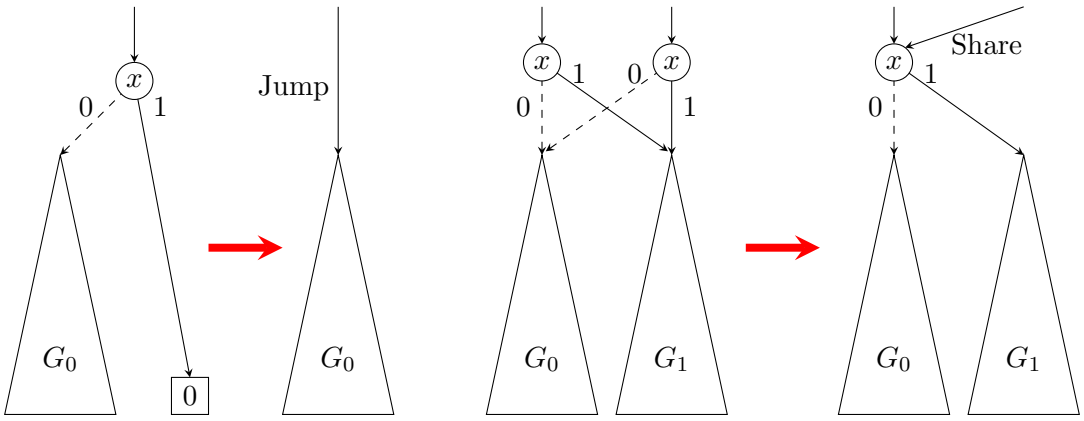
(a) 冗長頂点の削除 1–枝が 0–枝の値を持つ葉を指している場合に，この節点を取り除き，0–枝の行き先に直結させる（図 2.6(a)）.

(b) 等価頂点の共有 等価な節点（要素が同じで，0–枝どうし，1–枝どうしの行き先が同じ）を共有する（図 2.6(b)）.

2.5 フロンティア法

フロンティア法とは，節点の作成中に節点どうしの共有が行えるかを順次判断していき，可能な限り節点の共有を行うことで，作成される節点の数を減らし高速に ZDD を構築する手法である [11].

例として，図 2.7(a) に対する全域木を考える．ここで全域木の探索を BDD で行っている最中のグラフの例を図 2.7 の (b) と (c) に示す．図 2.7 で，破線の楕円で塗ら



(a) 冗長節点の削除　　**(b)** 等価節点の共有

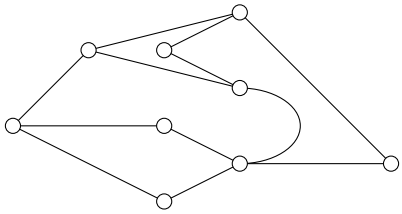
図 2.6: ZDD の圧縮規則

れた部分の左側に描かれた辺は処理済み，右側に描かれた辺は未処理である．(b) と (c) のグラフを見ると，どちらも e_a を使って e_b を使う，e_a を使って e_c を使う，または e_b を使って e_c を使うとき全域木が完成し，それ以外の辺の選び方をした場合は全域木が完成しないことがわかる．また，楕円に含まれる 3 点のうちの上の 2 頂点は赤色の成分に連結しており，下の 1 頂点は青色の成分に連結されている．このように，楕円に含まれる全ての頂点が同じ連結成分であるならば，楕円の右側で辺の選択をするときに，左側の選択の仕方を記憶しておく必要はない．このような楕円の中に含まれる頂点の集合をフロンティアと言う．つまり，フロンティアとは「処理済みの辺と未処理の辺の両方と接続している頂点の集合」のことである．

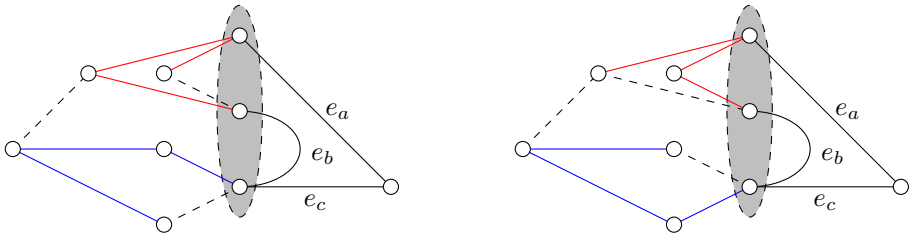
フロンティアにおける連結成分を把握するためには，各節点がどの連結成分に属しているかを状態を記憶しておくために，mate と呼ばれる配列を用意する．そして，フロンティアにおける mate 配列が等価ならば，等価な節点として 1 つにまとめるのである．

続いて，フロンティア法を用いた全域木の選択する辺の列挙を行う．そこで，まず ZDD の構築の仕方を擬似コードの形式で **Algorithm 2** に示す（本アルゴリズムは [10] **Algorithm 4.1**（s-t 経路の数え上げ）を参考にした．一部処理が異なる）．

Algorithm 2 の 1 行目は，配列のサイズ $m+1$ の確保，2 行目から 17 行目の for 文は木の各深さに関する処理，3 行目から 16 行目のの for 文は深さにおける各節点に関する処理である．4 行目から 15 行目の for 文は 0–枝（全域木の辺として使わない），1–枝（全域木の辺として使用する）の先にある節点を付け足していく処理である．5 行目で n_i の mate 配列の情報を上書きしてしまわないよう n' にコピーしておき，6 行目で n' の mate 配列の更新をし（枝刈りが生じた場合は 7 行目から 14 行目の処理は実行しない），7 行目から 11 行目の if 文で共有するかどうかを決める．そして，12 行目で節点がフロンティアから抜けるの時の枝刈りの判定をした時に枝刈りをされず，13 行目で終了条件の判定をした時に終了しなければ，14 行目で 1 つ下の深さの節点を指すように処理する．



(a) もとのグラフ



(b) 途中まで辺を処理したグラフ (1)　　**(c)** 途中まで辺を処理したグラフ (2)

