中央大学大学院理工学研究科情報工学専攻

修士論文

# 選挙区割問題に対するヒューリスティクスを用いた ZDD 構築の効率化

Efficient ZDD Construction Using Heuristics for the Electoral Districting Problem

千原 良太
Ryota CHIHARA
学籍番号　21N8100011I

指導教員　今堀 慎治 教授

2023 年 3 月

# 概　要

衆議院議員選挙小選挙区制における選挙区割問題とは，各都道府県ごとに議席数 (区割数) が定められており，市区町村からなる小地域を組み合わせて区割を構成し，その中から最も良い区割を見つける離散最適化問題の一種である．実際の選挙区割では，人口の偏りによる「一票の格差」が問題提起されており，人口の格差を最小にした区割の導出が求められている．

この問題の解法として，ゼロサプレス型二分決定グラフ (ZDD) を用いた区割列挙が知られている．区割数や各区割の人口の上限・下限などを制約として与え，その制約から枝刈りを行うことで，解候補を列挙することができる．ただし，区割人口の上下限制約は，平均人口から一律に定められた格差許容定数を用いて計算し，メモリ不足等で解が導出できない場合のみ値を変更する手法が多く取られていた．

本論文では，ヒューリスティクスを用いて区割人口の上下限制約を定め，それを基に ZDD を構築することで，従来よりも効率的に解候補を得る手法を提案する．また，計算機実験を行い，従来手法よりも ZDD 構築における計算時間とメモリ使用量が削減できることを確認する．

**キーワード:** 離散最適化，選挙区割問題，ZDD，ヒューリスティクス．

# 目 次

# 第1章　はじめに

日本の衆議院議員選挙における小選挙区制の区割は，総定数から各都道府県に何議席を割り当てるかを決める定数配分問題と，都道府県内で割り当てられた議席数分の選挙区を市区町村を組み合わせて構築する区割画定問題を解くことにより，定めることができる．

定数配分問題は，法学，公共政策学，数理情報学などの様々な観点から取り組まれており，過去200年以上にわたり多くの手法が提案されている．2022年12月28日には，公職選挙法の一部を改正する法律（区割り改定法）が施行され，衆議院小選挙区選出議員の選挙区について「アダムズ方式」を用いた議席の配分が行われた [1]．アダムズ方式はアメリカ6代目大統領ジョン・アダムズが考案したとされており，簡単に説明すると「各都道府県の人口をある自然数で割った商の小数点以下を切り上げた数を，その都道府県の議席数とする」手法である．この手法は，一票の格差の是正には効果的とされているが,「アラバマ・パラドックス」と呼ばれる改定時に議席総数が増加した際に，ある地区では配分される議席数が改定前より減る現象が起こる場合がある．また，過去に提案された手法を比較したときに，一票の格差がほぼ等しい場合でも，各都道府県の人口が多い方が有利な手法，少ない方が有利な手法といった差が現れ，どの手法も一長一短であることから，選挙制度の意義等も踏まえつつ議論する必要がある．本研究では，定数配分問題については主に扱わず，2021年に行われた第49回衆議院議員選挙の定数配分をそのまま利用する．

区割画定問題は，都道府県内の選挙区の組合せが市区町村数の指数通り存在し，NP困難であるとして，20世紀末までは最適性の保障のない解の導出の研究が主であった．しかし，2003年に根本・堀田が数理モデルによる定式化を提案 [2] して以降，数理的な観点から多くの研究が取り組まれており，厳密解を導出するための手法がいくつか提案されている．その中の手法の一つとして，ゼロサプレス型二分決定木（ZDD）を用いた区割列挙があり，フロンティア法によってトップダウンにZDDを構築することで，高速に選挙区割を求めることが可能となっている．ただし，いくつかの都道府県においては計算機のメモリ不足により解を導出することが困難である．

本研究では，区割画定問題（以下「選挙区割問題」と称する）におけるZDDを用いた区割列挙について扱い，ヒューリスティクスを用いて効率的に区割列挙を行う手法を提案する．本稿の第2章では選挙区割問題について定義し，数理モデルによる定式化を説明す

る．第 3 章では ZDD とフロンティア法，それを用いた区割列挙アルゴリズムについて詳しく述べる．第 4 章では，第 3 章で説明したアルゴリズムとヒューリスティクスを組み合わせることで ZDD 構築を効率化する手法を提案する．そして，第 5 章で計算機実験の結果とその考察を示し，第 6 章で結論と今後の課題について述べる．

# 第2章　選挙区割問題

本章では，選挙区割問題について，国の公表資料から区割作成方針を示し，問題の定義，数理モデルによる定式化について説明する．

## 2.1　区割作成方針

衆議院議員選挙の選挙区割の改定案は，衆議院議員選挙区画定審議会によって作成される．当審議会では，令和4年2月21日に『区割り改定案の作成方針』を公表しており，作成方針を簡潔に述べると以下の6点となる．

1. 一票の格差は2倍未満とする．
2. 議員1人当たり人口が最も少ない県においては，各選挙区の人口をできるだけ均等にする．
3. 改定案の作成にあたり，選挙区の区域の異動は，区割り基準に適合させるために必要な範囲とする．
4. 選挙区は飛び地にしない．
5. 選挙区を構成する市区町村は原則分割しない．
6. 地勢，交通，人口動向などの自然的，社会的条件を総合的に考慮する．

本研究では，項目1,2,4,5を主に考慮する．ただし，項目2は，人口最小の県だけでなく，全ての都道府県において各選挙区の人口をできるだけ均等にすることを目指す．項目3は，改定前との区割の比較が研究の主旨ではないため考慮しない．項目6は，モデル化するには複雑であるため，今回は飛び地にしないことで，自然的，社会的条件を満たしているとみなす．

## 2.2 問題定義

区割作成方針をもとに問題を定義する．まず，都道府県ごとに市区町村とその隣接関係，各市区町村の人口を重みつきグラフ $G = (V, E, w)$ で表現する．入力は，市区町村数を $l$ として，市区町村集合 $V = \{v_1, ..., v_l\}$, 市区町村の隣接関係 $E = \{\{v_i, v_j\}|$ 市区町村 $v_i$ と $v_j$ は隣接 $\}$, 市区町村 $v_i$ の人口 $w_i$, 選挙区数 $d(< l)$ が与えられる．そして出力は，$d$ 個の選挙区の集合 $S = (S_1, ..., S_d)$ であり，$S_k$ は $k$ 番目の選挙区に属する市区町村の集合を表す．ただし，選挙区は以下の制約を満たす必要がある．

**制約 1**：選挙区に属する市区町村から誘導される部分グラフは連結である．

**制約 2**：全ての市区町村は唯一つの選挙区に属す．

**制約 3**：選挙区は空集合ではない．

また，選挙区ごとの人口の和 $P = \{P_1, ..., P_d\}$ $(P_k = \sum_{v_i \in S_k} w_i \ (k = 1, ..., d))$ を計算し，選挙区人口の最小値 $\min(P)$ と最大値 $\max(P)$ を調べる．制約を満たす選挙区割の中で，一票の格差が最小のもの，すなわち $\frac{\max(P)}{\min(P)}$ の値が最小であるものを最適な選挙区割と定める．

## 2.3 モデル表現

選挙区割を表す数理モデルの代表例として，集合分割型モデル [2] を説明する．まず，制約 1 から連結である市区町村の集合をブロックと名付ける．空集合を除く全てのブロック集合を $\mathscr{B}$ で表し，ブロック集合 $\mathscr{B}$ から選んだ $d$ 個のブロックが制約 2 を満たすと，実行可能な区割となる．その中で一票の格差が最小な区割を見つける．

**入力データ**:市区町村集合 $V$，選挙区数 $d$，ブロック集合 $\mathscr{B}$ を表す行列 $[b_{ij}|i = 1, ..., n,\ j = 1, ..., |\mathscr{B}|]$：市区町村 $v_i$ がブロック $j$ の構成要素のとき $b_{ij} = 1$；そうでないとき $b_{ij} = 0$，ブロック $j$ の人口 $q_j = \sum_{i \in n} b_{ij} w_i \ (j = 1, ..., |\mathscr{B}|)$.