—— ——：処理済の辺（使った）　- - -：処理済の辺（使わなかった）　——：未処理の辺

図 2.7: 全域木を列挙する途中状態のフロンティア

Algorithm 2 は mate 配列の取り方および (a)，(b)，(c) の条件を変えることで様々な構造の問題に対する選択する辺の列挙が出来る．今回 $G = (V, E)$ の部分グラフ $S(V, E_T)$ が全域木であるためには，mate 配列の取り方および (a)，(b)，(c) の条件を次のようにすればよい．

mate 配列の取り方

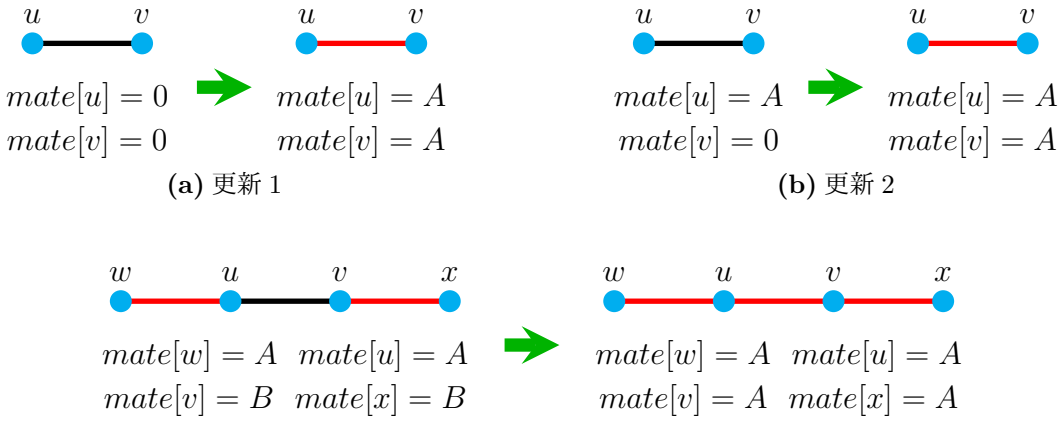
mate 配列は，各節点が属する連結成分（$1 \cdots m + 1$，以下 ID と表記する）を記憶する．初期状態では，どの節点も連結していないので mate$[i]$ の ID を全て 0 と設定する．この mate 配列を更新していくことでどの節点が同じ連結成分に含まれているかを管理していき，最終的には全ての節点が同じ ID，つまり 1 つの連結成分に含まれるようになることで全域木を形成していく．

（a）n' の mate 配列の更新

Algorithm 3 に従い mate 配列の更新をする．**Algorithm 3** の 1 行目から 2 行目は辺の両端点の ID が一度も更新されていない場合の更新である（図 2.8(a)）．この更新では新たな連結成分が生じることとなるため，新規で ID を作成し両端点ともに同じ連結成分に分類する．3 行目から 6 行目は一方の端点の ID が一度も更新されておらず，もう一方の端点が更新されたことがある場合の更新である（図 2.8(b)）．この更新は既存の連結成分に新しく成分を加える操作である．7 行目から 17 行目は両端点ともに ID が割り当てられている場合の更新である（図 2.8(c)）．このとき両

Algorithm 2 フロンティア法による ZDD の構築

1: $N_1 \leftarrow \{n_{root}\}$, $N_i \leftarrow \emptyset$ for $i = 2, 3, \cdots, m+1$
2: **for** $i = 1$ to m **do**
3: **foreach** 節点 $n_i \in N_i$ **do**
4: **foreach** $x \in \{0, 1\}$ **do**
5: 新しいノード n' を作成．n' に n_i をコピー．
6: （a）n' の mate 配列の更新．
7: **if** n' と等価な $n'' \in N_{i+1}$ が存在． **then**
8: $n' \leftarrow n''$
9: **else**
10: $N_{i+1} \leftarrow N_{i+1} \cup \{n'\}$
11: **end if**
12: （b）節点がフロンティアから抜けるの時の枝刈りの判定．
13: （c）終了条件の判定．
14: n_i の $x-$ 枝を作成．n' を指す．
15: **end for**
16: **end for**
17: **end for**



(a) 更新 1　(b) 更新 2

(c) 更新 3

図 2.8: mate 配列の更新

Algorithm 3 mate 配列の更新

```
1: if mate[u] = 0 and mate[v] = 0 then
2:    mate[u] および mate[v] の ID を新規で作成した ID に更新.
3: else if mate[u] = 0 and mate[v] ≠ 0 then
4:    mate[u] 側の連結成分の ID を mate[v] 側の ID に更新.
5: else if mate[u] ≠ 0 and mate[v] = 0 then
6:    mate[v] 側の連結成分の ID を mate[u] 側の ID に更新.
7: else if mate[u] ≠ 0 and mate[v] ≠ 0 then
8:    if mate[u] = mate[v] then
9:       閉路が生じた → 枝刈り
10:   else if mate[u] ≠ mate[v] then
11:      変数 tmp に mate[v] の ID を記憶
12:      for i = 1 to m + 1 do
13:         if mate[i] = tmp then
14:            mate[i] = mate[u]
15:         end if
16:      end for
17:   end if
18: end if
```

端点の ID が (0 以外で) 等しいならば閉路が生じてしまうため，9 行目から 10 行目その枝刈りをする．12 行目から 16 行目では連結成分どうしの結合をする．片方の連結成分の ID を更新したとき，その連結成分に更新前から結合していた節点の ID も同じ ID に更新する必要がある．

（b）節点がフロンティアから抜けるの時の枝刈りの判定

ある節点がフロンティアから抜けると，その節点は図 2.8(c) の更新 3 による更新以外行われなくなる．つまりフロンティアから抜けた段階で節点がどの連結成分にも含まれていなければ今後連結成分に含まれることは起き得ない．ゆえに節点がフロンティアから抜けたとき，mate の値が 0 のままであるならば枝刈りをする．

（c）終了条件の判定

Algorithm 2 の 2 行目の for 文の i が m までループを終えたときこのアルゴリズムは停止する．このとき，全域木であれば全ての節点における連結成分の ID が同じであるが，複数の連結成分が生じてしまう場合も生じる．つまり連結成分の ID が複数存在する状態になると全域木は形成されない．ゆえに全ての節点における mate の値が同じでなければ枝刈りをする．

Algorithm 4 凸多面体の構成の構築

Input: カット線の集合 C

1: **for** $i = 1$ to 面の数 **do**
2:　**for** $j = 1$ to 面 i を構成する辺の数 **do**
3:　　$\mathrm{adj}(i, j) = -1$
4:　**end for**
5:　**for** $k =$ 面を構成する辺 $\setminus C$ **do**
6:　　$\mathrm{adj}(j, k) =$ 隣り合う面
7:　**end for**
8:　**for** $l = 1$ to 面 i を構成する辺の数 **do**
9:　　$\mathrm{ring}(i, l) \leftarrow$ 面 i を構成する辺（時計回り）
10:　**end for**
11: **end for**

2.6　辺展開図の各面の中心座標の計算

辺展開図の各面の中心座標を求める前に対象とする凸多面体の構成を構築しておく必要がある．**Algorithm 4** に擬似コードの形でその構築方法を示す．

F を面の集合，E を辺の集合とするとき，関数として $\mathrm{adj} : F \times E \to F$ を定義し，面同士の隣り合うかどうかの関係を表す．例えば面 0 と面 4 が辺番号 3 の辺を共通して持つ，つまり面 0 と面 4 が辺番号 3 の辺を挟んで隣り合うとき，$\mathrm{adj}(0, 3) = 4$，$\mathrm{adj}(4, 3) = 0$ と表記する．また N を面 F を構成する辺の本数，S を面 F を構成する辺の集合とするとき，関数 $\mathrm{ring} : F \times N \to S$ を定義し，面を構成する辺を格納していく．なお，ここでは「隣り合う面」を凸多面体を辺展開したとき面と面が同じ辺を持つことを指し示し，2 章で定義した「隣接する面」と区別して考える．

Algorithm 4 の 2 行目から 4 行目では，全ての面どうしが隣り合わないとし初期化をしている．5 行目から 7 行目では非カット線を辺として共通に持つ面の組み，つまり隣り合う面の組みを記述する．8 行目から 10 行目では各面を構成する辺を時計回りに記述する．

構築した凸多面体の構成をもとに，各面の中心座標を計算する．**Algorithm 5** に擬似コードの形でその計算方法を示す．**Algorithm 5** の 2 行目から 6 行目では，8 行目以降でリングバッファを使用するために，注目している辺番号は，**Algorithm 4** の ring 配列で確保した辺のうち，どの辺を指し示しているかを変数 k に記憶する．7 行目に関しては 10 行目で説明する．8 行目から 21 行目では面を構成する辺を辿っていき，再起呼び出しを行うことで座標の計算をしていく．9 行目は配列に確保されている次の辺に着目し，10 行目では注目している辺が変わると角度が変わるため d 更新をしている．角度を半時計回りに 1 度回しておかないと，$i = k$ つまり今注目している辺の角度がズレてしまうため 7 行目の操作が必要となる．11 行目から 20 行目では，隣り合う面が存在する場合を考える．13 行目から 19 行目では隣り合う面による場合分けを考えている．今回のアルゴリズムでは，凸多面体を構成する面の

Algorithm 5 calc_a 関数

Input: 注目している面番号，注目している辺番号，注目している面の中心座標 (x, y)，注目している面の内接円の半径，注目している面の中心座標から注目している辺の角度 d

```
1: 面の中心座標 (x, y) を出力
2: for i = 1 to 面を構成する辺の数 do
3:   if ring(面番号, i) = 辺番号 then
4:     k = i
5:   end if
6: end for
7: d = d + (360 /面を構成する辺の数)°
8: for i = k to (k + 面を構成する辺の数 − 1) do
9:   注目している辺番号 e = ring(面番号, i %面を構成する辺の数)
10:  d = d − (360 /面を構成する辺の数)°（時計回りに回転）
11:  if 隣り合う面 adj(面番号, i) ≠ −1 and 隣り合う面 s が未使用 then
12:    隣り合う面 s を使用済みとする
13:    if 隣り合う面が同じ多角形 then
14:      隣り合う面の内接円の半径 r′，(x′, y′) 座標を計算
15:      calc_a(s, e, (x′, y′), r′, d − 180)
16:    else if 隣り合う面が異なる多角形 then
17:      隣り合う面の内接円の半径 r′，(x′, y′) 座標を計算
18:      calc_b(s, e, (x′, y′), r′, d − 180)
19:    end if
20:  end if
21: end for
```

種類が 2 種類の場合を書いているが，1 種類の場合は **else if** の箇所が必要なく，3 種類以上の場合が **else if** の項目を増やせばよい．14 行目では，(x', y') の座標を求めるために，隣り合う内接円の半径をしたのち，(x', y') の座標を求める．元の (x, y) 座標，角度 d は既に求まっており，半径は $r + r'$ で求めることが出来るため，(x', y') の座標は次の式で表すことが出来る．

$$\begin{aligned} x' &= x + (r + r') * \sin(d * \pi/180) \\ y' &= y + (r + r') * \cos(d * \pi/180) \end{aligned} \tag{2.2}$$

15 行目では関数を再帰で呼び出す．ここで d を 180° 回転させているが，これは，注目している辺は変わってないが，注目している面の中心座標が注目している辺を挟んで反対側へと移動しているからである．17 行目，18 行目に関しても同様である．

第3章　辺展開図の自己重複の確認

3.1　自己重複確認アルゴリズム

既存研究における凸多面体の辺展開図の自己重複を確認するための流れを下記に示す.

(1). 辺の本数，頂点の個数，各辺の両端点をグラフデータファイルから読み込む

(2). フロンティア法を用いてカット線を列挙する（2.5 節）

(3). 各面の中心座標を計算する（2.6 節）

(4). 隣接しない面を計算する（2.3 節）

(5). 重複の有無を確認する（2.3 節）

本研究では (3)，(4) の項目の計算回数を削減することで，アルゴリズムの高速化に成功した.

3.2　カット線と辺展開図の関係性

本論文では (3)，(4) の項目の計算回数の削減をするために用いる性質を，辺展開図の描画結果から見出した．辺展開図を描画は，2.6 節の **Algorithm 5** を書き換えることで出来る．**Algorithm 6** に擬似コードの形で描画の仕方を示す．**Algorithm 6** の 1 行目から 5 行目は，**Algorithm 5** の 2 行目から 6 行目の処理と同様である．6 行目から 20 行目では，面を構成する辺を辿っていき，再帰呼び出しを行うことで描画をする．7 行目では (x_1, y_1) と (x_2, y_2) の間に直線を描画する処理をする．8 行目から 16 行目は **Algorithm 5** の 11 行目から 20 行目の処理と同様である．17 行目では回転行列（式 (3.1)，図 3.1）を用いて次の辺を描くための座標を回転させる処理をする.

$$\begin{pmatrix} x_1' \\ y_1' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \tag{3.1}$$

Algorithm 6 に基づき描画した切頂四面体の辺展開図の一部を図 3.2 に示す（赤色の丸印が座標 $(0, 0)$ で，この面が「面 0」を表す，なお [12] に各多面体における展開図を描画した結果をまとめた）．各辺展開図に対して，2.3 節で示した判定を用い

Algorithm 6 Unfoldoing_a 関数

Input: 注目している面番号，注目している辺番号，注目している辺の左端点の座標 (x_1, y_1)，注目している辺の左端点の座標 (x_2, y_2)

for $i = 1$ to 面を構成する辺の数 **do**
  if ring(面番号, i) = 辺番号 **then**
    $k = i$
  end if
end for
for $i = k$ to $(k +$ 面を構成する辺の数 $- 1)$ **do**
  print "\draw (x_1, y_1) -- (x_2, y_2);"
  注目している辺番号 e =ring(面番号, i % 面を構成する辺の数)
  if 隣り合う面 adj$(i, j) \neq -1$ **and** 隣り合う面 s が未使用 **then**
    隣り合う面 s を使用済みとする
    if 隣り合う面が同じ多角形 **then**
      Unfolding_a $(s, e, (x_1, y_1), (x_2, y_2))$
    else if 隣り合う面が異なる多角形 **then**
      Unfolding_b $(s, e, (x_1, y_1), (x_2, y_2))$
    end if
  end if
  辺を回転させて $(x_1, y_1), (x_2, y_2)$ を更新.
end for

ることで，自己重複の有無を確認することが出来る．しかし，凸多面体を構成する面が増えると辺展開図の個数が多くなるだけでなく，判定すべき面の組みも増えてしまい判定が困難になることが予想される．