**変数**：一つの選挙区の人口上限を示す変数 $u$，下限を示す変数 $l$，バイナリ変数 $x_j$ $(j = 1, ..., |\mathscr{B}|)$：区割にブロック $j$ を使用するとき $x_j = 1$；しないとき $x_j = 0$.

**定式化：**

$$\text{minimize} \qquad u/l \tag{2.1}$$

$$\text{subject to} \qquad q_j x_j \leq u \quad (j = 1, ..., |\mathscr{B}|) \tag{2.2}$$

$$\alpha(1 - x_j) + q_j x_j \geq l \quad (j = 1, ..., |\mathscr{B}|) \tag{2.3}$$

$$\sum_{j=1,...,|\mathscr{B}|} b_{ij} x_j = 1 \quad (i = 1, ..., n) \tag{2.4}$$

$$\sum_{j=1,...,|\mathscr{B}|} x_j = d \tag{2.5}$$

$$x_j \in \{0, 1\} \quad (j = 1, ..., |\mathscr{B}|) \tag{2.6}$$

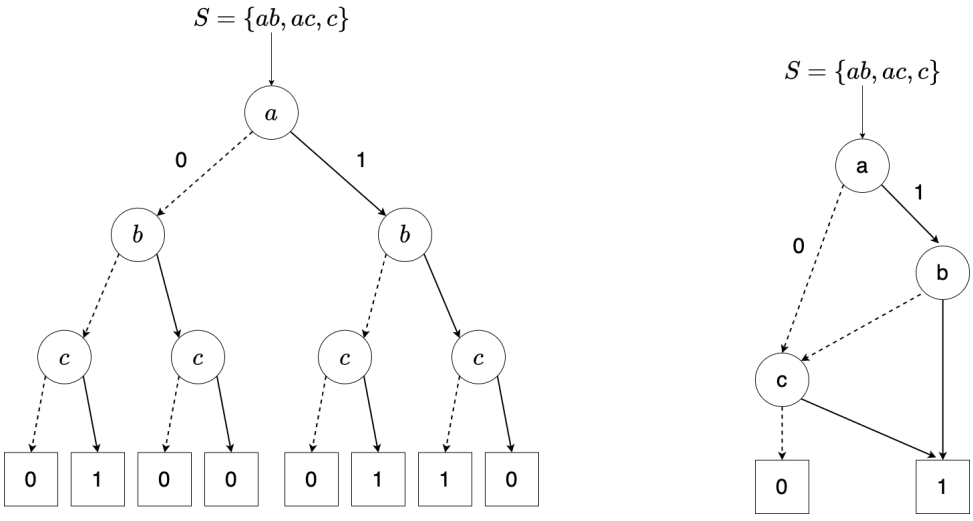
ここで，$\alpha$ は十分大きな定数とする．式 (2.1) は一票の格差を最小化することを目的関数として示している．式 (2.2) と (2.3) は変数 $u$ と $l$ を正しく表すために必要な制約である．式 (2.4),(2.5),(2.6) は制約 2 を表現している．また，制約 1 と 3 についてはブロック集合 $\mathscr{B}$ から選挙区を構成しているため，条件を満たしている．よって，上述の形で選挙区割問題を定式化することができる，

# 第3章　ZDDを用いた区割列挙手法

本章では，ZDD と呼ばれるデータ構造について解説し，川原らが提案した頂点重み格差を考慮した部分グラフ列挙アルゴリズム [3] を選挙区割問題に適用する手法について述べる．

## 3.1　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，組合せ集合を表す非巡回有向グラフで，1993年に湊真一によって考案された [4]．組合せ集合とは「$n$ 個のアイテムから任意個を選ぶ組合せ」を要素とする集合である．$n$ 個のアイテムから任意個を選ぶ組合せは $2^n$ 通りあるので，その組合せ集合は，$2^{2^n}$ 通り存在する．例えば，$a, b, c$ の 3 つの要素から組合せ集合を作る場合，$\{ab, ac, c\}, \{a\}, \{\lambda, abc\}, \phi$ などが挙げられる（$\lambda$ は空の組合せ要素，$\phi$ は空の集合を表す）．




**(a)** 場合分け二分木　　**(b)** ZDD

**図 3.1:** 組合せ集合 $S = \{ab, ac, c\}$ のグラフ表現

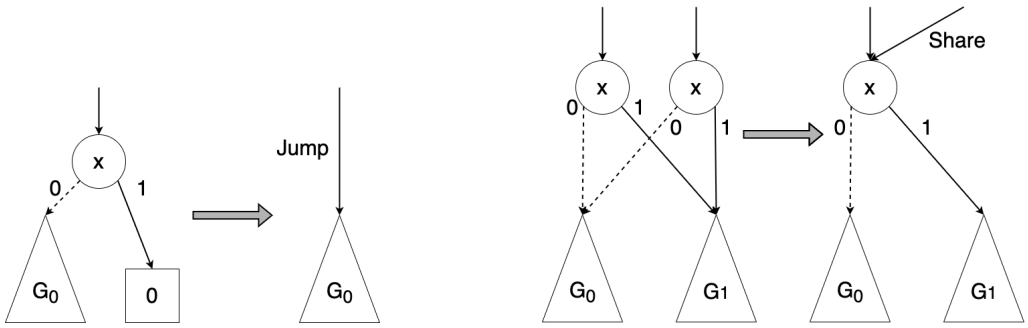
このような指数的な数の集合は，ZDD を用いることで効率的に表現することができる．図 3.1 は，組合せ集合 $S = \{ab, ac, c\}$ を場合分け二分木と ZDD の両方で表した例である．

この二つのグラフは，各節点にアイテム名を表すラベルが割り当てられていて，0 と 1 のラベルが付与された 2 種類の枝が分岐している．そして葉には 0 または 1 の値が記入されている．以降は 0 のラベルをもつ枝を 0-枝 (破線)，1 のラベルをもつ枝を 1-枝 (実線) とし，また，0 の値が記入された葉を 0-終端 (0-terminal)，1 の値が記入された葉を 1-終端 (1-terminal) と呼ぶ．これらのグラフでは，1-枝と 0-枝はその節点の要素を選ぶかどうかの場合分けを表し，葉の値はその葉に対応する組合せが集合に属するかを示している．

場合分け二分木と ZDD を比較すると，ZDD は場合分け二分木で集合の表現に不要な頂点と枝を削除，圧縮していることがわかる．場合分け二分木から ZDD を構築するには，次の 2 つの規則を可能な限り適用する．