そこで図 3.3 のように図 3.2 に対して色付けを行い座標の位置が変わらない面に注目してみる．例えば図 3.2 の (1)〜(6) の辺展開図に注目してみると，桃色で塗られた面の集合どうしでの自己重複の確認は (1) のみで行えばよく，(2)〜(6) では何も塗られていない面と，桃色で塗られた面の集合との面の組みの，自己重複の有無の判定を行えば良いことが分かる．(7)〜(12)，(13)〜(17)，(18)〜(23) の辺展開図に関しても同様のことが言える．この性質を利用して色が塗られている面の集合の，座標

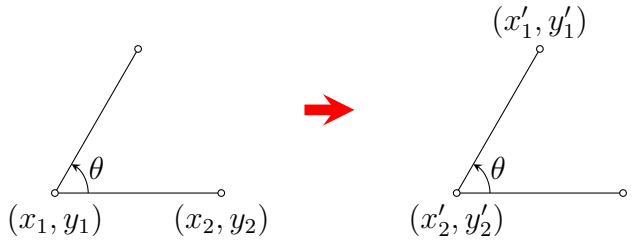


図 3.1: 辺の回転

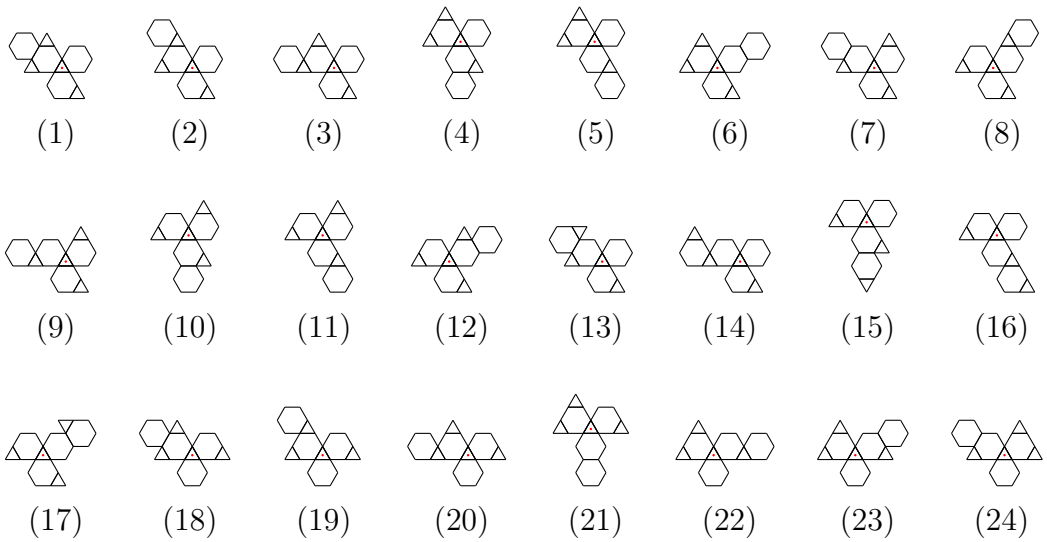


図 3.2: 切頂四面体の辺展開図の一部

や重複の確認の計算は，初めの 1 回 (色が切り替わった時) だけ行い，塗られていない面の集合の座標や重複の確認だけを各辺展開図で計算することで，計算時間の高速化を図る．

各面の中心点を節点とし，隣り合う面の節点どうしを辺で繋いだグラフを**双対グラフ**と呼ぶ．各辺展開図の双対グラフを考える．辺展開図番号 (i) と辺展開図番号 $(i+1)$ の双対グラフの最大共通部分グラフ G' を求め，辺展開図番号 $(i+2)$ 以降の双対グラフに G' が含まれるかどうかをみていく．もし，辺展開図番号 (j) の双対グラフには G' が含まれないのであれば $i = j$ とする．そして，辺展開図番号 (i) と辺展開図番号 $(i+1)$ の最大共通部分グラフ G'' を求め，辺展開図番号 $(i+2)$ 以降の双対グラフに G'' が含まれるかどうかをみていく．この操作を辺展開図番号の最後まで繰り返しいく．この操作は図 3.3 で同じ色に塗るという操作に該当する．ここで示した最大共通部分グラフ G' に該当する辺展開図のことを**基本形**と呼ぶ．

双対グラフの最大共通部分グラフを求めるにあたり，辺展開図はカット線の集合に基づき計算されているため，カット線の集合との関係を見てみた．すると次の法則が見えてきた．基本形以外の面の集合を**基本形以外の面**，基本形以外の面が別の座標に移り変わることを**面の移動**と定義する．また，基本形から基本形以外の面が計算出来ない場合や，基本形が変わらず，基本形以外の面の形が変わった場合については，基本形が変わるとして扱うものとして定義する．

法則 i 番目の辺展開図のカット線の集合と，$i+1$ 番目の辺展開図のカット線の集合の対称差集合を取るとき

(A) 対称差集合の要素数が 2 個であれば，一部例外を除き基本形は変わらない

(B) 対称差集合の要素数が 4 個以上であれば基本形が変わる

法則 (A) を補題 3.1 の一般形で表す．図 3.3 の (1) と (2) の辺展開図（基本形は変

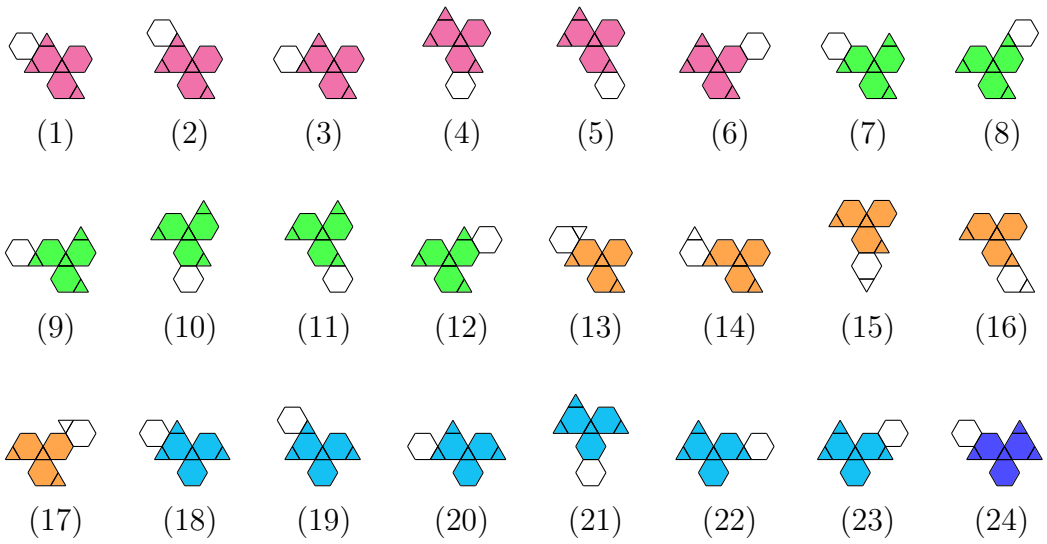


図 3.3: 色付けされた切頂四面体の辺展開図の一部

わらない）がこれに該当する．

補題 3.1 (法則 (A) の一般形)．対称差集合を $\{y_a, y_b\}$ とする．(a) の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_a\}$ ($j =$ カット線の本数 $- 1$)，(b) の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_b\}$ とする．このとき一部例外を除き基本形は変わらない．

証明． 補題 3.1 を対称差集合の要素から証明する．面 A に対して面 B が隣り合う面である，もしくは面 B が新たに隣り合う面になることを**面が繋がる**と呼び，面 C と面 D が隣り合う面の関係であったのが，隣り合わなくなることを**面を切り離す**と呼ぶ．立体を見てみると，面を構成する辺に「辺番号 y_a」を持つ面の集合は $\{F_{s_1}, F_{s_2}\}$，面を構成する辺に「辺番号 y_b」を持つ面の集合は $\{F_{s_1}, F_{s_3}\}$ であることが分かる．この積集合を取ると，$\{F_{s_1}\}$ が出てくる．このことから，(a) の辺展開図では「面 F_{s_1}」が「面 F_{s_2}」に繋がっていたが，(b) の辺展開図となると「面 F_{s_1}」と「面 F_{s_3}」の間は切り離され，変わって「面 F_{s_1}」は「面 F_{s_3}」に繋がる．つまり「面 F_{s_1}」が移動するという処理のみをすればよく，基本形が変わらないということが言える． □

補題 3.1 は，移動する面が 1 個の場合に限らず，移動する面が 2 面以上の場合についても同様のことが言える．その際，先ほどの「面 F_{s_1}」に加え複数の面が移動するということになる．そこで，双対グラフを取り「面 0」を根とする根付き木を考える．すると「面 F_{s_1}」に加え，その子孫にあたる面が移動するという言い換えることが出来る．ゆえに，この場合も補題 3.1 が成り立つ．

続いて法則 (B) を補題 3.2 の一般形で表す．図 2.3 の (6) と (7) の辺展開図（基本形が変わる）がこれに該当する．

補題 3.2 (法則 (B) の一般形)．対称差集合を $\{y_1, \cdots, y_l, z_1, \cdots, z_l\}$ の $(l \times 2)$ 個とする．(c) の辺展開図のカット線の集合を $\{x_1, \cdots, x_k, y_1, \cdots, y_l\}$ ($k + l =$ カット線の本数)，(d) の辺展開図のカット線の集合を $\{x_1, \cdots, x_k, z_1, \cdots, z_l\}$ とする．このとき基本形はが変わる．

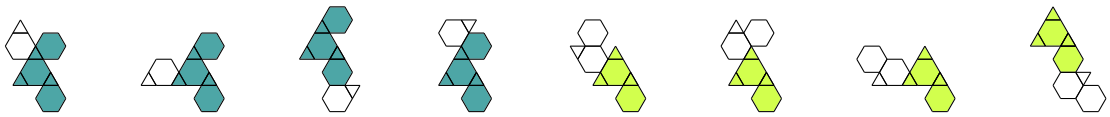


図 3.4: 切頂四面体の辺展開図の一部（法則 (A) の例外 1）

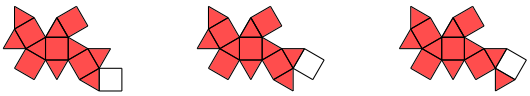


図 3.5: 立方八面体の辺展開図の一部（法則 (A) の例外 2）

証明. 補題 3.2 の証明をする．法則 (A) は 1 箇所のみ切る箇所を変える，つまり辺展開図を基本形と基本形以外の面に分割し，基本形以外の面を動かすという操作だった．しかし，法則 (B) は 2 箇所以上切る箇所を変える，基本形もしくは基本形以外の面のいずれかをさらに分割しなければならない．ゆえに基本形の定義に反し基本形が変わることが言える． □

最後に法則 (A) の例外となる場合を示していく．例外となる場合は 3 つある．まず補題 3.3 に 1 つ目の例外を示す．図 3.4 の (479) と (480) の辺展開図（基本形が変わる）がこれに該当する．

補題 3.3 (法則 (A) の例外 1)**.** 対称差集合を $\{y_d, y_e\}$ とし，(d) の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_d\}$，$(e)$ の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_e\}$ とする．「辺番号 y_d」を持つ面の集合を $\{F_{d_1}, F_{d_2}\}$，面を構成する辺に「辺番号 y_e」を持つ面の集合を $\{F_{e_1}, F_{e_2}\}$ とする．いま $F_{d_1} = F_{e_1}$，$F_{d_1} = F_{e_2}$，$F_{d_2} = F_{e_1}$，$F_{d_2} = F_{e_2}$ のいずれも成り立たないとする．このとき基本形が変わる．

証明. 補題 3.3 の証明をする．補題 3.1 では，$\{F_{s_1}, F_{s_2}\}$ と $\{F_{s_1}, F_{s_3}\}$ の積集合を取ることで $\{F_{s_1}\}$ を移動する面として導き出した．しかし補題 3.3 では積集合を取っても $\emptyset$ となるため，どの面が移動したかを定めることが出来ない．よって，基本形から基本形以外の面を計算することが出来ないため，定義に基づき基本形が変わる． □

次に補題 3.4 に 2 つ目の例外を示す．図 3.5 の (27) と (28) の辺展開図（基本形が変わる）がこれに該当する．（切頂四面体の場合このようなケースが生じないため，立方八面体を例とする）．

補題 3.4 (法則 (A) の例外 2)**.** 対称差集合を $\{y_f, y_g\}$ とし，(f) の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_f\}$，$(g)$ の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_g\}$ とする．非カット線の集合のうち基本形に含まれる部分集合を $\{w_1, \cdots, w_j, y_g\}$ とする．このとき基本形が変わる．

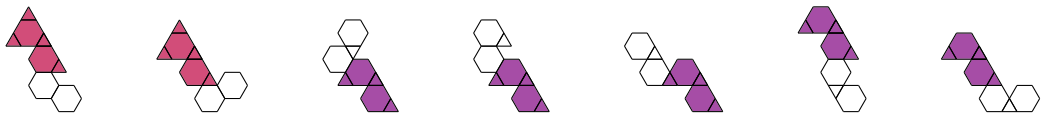


図 3.6: 切頂四面体の辺展開図の一部（法則 (A) の例外 3）

証明. 補題 3.4 の証明をする．非カット線の集合のうち基本形に含まれる部分集合に，カット線 y_g が含まれている．ゆえに基本形が分割され定義に反することから基本形が変わる． □

最後に補題 3.5 に 3 つ目の例外を示す．図 3.6 の (434) と (435) の辺展開図（基本形が変わらない）がこれに該当する．

補題 3.5 (法則 (A) の例外 3)**.** 対称差集合を $\{y_h, y_i\}$ とし，(h) の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_h\}$，$(i)$ の辺展開図のカット線の集合を $\{x_1, \cdots, x_j, y_i\}$ とする．非カット線の集合のうち基本形以外の面に含まれる部分集合を $\{w_1, \cdots, w_j, y_i\}$ とする．このとき基本形が変わる．