**冗長頂点の削除** 1-枝が 0-終端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 3.2a).

**等価節点の共有** 等価な節点（ラベルが同じで，0-枝同士，1-枝同士の行き先が同じ）を共有する (図 3.2b).




**(a)** 冗長節点の削除　　**(b)** 等価節点の共有

**図 3.2:** ZDD の圧縮規則

また，ZDD は組合せ集合をコンパクトに表現できるだけでなく，ZDD 同士で集合演算が定義されており，共通集合や差集合などを表す ZDD を高速に得ることができる．これを利用することで，愚直に場合分け二分木を作るよりも高速に ZDD を構築するボトムアップな構築手法 [4] が知られている．ただし，アイテム数に対して指数的な個数の組合せを生成するような集合演算を行う場合は注意が必要であり，集合演算の順序によっては圧縮がうまく機能せず，ZDD のサイズが指数的に増大する恐れがある．このような場合は，計算を行う前に圧縮度を推定することは一般的に困難である．

近年の研究では，ボトムアップな構築手法ではなく，根から順に ZDD を作成するトップダウンな構築手法 [5][6] がよく用いられている．アイテム $\{a_1, \ldots, a_m\}$ から特定の組合

**Algorithm 1** トップダウンな ZDD 構築アルゴリズム

```
1: 根ノード n_root を作成
2: for i = 1, ..., m do // m: アイテム数
3:     for 既に作成済みの i 段目の各ノード n について do
4:         for all x ∈ {0, 1} do // 0-枝，1-枝の処理
5:             終端条件の判定
6:             新しいノード n' を作成 (i + 1 段目とする)
7:             n' の情報を更新
8:             if n' と等価なノード n'' が既に存在 then
9:                 n' ← n''
10:            end if
11:            n の x-枝の先を n' とする．
12:        end for
13:    end for
14: end for
```

せ集合を表す ZDD を構築するアルゴリズムについて，疑似コードの形で **Algorithm 1** に記した．

**Algorithm 1** の 2 行目が ZDD の各段についての処理，3 行目がその段の各ノードについての処理である．4 行目が $x$-枝 $(x = 0, 1)$ の先のノードを作成する．$x = 1$ はアイテム $a_i$ を採用する場合，$x = 0$ はアイテム $a_i$ を採用しない場合に対応する．5 行目では，終端条件の判定を行う．その時点で組合せ集合の要素に含まれないと判定できる場合には，$n$ の $x$-枝の先を 0-終端に接続する．終端に接続する場合は，6-11 行目は実行しない．7 行目では，ノードに記憶させる情報を更新する．記憶させる情報は問題によって様々であるため，内容は後述する．8 行目でノードが共有可能であるか判定を行う．

トップダウンな構築手法では，フロンティアというラベル $a_i$ がついた枝を処理した状態における頂点集合 $F_i = (\bigcup_{j=1,\ldots,i} a_j) \cap (\bigcup_{j=i+1,\ldots,m} a_j)$ を用いることから，一般的に「フロンティア法」と呼ばれる．フロンティアは，処理が途中のアイテムのみを保持するため，**Algorithm 1** の 5 行目の処理では，フロンティアに含まれるノードの情報のみ参照すればよい．アイテムの総数に対して，フロンティアのサイズが小さければ効率よく計算を行うことができる．

なお，5-7 行目の内容は，同じフロンティア法でも解く問題の種類によって細かな違いがある．選挙区割問題の実装については後節で説明する．

## 3.2 ZDD での選挙区割列挙の表現

本節では，選挙区割の列挙を表現する組合せ集合と ZDD について述べる．選挙区割の集合は，選挙区となる部分グラフを構築に利用できる全ての辺の組合せの集合と言い換えることができる．例として，図 3.3 から 2 つの部分グラフを列挙することを考える．

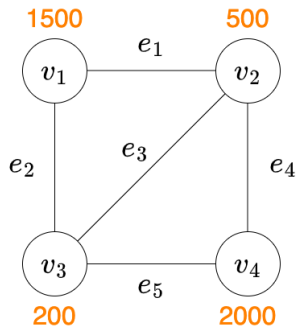



**図 3.3:** 4 頂点の重み付きグラフ

図 3.3 の橙色で書かれた数字は，頂点の重み，すなわち市区町村の人口である．グラフ分割する際には，各グラフの重みをなるべく均等にしたいため，今回は解となる頂点 (市区町村) の集合を $\{\{v_1, v_2\}, \{v_3, v_4\}\}, \{\{v_1, v_2, v_3\}, \{v_4\}\}$ の 2 つとすると，部分グラフの重みは，それぞれ $\{2000, 2200\}, \{2200, 2000\}$ となる．

ここで部分グラフを構成する辺に注目してみる．頂点集合から部分グラフの構成に利用できる全ての辺の集合は，$\{e_1, e_5\}, \{e_1, e_2, e_3\}$ である．この辺集合から ZDD を構築することで，選挙区割の列挙を行うことができる (図 3.4)．

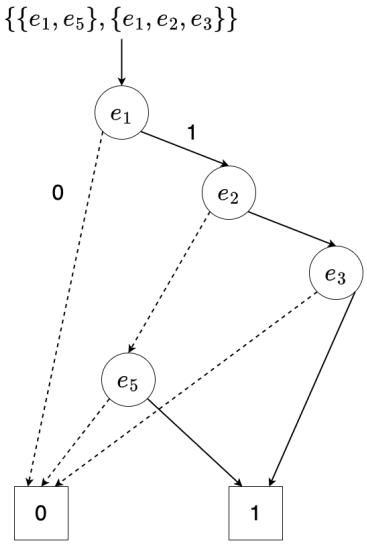



**図 3.4:** 選挙区割の集合を表す ZDD

図 3.4 では，節点のラベルに辺の名前が記載されている．1-枝は節点のラベルに記されたグラフの辺を採用，0-枝は採用しないことを表している．そして，1-終端に向かうパスを辿ることで，求めたい部分グラフ集合を容易に見つけることができる．

## 3.3 区割列挙アルゴリズム

本節では，川原らが考案した重み付きグラフ分割列挙アルゴリズム [3] を用いて，選挙区割を列挙する ZDD の構築アルゴリズムを説明する．

アルゴリズムを説明する前準備として，用語・変数について定義する．まず，元のグラフを $G = (V, E, w)$，辺 $E$ の総数を $m$，列挙したい部分グラフ集合を $\mathcal{G}$，分割数 (区割数) を $d$ とする．なお，辺集合 $E$ の順序付けは，ヒューリスティックアルゴリズムを用いて，フロンティアの大きさが出来るだけ小さくなるように行う．$N$ は ZDD のノードの集合を表し，$N_i$ はグラフ $G$ の辺 $e_i (\in E)$ をラベルにもつノード集合を指す．$F_i$ は，ラベル $e_i = \{u, v\}$ をもつ ZDD ノードを処理している時点での，グラフの頂点集合からなるフロンティアであり，

$$F_i = (\bigcup_{j=1,\ldots,i} e_j) \cap (\bigcup_{j=i+1,\ldots,m} e_j)$$

から求められる．

ZDD のノード $n$ には，終端を除いて以下の情報を含んでいる．

$n$.comp
: フロンティア上での，部分グラフの頂点の集合を示す．
例えば，$n$.comp $= \{\{v_1, v_2, v_3\}, \ldots\}$ とあれば，頂点 $v_1, v_2, v_3$ が同一部分グラフの頂点であり，連結していることを表す．