証明. 補題 3.5 の証明をする．非カット線の集合のうち基本形以外の面に含まれる部分集合に，カット線 y_i が含まれている．ゆえに基本形以外の面が分割され定義に反することから基本形が変わる． □

3.3 アルゴリズムの高速化

本論文では，3.1 節の (3)，(4) に該当する各面の中心座標および，隣接しない面の組みの個数を，3.2 節の法則を適用することで削減，アルゴリズムを高速化する．各面の中心座標の計算回数の削減の方法と，隣接しない面の組みの個数の削減の方法をそれぞれ示していく．

3.3.1 各面の中心座標の計算回数の削減

まず，基本形以外の面が移動するとき，どの面のどの辺に繋がるように移動するかが問題となってくる．そこで，法則 (A) に立ち返って見てみると，「F_{s_1} の面」は「F_{s_3} の面」に繋がることが分かる．そのため，移動後の繋がる，基本形に属する面は容易に計算することが出来る．移動する前に移動する面が繋がっていた基本形が属する面を**移動する前に繋がっていた面**，移動する前に繋がっていた面と移動する面が共通に持つ辺を**移動する前の辺**，移動した後に移動する面が繋がっている基本形が属する面を**移動した後に繋がる面**，移動した後に繋がる面と移動する面が共通に持つ辺を**移動した後の辺**と呼ぶ．

一方で図 3.2 の (13)〜(17) や，図 3.4 の (480)〜(483) のように，移動する面に子孫が存在する場合を見てみると，移動する面は基本形に対して回転しているというこ

Algorithm 7 繋がる面の変更による回転

Input: 移動した後に繋がるの面 s，移動した後の辺 e，移動する前に繋がっていた面の中心から見た移動する面の中心の絶対角度 d

Output: $(d' - d)$ % (-360)

```
for i = 1 to 面の数 do
  if root(i, s) = 1 then
    k = i
  end if
end for
d' = (deg(k, s).first−180) % (−360)
if s が正 n 角形 then
  for i = 1 to n do
    if ring(s, i) = deg(k, s).second then
      l=i
      break
    end if
  end for
  for i = l to (l + n − 1) do
    if ring(s, i % n) = e then
      break
    end if
    d' = (d' − (360/n)) % (−360)
  end for
else if s が正 m 角形 then
  if の場合と同様の処理
end if
```

とが分かる．この回転というのは，次に示すの 2 種類の足し合わせによって起きている．

(a) 繋がる面の変更による回転

(b) 基本形以外の面自身の回転

まず (a) の場合の回転を **Algorithm 7** に擬似コードの形で計算の仕方を書く．S を面の集合とし，関数 root：$S \times S \to \{0, 1\}$ は親子関係の有無を表すとする．例えば 1 の面が 0 の子であるならば，root$(0, 1)$ は 1 であるが root$(1, 0)$ は 0 となる．また，M を多角形の辺の本数，D を親の面に対する子の面の絶対角度，E を親の面と子の面に共通する辺番号とするとき，関数 deg：$S \times M \to D \times E$ を親と子がどのように繋がっているかを表すとする．**Algorithm 7** では，.first で D の値を.second で E の値を指し示す．

Algorithm 8 基本形以外の面自身の回転

Input: 移動する面 s，移動する前の辺 eb，移動した後の辺 ea
Output: d

```
1: d = 0
2: if s が正 n 角形 then
3:    for i = 1 to n do
4:       if ring(s, i) = eb then
5:          k=i
6:          break
7:       end if
8:    end for
9:    for i = k to (k + n − 1) do
10:      if ring(s, i % n) = ea then
11:         break
12:      end if
13:      d = (d − (360/n))
14:   end for
15: else if s が正 m 角形 then
16:   if の場合と同様の処理
17: end if
```

Algorithm 7 は，移動した後に繋がる面の親を探すことで，移動した後に繋がる面に対する移動する面の絶対座標を求め，d との差を求める（23 行目）アルゴリズムである．1 行目から 5 行目は，移動した後に繋がるの面の親を探す操作である．6 行目は移動した後に繋がるの面から移動した後に繋がるの面の親への絶対角度を求める計算である．ここで 180° 回転させているが，これは移動した後に繋がるの面が，子を探す操作に切り替わるからである．8 行目から 13 行目は移動した後に繋がるの面の親と移動した後に繋がるの面の間に共通する辺番号 x が，関数 ring のどこに含まれているかを探しており，14 行目から 19 行目は移動した後の辺 e が x から何度回転した場所にあるかを計算している．

次に (b) の場合の回転を **Algorithm 8** に擬似コードの形で計算の仕方を書く．この **Algorithm 8** は非常にシンプルで，移動する面が移動の前後で何度回転したかを計算するだけである．

この (a) と (b) の回転角度を使うことで基本形以外の面の座標が，基本形の座標を再度計算することなく，求めることが出来るのである．基本形以外の面の座標の求め方を **Algorithm 9** に擬似コードの形で示す．なお面番号 n における x 座標の値が計算できたら，配列 x[n] で値を保持し，y 座標の値の計算ができたら，配列 y[n] で値を保持する．そして，基本形が変わる場合は配列 x, y 全ての値を更新し，基本形が変わらない場合は基本形以外の面の配列の値を更新する．

Algorithm 9 の 1 行目は移動した面の種類に応じて場合分けを行い，3,4 行目で

Algorithm 9 Recalc 関数

Input: 移動した後に繋がるの面 sa，移動した面 s，(a)+(b) の角度 d，移動した後に繋がるの面の内接円の半径 l

1: $nl =$ 移動した面の内接円の半径
2: $d = (\ \deg(sa, s).\text{second} + d + 180) \ \% \ 360$
3: $\text{x}[s] = \text{x}[sa] + (l + nl) * \sin(d * (\pi/180))$
4: $\text{y}[s] = \text{y}[sa] + (l + nl) * \cos(d * (\pi/180))$
5: **for** $i = 1$ to 面の数 **do**
6:　**if** $\text{root}(s, i) = 1$ **then**
7:　　$\text{Recalc}(s, i, d, nl)$
8:　**end if**
9: **end for**

使うための移動した面の内接円の半径を決める．2 行目は，deg 関数に記憶されている，つまり基本形が変わった時の親から子への角度を呼び出し，先ほど計算した d を加算したのち，子は親となる．3,4 行目では，(x, y) 座標の更新を行っている．5 から 8 行目では，先ほど親となった面に子が存在するかを探索し，子が存在するならば再帰計算を行い，存在しないのであれば計算を終了する．

3.3.2　隣接しない面の組みの削減

2.3 節では，隣接しない面の組みを求めるとき，全ての面の集合と全ての面の集合の組み合わせから隣接する面の組み合わせを取り除くことで，判定すべき組み合わせを見つけていた．しかし，基本形に含まれる面どうしの組みは座標が変わることが無いため 1 度だけ判定すればよく，基本形以外の面の集合と全ての面の集合との組み合わせから隣接する面の組み合わせを取り除くだけでよい．ゆえに，隣接しない面の組み合わせの個数が削減でき，高速な判定が可能となるのである．