$n$.fps
: 接続を禁止する頂点のペアセット (forbidden pair set)．
例えば，部分グラフの頂点集合をそれぞれ $C_1, C_2$ とすると，$n$.fps $= \{\{C_1, C_2\}, \ldots\}$ は，$C_1$ に含まれる頂点と $C_2$ に含まれる頂点を接続させてはいけないことを表す．

$n$.cc
: 確定した部分グラフの個数の値が入る．
フロンティアからある部分グラフ内の頂点が全て取り除かれたとき，その部分グラフは他のグラフと連結することがないため，$n$.cc をインクリメントする．

$n$.weight

部分グラフごとに頂点重みの和を保存する．

$n$.min

フロンティアから離脱した部分グラフの頂点重みの最小値．初期値は非常に大きい数とする．

$n$.max

フロンティアから離脱した部分グラフの頂点重みの最大値．初期値は 0.

なお，根ノード $n_{root}$ のもつ情報は，$n$.comp，$n$.fps, $n$.weight は空集合 $\emptyset$，$n$.cc $= 1$ と定義する．次に，選挙区割の ZDD を構築するフロンティア法を **Algorithm 2** に示す.

**Algorithm 2** ConstructZDD

```
1: N_1 ← {n_root}
2: N_i ← ∅ for i = 2, ..., m + 1
3: for i = 1, ..., m do
4:     for all n ∈ N_i do
5:         for all x ∈ {0, 1} do
6:             n' ← MakeNewNode(n, i, x)
7:             if n' ≠ 0, 1 then
8:                 if n' と等価なノード n'' ∈ N_{i+1} が既に存在 then
9:                     n' ← n''
10:                else
11:                    N_{i+1} ← N_{i+1} ∪ {n'}
12:                end if
13:            end if
14:            n の x-枝の先を n' とする．
15:        end for
16:    end for
17: end for
```

**Algorithm 2** の全体の流れは，**Algorithm 1** と大きく異ならないため，詳細な説明は省略する．注目すべき点は，6 行目の MakeNewNode という関数である．これは，終端条件の判定と，$i + 1$ 段目に新しいノード $n'$ を作成する処理が含まれていて，新しい節点も

しくは，0-終端，1-終端のいずれかを返す．この関数の処理を，人口制約がない場合とある場合のそれぞれで説明する．

### 3.3.1 人口制約なしの場合

まず，人口制約がない場合，ZDD のノードに含まれる情報は，$n$.comp，$n$.fps，$n$.cc の 3 種であり，$n$.weight などは含まれない．MakeNewNode の擬似コードを **Algorithm 3** に示す．

**Algorithm 3** MakeNewNode

**Require:** $n, i, x$

**Ensure:** $n'$ or **0** or **1**

1: 引数 $i$ から $e_i = \{v, w\}$ を取得する．
2: $n' \leftarrow n$
3: **for all** $u \in \{v, w\}$ **do**
4:  **if** $u \notin F_{i-1}$ **then**
5:  $n'.\mathrm{comp} \leftarrow n'.\mathrm{comp} \cup \{\{u\}\}$
6:  **end if**
7: **end for**
8: $n'$.comp の $v$ と $w$ を含む頂点集合をそれぞれ $C_v$ と $C_w$ とする．
9: **if** $x = 1$ **then**
10:  $n'.\mathrm{comp} \leftarrow (n'.\mathrm{comp} \backslash \{C_v\} \backslash \{C_w\}) \cup \{C_v \cup C_w\}$
11:  **if** $C_v \neq C_w$ and $\{C_v, C_w\} \in n'.\mathrm{fps}$ **then**
12:  **return 0**
13:  **else**
14:  $n'$.fps 内の要素 $C_v$ と $C_w$ を全て $C_v \cup C_w$ に置き換える．
15:  **end if**
16: **else**
17:  **if** $C_v = C_w$ **then**
18:  **return 0**
19:  **else**
20:  $n'.\mathrm{fps} \in n'.\mathrm{fps} \cup \{\{C_v, C_w\}\}$
21:  **end if**
22: **end if**

23: **for all** $u \in \{v, w\}$ **do**
24: **if** $u \notin F_i$ **then**
25: **if** $\{u\} \in n'.\mathrm{comp}$ **then**
26: $n'.\mathrm{cc} \leftarrow n'.\mathrm{cc} + 1$
27: **if** $n'.\mathrm{cc} > d$ **then**
28: **return 0**
29: **end if**
30: **end if**
31: $n'.\mathrm{comp}$ から $u$ を取り除く.
32: $n'.\mathrm{fps}$ から $\{\{u\}, X\}$ を取り除く $(\forall X \in n'.\mathrm{comp})$.
33: **end if**
34: **end for**
35: **if** $i = m$ **then**
36: **if** $n'.\mathrm{cc} = d$ **then**
37: **return 1**
38: **else**
39: **return 0**
40: **end if**
41: **end if**
42: **return** $n'$

**Algorithm 3** の関数 MakeNewNode の引数は,

- $n$：作成するノードの親ノード
- $i$：作成するノードの深さ（根を 0, 辺 $e_i$ の処理を行う）
- $x$：親ノードから接続されている枝（0-枝または 1-枝）

となり，戻り値は，$n'$（新しい節点），**0**（0-終端），**1**（1-終端）のいずれか一つである．3-7 行目では，グラフ頂点 $u$ が初めてフロンティアに入るとき，$n'.\mathrm{comp}$ に集合 $\{u\}$ を追加している．9-15 行目は，ノードが 1-枝に接続されている場合の処理である．このとき，$v$ と $w$ は接続されるため，$n'.\mathrm{comp}$ の情報を更新し，接続禁止ペアセット $n'.\mathrm{fps}$ に $\{C_v, C_w\}$ が含まれている場合は，**0** を返し，処理を終える．含まれていない場合には，$n'.\mathrm{fps}$ 内の要素を置き換え，23 行目以降の処理に移る．16-22 行目は，ノードが 0-枝に接続されている場合の処理である．17 行目の条件文が真のとき，$C_v$ と $C_w$ は，同一部分グラフの頂点

集合となるので，0-枝では条件を満たせないため，**0** を返す．そうでない場合には，今後一切 $C_v$ と $C_w$ を接続しないように，新たに $n'$.fps に $\{C_v, C_w\}$ ペアを追加する．23-34 行目は，$u$ がフロンティアから抜けるときの処理である．$u$ がフロンティア内において，他の連結成分がない場合，$u$ を含む部分グラフは完全に分離しているため，$n'$.cc を 1 増やす．また，このときグラフの分割数が $d$ を超えた場合，**0** を返して処理を終える．35-41 行目は，グラフ辺の探索を最後まで終えたときの処理であり，$n'$.cc と $d$ が一致しているかの判定をする．一致している場合は **1**，一致していない場合は **0** を返す．まだ，探索が続く場合には，作成した $n'$ を返す．

### 3.3.2 人口制約ありの場合

人口制約とは，部分グラフ内の頂点重みの和に上限と下限を設けるものであり，この制約を満たす区割を列挙する手法について考える．まず，許容格差定数 $r$ を定め，部分グラフの重み上限を $U$, 下限を $L$ とおく．各部分グラフの重みの和を昇順に $a_1, \ldots, a_d$ とすると，グラフ $G$ の重みの和を $W$ として，

$$W \geq (d-1)a_1 + a_d \geq \frac{(d-1)a_d}{r} + a_d$$

から，$U$ を定義すると，

$$U \coloneqq \frac{rW}{r+(d-1)} \geq a_d$$

となる．また，

$$W \leq a_1 + (d-1)a_d \leq a_1 + r(d-1)a_1$$

から，$L$ を定義すると，

$$L \coloneqq \frac{W}{r(d-1)+1} \leq a_1$$

となる．

人口制約 $U, L$ を ZDD の構築に用いるには，ZDD のノードに $n$.weight の情報を加え，**Algorithm 3** を一部修正する必要がある．$n$.weight の要素は，$n$.comp の集合要素とインデックスが 1 対 1 で対応しており，部分集合の (フロンティアを外れた頂点も含めた) 重みが保存されている．アルゴリズムでは，comp の代入処理の後に，その頂点や集合の重みを weight の対応要素に加算すれば良い．また，1-枝で部分グラフに頂点を結合する際には，上限制約 $U$ を満たしているかの判定を行い，条件を満たさない場合は，その時点で **0** を返す．さらに，部分グラフがフロンティアから外れる際にも，下限制約 $L$ を満たしているかを判定し，条件を満たさない場合は **0** を返す．そして，部分グラフがフロンティアか

ら離脱するときに $n.\min$, $n.\max$ を更新する．このとき，$n.\max/n.\min > r$ であれば，許容格差を超えているため **0** を返す処理を追加する．以上の操作を加えることで効率的に枝刈りを行い，制約を満たす解のみを計算することができる．

人口制約を含んだ ZDD は，含まないものに比べて解の個数が指数的な数から大きく減るため，列挙後の最適解の計算や解候補の分析などがしやすいメリットがある．しかし，ZDD のノードに weight の情報が存在すると，ZDD の等価節点の共有が行いづらくなり，ノード数が増えることで，一部の都道府県では計算機のメモリ不足に陥る場合がある．これは，後の計算機実験で詳しく検証する．

# 第4章　ヒューリスティクスを用いた手法

本章では，選挙区割問題を解くヒューリスティクスを作り，その解から得られた選挙区の人口上限と下限を用いて，ZDD 構築を効率化する手法について提案する．

## 4.1　概要

3 章の人口制約付き ZDD 構築アルゴリズムでは，パラメータとして部分グラフの重み上限 $U$, 下限 $L$ を用いた．許容格差定数 $r$ を用いると，$r$ の値によって，$U, L$ の範囲が必要以上に広くなることや $U, L$ の範囲に解が一つも存在しないことがあり得る．既存手法では，$r$ の値を都道府県によって手動で調整するものがほとんどである．そこで，本研究は選挙区割問題をヒューリスティクスで解き，解として得られる選挙区から，最大人口の選挙区の重みを $U$, 最小人口の選挙区の重みを $L$ と定義する手法を提案する．$U, L$ が最適解に近くなればなるほど，ZDD の構築時に，最適解でない解候補の枝刈りの回数が多くなる．その結果，従来手法に比べてメモリの使用量の減少及び計算時間の短縮が期待できる．

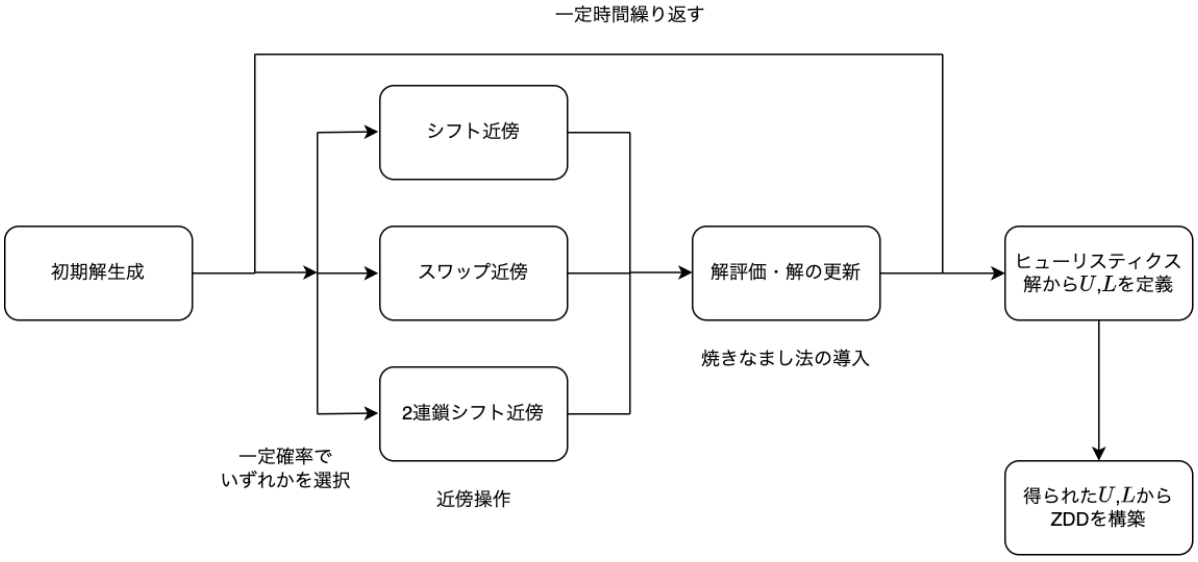



**図 4.1:** ヒューリスティクスを用いた提案手法

ヒューリスティクスでは，解を $x$ とおき，評価に用いるスコアを，$f(x) = \frac{\max(P)}{\min(P)}$ として，これを最小化することを目指す．

提案手法の全体像を図 4.1 に示す．始めに選挙区割問題の初期解を生成し，次に一定の確率でシフト近傍，スワップ近傍，2 連鎖シフト近傍のいずれかを選択する．選択した近傍操作を行って近傍解を生成し，近傍解が選挙区割の制約（選挙区が $d$ 個のみ存在する，飛び地がない）を満たすか判定をする．制約を満たしている場合には，スコアを計算し，焼きなまし法により解の遷移を行う．これを一定時間繰り返すことで，解を得ることができる．その解から，$U = \max(P), L = \min(P)$ を定義し，それらを用いて ZDD を構築する．次節から，初期解生成や近傍探索のアルゴリズムについて詳細に説明する．

## 4.2 初期解生成

初期解は，市区町村を表現したグラフから選挙区を構成する部分グラフの $d$ 個の根を定め，それぞれの根から幅優先探索 (BFS) を行い，探索した頂点を部分グラフに含めることで構築する．初期解生成の擬似コードを **Algorithm 4** に示す．