第4章　計算機実験

4.1　実験設定

本研究では，自己重複の判定の計測時間が，既存手法にに比べ提案手法がどれぐらい早くなったかを調べるために，既存手法および提案手法の実装を行った．実装の際，全域木の数え上げやカット線の列挙には，容易に ZDD を構築することができる C++用のライブラリ TdZdd ライブラリ [13] を用いた（TdZdd ライブラリの詳しい使い方は [14] を参照）．また，展開図の座標および各面の円周を C++で計算し，Python の math モジュールを用いて重複の確認を行った．なお，丸め誤差を小さくするために，C++では式の形で一度標準出力し，Python で標準入力で読み込み計算をした．2.3 で説明した外接円の半径に関しては，$\Delta = 0.001$ 大きく設定し，計算結果は全て小数点以下 4 桁目で丸めた．実験に使った計算機は Intel(R) Xeon(R) CPU E5-2643 v4 3.40GHz, CentOS 7.9, 512 GB メモリである．

4.2　実験結果と評価

各種凸多面体に対して，既存の自己重複の判定および提案アルゴリズムによる判定を行い各時間を 50 回ずつ計測し平均を取った．その結果を表 4.1 に示す．既存手法 (A) および提案手法 (B) の計測時間の単位は (秒)，(A)−(B) は既存手法を提案手法に変えた時の時間の差（単位は (秒)），100×((A)−(B))/(A) は，既存手法の計測時間を 100 %としたとき，何%の時間を削減することができたかを表す．この結果から本論文で取り扱った全ての凸多面体について計測時間が短縮されたことが分かり，提案アルゴリズムは効果的に動作していると言える．

計測時間が短くなるための鍵となるのが，3.3 節で記した各面の中心座標の計算回数の削減および隣接しない面の組みの削減である．それぞれの結果を表 4.2 および表 4.3

凸多面体	展開図の数	既存手法 (A)	提案手法 (B)	(A)−(B)	100×((A)−(B))/(A)
正四面体	16	0.049	0.047	0.002	4.1
正六面体	384	0.377	0.370	0.007	1.9
正八面体	384	0.559	0.548	0.011	2.0
切頂四面体	6,000	7.241	6.762	0.479	6.6
立方八面体	331,776	1170.683	1078.689	91.994	7.9

表 4.1: 各凸多面体の自己重複の判定の計測時間

凸多面体	既存手法 (A)	提案手法 (B)	100×((A)−(B))/(A)
正四面体	16	8	50
正六面体	384	157	59.1
正八面体	384	199	48.2
切頂四面体	6,000	2,335	61.1
立方八面体	331,776	187,381	43.5

表 4.2: 各凸多面体に対する辺展開図の全ての面の中心座標を計算する回数

凸多面体	既存手法 (A)	提案手法 (B)	100×((A)−(B))/(A)
正四面体	12	12	0
正六面体	2,640	2,012	23.8
正八面体	4,248	3,031	28.6
切頂四面体	102,780	74,695	27.3
立方八面体	20,367,936	14,025,808	31.1

表 4.3: 各凸多面体に対する隣接しない面の組みの数

に示す．表 4.2 の既存手法 (A) および提案手法 (B) の単位は (回)，100×((A)−(B))/(A) は，既存手法における全ての面の中心座標を計算する回数を 100 %としたとき，何% の辺展開図が全ての面の中心座標を計算する必要が無くなったかを表す．表 4.3 の既存手法 (A) および提案手法 (B) の単位は (組)，100×((A)−(B))/(A) は，既存手法における隣接しない面の組みの数を 100 % としたとき，何%の隣接しない面の組みを削減できたかを表す．また，切頂四面体を例に，アルゴリズムの各ステップにおいて何%削減することができたかを図 4.1 および図 4.2 に示す．なお，ステップ (5) における判定が計測時間の殆どを占めているため，C++で実装したステップと Python で実装したステップに分けて表している．また，ステップ (1) は極めて短時間なためステップ (2) とまとめて表記している．

これらの結果から，展開図の個数が多くなればなるほど，面の個数つまり面の組み合わせが多くなればなるほど，既存手法による計算時間に対する短縮された時間に対する割合は大きくなっていき，より効率的に自己重複の確認が出来るのでは無いかと考えられる．なお正四面体に関しては，削減した面の組みの個数 0 であるのにも拘らず，既存手法による計算時間に対する短縮された時間に対する割合が 4.1% と正六面体や正八面体の場合より高くなっている．これは計測時間が約 0.05 秒と短時間であるため，多面体の辺どうしの繋がりを表すファイルの読み込み時間のブレの影響を大きく受けてためではないかと考える．