**Algorithm 4** MakeInitialSolution

**Require:** $n, d, ave, adj\_list, w$

**Ensure:** $x$

$x \leftarrow [-1] * n$
$P \leftarrow [0] * d$
頂点重みが最も大きいものから順に $d$ 個の頂点番号を配列 $root$ に保存する．
$root$.reverse()
**for** $i = 1, \ldots, d$ **do**
    $queue \leftarrow \phi$
    $x[root[i]] \leftarrow i$
    $P[i] \leftarrow P[i] + w[nv]$
    $queue$.enqueue($root[i]$)
    **while** $queue$.size() $\neq 0$ and $P[i] < ave$ **do**
        $v \leftarrow queue$.dequeue()
        **for all** $nv \in adj\_list[v]$ **do**
            **if** $P[i] \geq ave$ **then**
                break
            **else if** $x[nv] = -1$ **then**

```
16:                 x[nv] ← i
17:                 P[i] ← P[i] + w[nv]
18:                 queue.enqueue(nv)
19:             end if
20:         end for
21:     end while
22: end for
23: while x の要素に −1 が含まれる do
24:     for vi = 1, ..., n do
25:         if x[vi] = −1 then
26:             for all nv ∈ adj_list[v] do
27:                 if x[nv] ≠ −1 then
28:                     x[vi] ← x[nv]
29:                     break
30:                 end if
31:             end for
32:         end if
33:     end for
34: end while
35: return x
```

**Algorithm 4** MakeInitialSolution の入力と出力については以下の通りである．

**入力**

- $n$：グラフの頂点 $v$ の個数
- $d$：区割（グラフ分割）数
- $ave$：一選挙区の平均人口
- $adj\_list$：グラフの隣接リスト
- $w$：グラフの頂点重み

**出力**

- $x$：頂点番号を添字とした配列で，選挙区（部分グラフ）のラベル（値は $1, \ldots, d$）が保存されている．初期値は $-1$.

2 行目の $P$ は部分グラフごとの重み和を保存する変数である．配列 $root$ は，BFS の始点となる $d$ 個の頂点を含める．3 行目にもある通り，$root$ は頂点重みが最も大きいものから順に指定する．4 行目で $root$ を逆順にするが，これは $root$ の頂点重みが小さいものから探索を行うためである．BFS では，キューを用いて探索を行い，探索した頂点を属する部分グラフに加えていく（5-22 行目）．ラベル $i$ の探索では，頂点 $nv$ を追加する際に $x[nv]$ に $i$ を代入し，部分グラフの重み和 $P[i]$ に $w[nv]$ を加算する．そして，重み和 $P[i]$ が平均重み $ave$ 以上になると，その部分グラフでの探索を中断する（10,13 行目）．これを全ての根 $root[1], \ldots, root[d]$ に対して行う．全ての BFS を終えた後，ラベルが付かなかった（$x$ の値が $-1$ の）頂点に対して隣接頂点を参照し，そこからラベルを割り振る処理を行なっている（23-34 行目）．最後に $x$ を出力し，これを初期解とする．

## 4.3 近傍操作

近傍操作とは，現在の解から少し形を変えた近傍解を生成する操作のことである．選挙区割問題における近傍操作では，次の 3 つを定義する．

- シフト近傍
- スワップ近傍
- 2 連鎖シフト近傍

3 つの近傍操作は確率を用いてその都度ランダムに選択する．近傍解は，問題の制約を満たさない（部分グラフが $d$ 個存在しない，連結でない）場合があるため，その都度 BFS などを用いて判定を行う．近傍解が問題の制約を満たしている場合，焼きなまし法の手順に沿って解を改善していく．

### 4.3.1 シフト近傍

シフト近傍は，1 頂点のラベルを変更する近傍操作である．シフト近傍の例として，図 4.2 を用いて説明する．青色の頂点集合からなる部分グラフを $\mathcal{G}_1$，赤色の頂点集合からなる部分グラフを $\mathcal{G}_2$ とする．$\mathcal{G}_1$ には頂点 $v_1$ が含まれており，$v_1$ は，別の部分グラフ（$\mathcal{G}_2$）の頂点 $v_2$ と隣接している．シフト近傍では，頂点 $v_1$ のラベル $x[v_1]$ を頂点 $v_2$ のラベル $x[v_2]$ に代入することで，$v_2$ を $\mathcal{G}_2$ から $\mathcal{G}_1$ に含めることができる．こうしてできた新たな解 $x$ を近傍解とする．

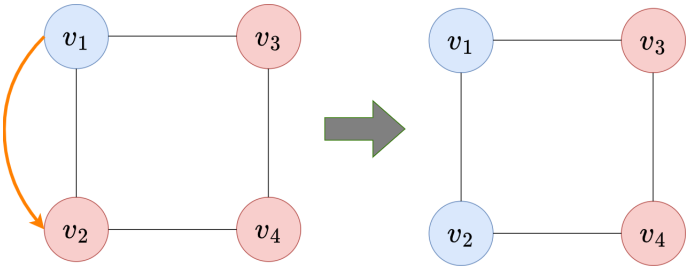



**図 4.2:** シフト近傍

### 4.3.2 スワップ近傍

スワップ近傍は，2 つの部分グラフに含まれる頂点のラベルを交換する近傍操作である．スワップ近傍の例として，図 4.3 を用いて説明する．先ほどと同様に，青色の頂点集合からなる部分グラフを $\mathcal{G}_1$，赤色の頂点集合からなる部分グラフを $\mathcal{G}_2$ とする．$\mathcal{G}_1$ には頂点 $v_1$ が含まれており，$v_1$ は．$\mathcal{G}_2$ の頂点 $v_3$ と隣接している．また，$\mathcal{G}_2$ には頂点 $v_4$ が含まれており，$v_4$ は．$\mathcal{G}_1$ の頂点 $v_2$ と隣接している．スワップ近傍では，頂点 $v_1$ のラベル $x[v_1]$ を頂点 $v_3$ のラベル $x[v_3]$ に代入し，頂点 $v_4$ のラベル $x[v_4]$ を頂点 $v_2$ のラベル $x[v_2]$ に代入する．すると，$\mathcal{G}_1$ と $\mathcal{G}_2$ に含まれる頂点がそれぞれ一つずつ交換された近傍解ができる．このように，2 つの部分グラフ間で頂点を交換することで，シフト近傍のみでは遷移できない近傍解が作られ，解の精度が上がりやすくなる．

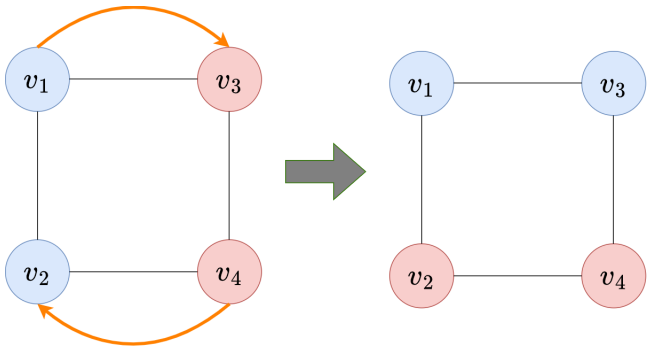



**図 4.3:** スワップ近傍

### 4.3.3 2 連鎖シフト近傍

2 連鎖シフト近傍とは，一つの部分グラフを主軸として，2 回のシフト近傍操作を同時に行うものである．2 連鎖シフト近傍の例として，図 4.4 を用いて説明する．青色の頂点集合からなる部分グラフを $\mathcal{G}_1$，赤色の頂点集合からなる部分グラフを $\mathcal{G}_2$, 緑色の頂点集合からなる部分グラフを $\mathcal{G}_3$ とする．$\mathcal{G}_2$ を主軸に置くと，$\mathcal{G}_1$ に頂点 $v_2$ を渡すが，$\mathcal{G}_3$ から，

頂点 $v_6$ を受け取る．このようにある部分グラフが，頂点を 1 つ渡し，1 つ受け取る操作を行うことで，シフト近傍やスワップ近傍では得られなかった近傍解を作成することができる．

なお，一般的に連鎖シフトは，3 回以上シフト近傍操作を行うものも存在するが，選挙区割問題では必要性は低いと判断し，実装は行わなかった．

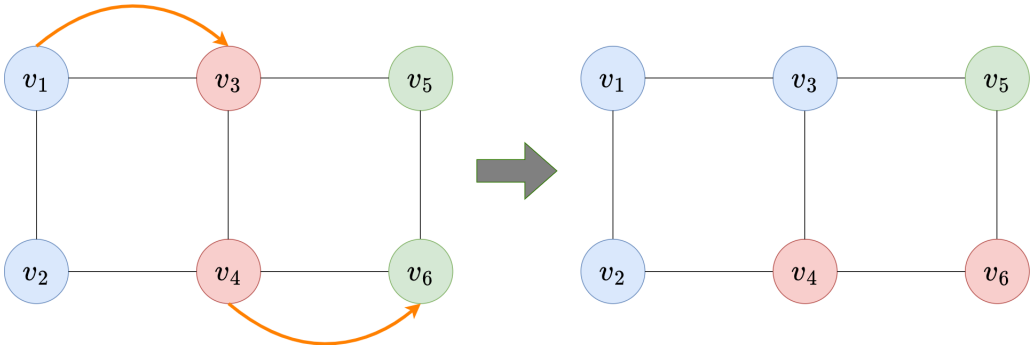



**図 4.4:** 2 連鎖シフト近傍

## 4.4 焼きなまし法

焼きなまし法とは，現在の解 $x$ から作られた近傍解 $x'$ が改善解なら移動し，改悪解ならばスコアの値の変化量 $\Delta = f(x') - f(x)$ に応じた確率で移動することで，質の低い局所最適解から脱出し，より良い解を探索する手法である [7]．選挙区割問題での改善解は $\Delta \leq 0$，改悪解は $\Delta > 0$ となる．改悪解に遷移する確率は，温度と呼ばれるパラメータ $t$ により $e^{-\Delta/t}$ と設定する．

焼きなまし法の手続きを以下にまとめる．

**Step 1:** 初期解 $x$ を生成する．暫定解 $x^\natural = x$ とする．初期温度 $t_s$ を設定し，$t = t_s$ とする．

**Step 2:** 3 つの近傍操作から 1 つをランダムに選択し，現在の解 $x$ から選んだ近傍操作で生成した近傍解を $x'$ とする．

**Step 3:** $x'$ が問題の制約を満たしているか判定し，満たしていなければ **Step 2** に戻る．

**Step 4:** $\Delta = f(x') - f(x)$ とし，$\Delta \leq 0$ ならば確率 1，$\Delta > 0$ ならば確率 $e^{-\Delta/t}$ で $x = x'$ とする．

**Step 5:** $f(x) < f(x^\natural)$ ならば $x^\natural = x$ とする．

**Step 6:** 終了条件を満たせば $x^\natural$ を出力して終了．そうでなければ，温度 $t$ を更新して **Step 2** に戻る．

焼きなまし法を実装するには，初期温度，温度の更新方法，アルゴリズムの終了条件などを定める必要がある．初期温度 $t_s$ は，一回の遷移で動きうるスコア幅の最大値程度にしたいため，考えられるスコアの最大値 – 最小値に近い値を設定する．スコアの最小値は全てのケースで 1 を下回ることはない．考えられるスコアの最大値は概算で求める．まず，グラフの頂点重みを昇順にソートしたものを $w'_1, \ldots, w'_l$ とおく．部分グラフの最大重みは，連結であることを考慮しなければ $\sum_{i=d}^{l} w'_i$，最小重みは $w'_1$ であるから，求めるスコアの最大値は，$(\sum_{i=d}^{l} w'_i)/w'_1$ となる．よって，初期温度は

$$t_s = \frac{\sum_{i=d}^{l} w'_i}{w'_1} - 1$$

と定める．また，終了温度 $t_e$ は一回の遷移で動きうるスコア幅の最小値程度にしたい．ただし，スワップ近傍で限りなく小さいスコア幅で遷移することもあるため，今回は $t_e = 0$ とし，現在温度が終了温度に達した（$t = t_e$）場合にアルゴリズムは終了条件を満たしたと判定する．

温度の更新方法については，制限時間 $T_{limit}$，経過時間 $T_{now}$ を用いて，現在温度 $t$ を定義すると，

$$t = t_s - (t_s - t_e) \cdot \frac{T_{now}}{T_{limit}}$$

となる．$T_{limit}$ は問題の規模に合わせて手動で調整する．

# 第5章　計算機実験

## 5.1　概要

本章では，提案手法であるヒューリスティクス解を用いたZDD構築の計算機実験を行い，既存手法の人口制約のないZDD構築手法，人口制約ありで許容格差定数を用いたZDD構築手法との比較をし，提案手法の評価及び考察を行う．許容格差定数$r$は，ほぼ全ての都道府県での制約を満たす値として1.4を採用した．

## 5.2　実験環境

実験環境は，次の通りである．

- OS：Ubuntu 20.04.5 LTS
- CPU：Intel Xeon E5-2687W v4（3.0GHz）
- メモリ：512GB

プログラムはC++17によって実装し，gccを用いて -O3, -march=native オプションを付与してコンパイルを行った．また，ライブラリとして，SAPPOROBDD[1]，TdZdd[2]を利用した．

## 5.3　入力データ

入力データとして，国土交通省が公開している「国土数値情報　行政区域データ」と令和2年国勢調査結果の「人口等基本集計」を利用し，市区町村を頂点，隣接関係を辺，人口を頂点重みとしたグラフを作成した．本論文ではいくつかの都道府県のインスタンスを抜粋して掲載する．入力データについて，各インスタンスのパラメータを表5.1にまとめた．

[1] https://github.com/Shin-ichi-Minato/SAPPOROBDD
[2] https://github.com/kunisura/TdZdd

**表 5.1:** 入力データ

| Name | $\|V\|$ | $\|E\|$ | $d$ |
|---|---|---|---|
| $G_1$(Aomori) | 40 | 84 | 3 |
| $G_2$(Miyagi) | 39 | 86 | 5 |
| $G_3$(Yamagata) | 35 | 85 | 3 |
| $G_4$(Fukushima) | 59 | 144 | 4 |
| $G_5$(Ibaraki) | 44 | 94 | 7 |
| $G_6$(Nagano) | 77 | 187 | 5 |
| $G_7$(Aichi) | 69 | 173 | 16 |
| $G_8$(Osaka) | 72 | 168 | 19 |

表 5.1 の $|V|$ は頂点数，$|E|$ は辺数，$d$ は分割数を表している．また，頂点の各重みについて分布図を図 5.1 にまとめた．横軸が頂点の重み，縦軸が度数を表している．