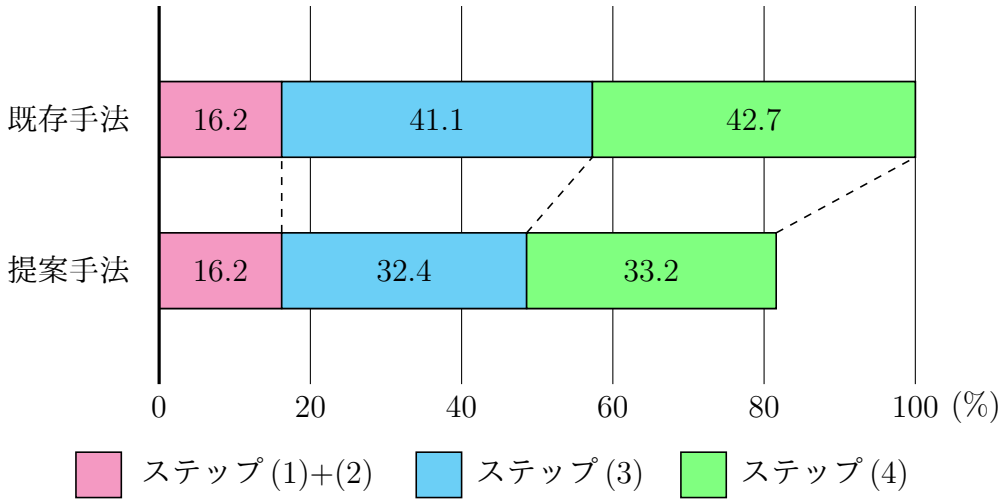


図 4.1: ステップ (1)〜(4) における減少の割合（C++で実装した箇所）

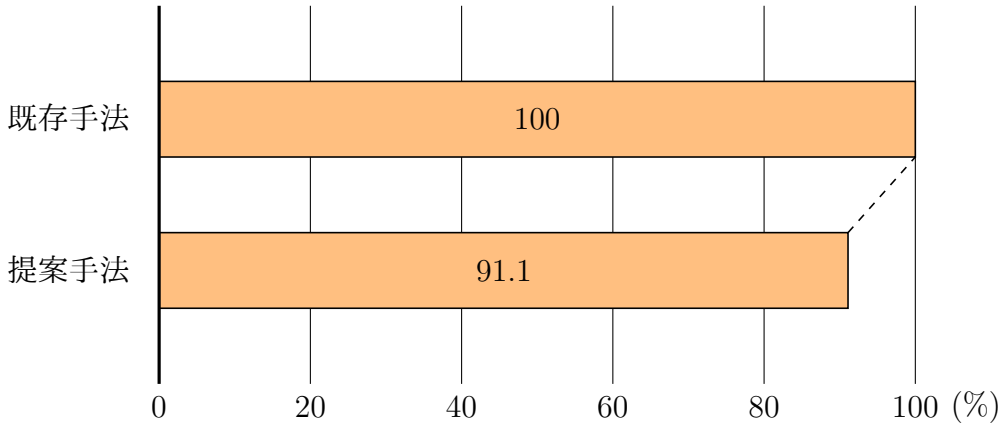


図 4.2: ステップ (5) における減少の割合（Python で実装した箇所）

第5章　おわりに

5.1　結論

本論文では各種凸多面体の辺展開図において，自己重複の確認の高速化に成功した．この高速化にともない，表 1.1 に「未解決」として記す重なりのある辺展開図の有無の判定や，重なりを持たない辺展開図の個数の計算に向けての兆しが見えた．

5.2　今後の課題

本論文では，凸多面体のうち辺展開図の個数が少ない多面体 5 種類に限った高速化を行った．今後は，本論文で提案したアルゴリズムを他の凸多面体に対しても適用し，どれぐらい高速に判定することが出来るようになるかを計測したいと考える．

しかし，現状では隣接しない面を愚直な方法で求めていたり，座標の再計算を遠回しな方法で行っていたりと課題が残る．また辺展開図の中には「同型な辺展開図（回転したり左右を反転したりすることで同じ形となる辺展開図）」が含まれているため，排除しなければならない．こうした高速化を妨げる要因を無くし，提案アルゴリズムを更に高速なものへとすることで,「未解決」とされている凸多面体へのアプローチが出来るのではないかと考える．

参考文献

[1] Albrecht Dürer. Underweysung der messung, mit dem zirckel und richtscheyt in linien ebenen unnd gantzen corporen. 1525.

[2] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, Jul. 2007. 上原 隆平 (訳). 幾何的な折りアルゴリズム—リンケージ、折り紙、多面体, 近代科学社, Nov. 2009.

[3] Takashi Horiyama and Wataru Shoji. Edge unfoldings of platonic solids never overlap. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12, 2011*, 2011.

[4] 廣瀬 健汰. 半正多面体の展開図の重なりについて. 埼玉大学工学部情報システム工学科, 2015. 卒業論文，参考：指導教員 堀山 貴史.

[5] Takashi Horiyama and Wataru Shoji. The number of different unfoldings of polyhedra. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, Vol. 8283 of *Lecture Notes in Computer Science*, pp. 623–633. Springer, 2013.

[6] Hallard T. Croft, Kenneth J. Falconer, and Richard K. Guy. *Unsolved Problems in Geometry*. Springer-Verlag, reissue edition, 1991.

[7] 上原 隆平. 計算折り紙入門. 近代科学社, 2018.

[8] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, Vol. 35, No. 8, pp. 677–691, 1986.

[9] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In Alfred E. Dunlop, editor, *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pp. 272–277. ACM Press, 1993.

[10] ERATO 湊離散構造処理プロジェクト. 超高速グラフ列挙アルゴリズム. 森北出版株式会社, 湊 真一, 2015.

[11] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E100.A, No. 9, pp. 1773–1784, 2017.

[12] Github truncated tetrahedron. `https://github.com/ShiotaTakumi/For-publication/blob/master/Unfolding/Labeled/Archimedean/truncated_tetrahedron.pdf`.

[13] GitHub TdZdd. `https://github.com/kunisura/TdZdd`.

[14] 戸田 貴久, 斎藤 寿樹, 岩下 洋哲, 川原 純, 湊 真一. Zdd と列挙問題 – 最新の技法とプログラミングツール. コンピュータソフトウェア, Vol. 34, No. 9, pp. 97–120, Jan. 2017.

謝辞

本研究を遂行するにあたり，配属の相談をさせて頂いてた時期を含め 1 年以上丁寧に指導して下さった斎藤先生に深く感謝申し上げます．コロナ禍で大学へ行く回数も減った中，ここまで研究に打ち込むことが出来たのは，一重に先生のおかげです．本当にありがとうございます．

創作プロジェクト II の発表で「言っていることの意味が分からない！」とキッパリ断言して頂いた宮野先生には感謝の念が足りません．すごく，自己満足な発表になっているということに気付き，これをバネに若手 OR 発表会で成果を残すことが出来ました．また，合同ゼミの発表で「ここは，こういう意味で合ってる？」と自分の説明での言葉の不足を補って頂いた藤本先生に感謝申し上げます．

拙い英語での質問にも答えて頂いたり，折り紙工学に関する共同研究会を紹介してくれた Duc 氏．ありがとうございました．共同研究会に参加出来たことは良い刺激となり，折り紙の研究は奥が深く，美しく，そして本当に楽しいものであると改めて感じることが出来ました．

本研究をしたいと決めるきっかけを頂き，その後も斎藤先生を通じて，研究に関するアドバイスを頂いた北陸先端技術大学院大学の上原先生に感謝申し上げます．朝イチで小松まで飛んでいなかったら，今の私は折り紙やパズルの楽しさというものに気付けないまま生涯を過ごすことになってました．

隣の席で研究室の以呂波を教えてくれた咸先輩，TdZdd ライブラリの使い方全般を何日も夜遅くまで zoom をつないで教えて下さった高瀬先輩，研究の要となる全域木の実装の方法を教えて下さった新谷先輩．本当にありがとうございました．

武田くん，土井くん，石井くん，永留くん．一緒の研究室で研究を進めていくことが出来て良かったと本当に感謝をしています．斎藤先生には研究は勝負では無いとは言われましたが，本当に良い刺激を受けました．特に，若手 OR 発表会の練習に付き合ってくれてありがとうございました．みんなのコメントのおかげで取れた最優秀賞だと思ってます．

最後になりますが，紙に印刷した展開図から立体モデルを組み立てる作業や，ゲオマグで作成した立体モデルに貼っていたラベル番号を書いたシールを剥がす作業を手伝ってくれた家族に感謝の気持ちを送ります．そして，ずっと家に居たせいで研究で行き詰まった時のストレスを幾度となくぶつけてしまってごめんなさい．

研究業績

- **受賞**

  1. 日本オペレーションズ・リサーチ学会九州支部
     若手 OR 交流会 2020 最優秀賞発表賞 (学部生の部),
     “フロンティア法によるアルキメデスの立体の辺展開図の列挙”,
     令和 2 年 11 月 28 日.