## 5.4 実験結果

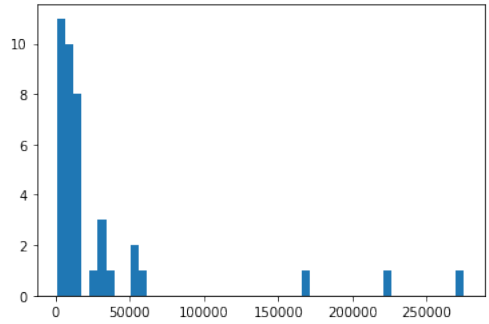
### 5.4.1 人口制約なし

第 3.3.1 項にて述べた人口制約のない区割列挙アルゴリズムを，各インスタンスにて行った結果を表 5.2 に示す．

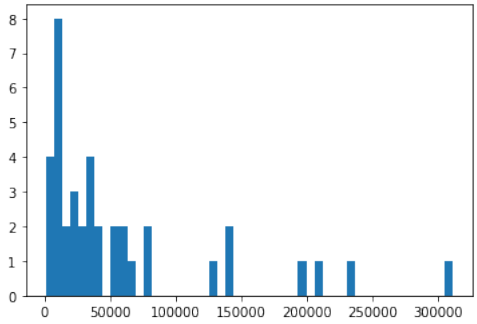
**表 5.2:** 人口制約なし区割列挙

| Name | node | solve | time(sec) | memory(MB) |
|---|---|---|---|---|
| $G_1$(Aomori) | 5196 | 10452641 | 0.02 | 6 |
| $G_2$(Miyagi) | 2891 | 1.98E+10 | 0.01 | 4 |
| $G_3$(Yamagata) | 2672 | 490516246 | 0.01 | 4 |
| $G_4$(Fukushima) | 233446 | 1.51E+14 | 1108.69 | 16859 |
| $G_5$(Ibaraki) | 10757 | 3.24E+13 | 0.02 | 6 |
| $G_6$(Nagano) | 48612 | 3.82E+17 | 1.48 | 472 |
| $G_7$(Aichi) | 1145106 | 3.02E+29 | 2.41 | 395 |
| $G_8$(Osaka) | 955147 | 1.73E+30 | 0.5 | 92 |

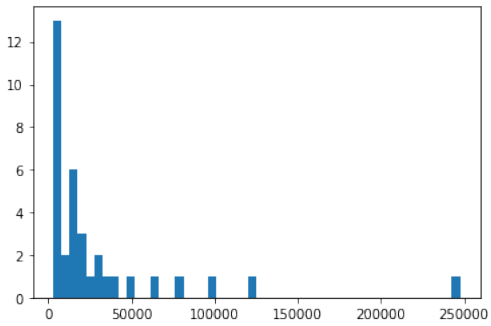
表の各カラムについて，Name はインスタンス名，Node は構築した ZDD のノード数，



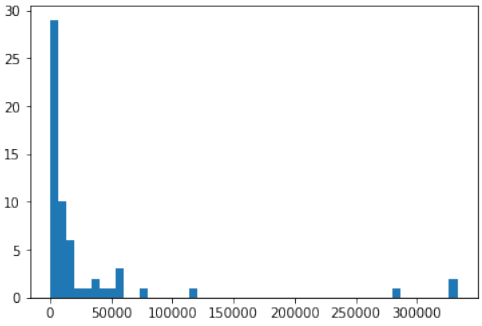

(a) $G_1$



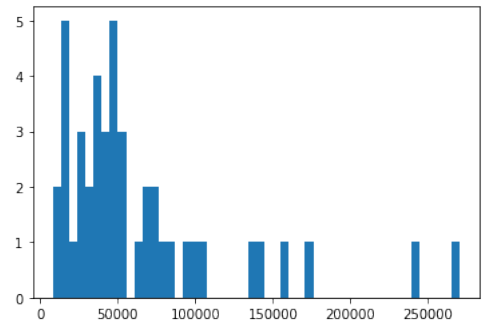

(b) $G_2$



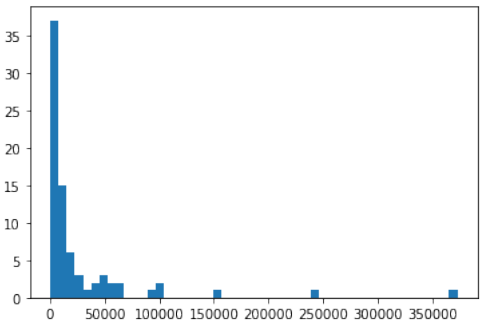

(c) $G_3$



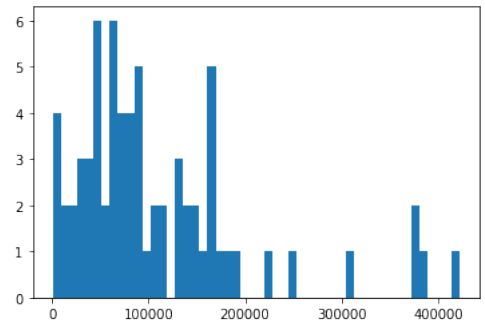

(d) $G_4$



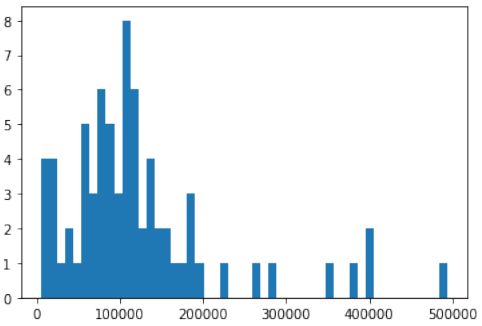

(e) $G_5$




(f) $G_6$




(g) $G_7$




(h) $G_8$

**図 5.1:** 頂点の重み分布

solve は ZDD にて列挙した解の個数，time は ZDD 構築にかかった時間（単位は秒），memory は計算機で使用したメモリ容量（単位は MB）を表している．人口制約がない場合，頂点に重みを保持する必要がないため，$G_4$ を除いて数秒以内に計算を終えることができた．$G_4$ はこの中で一番時間がかかるものの，計算時間やメモリ使用量は一般的に許容範囲内である．ただし，解の個数は非常に多く，分割数の最も多い $G_8$ では 1.73E+30 個という結果になった．

### 5.4.2 人口制約あり：許容格差定数を用いる場合

許容格差定数 $r = 1.4$ とし，各インスタンスごとに重み下限 $L$, 重み上限 $U$ を第 3.3.2 項にて述べた計算方法にて求めた．それを用いて ZDD を構築した結果を表 5.3 に示す．

**表 5.3:** $r = 1.4$ とした区割列挙

| Name | $L$ | $U$ | node | solve | time(sec) | memory(MB) |
|---|---|---|---|---|---|---|
| $G_1$(Aomori) | 325785 | 509759 | 23749 | 668154 | 2.25 | 405 |
| $G_2$(Miyagi) | 348787 | 596814 | 6581 | 40106 | 3.38 | 652 |
| $G_3$(Yamagata) | 281059 | 439776 | 319171 | 7493473 | 33.81 | 6702 |
| $G_4$(Fukushima) | 353649 | 585129 | N/A | N/A | N/A | N/A |
| $G_5$(Ibaraki) | 305000 | 542408 | 1077156 | 36745326 | 212.75 | 30690 |
| $G_6$(Nagano) | 310304 | 530966 | N/A | N/A | N/A | N/A |
| $G_7$(Aichi) | 342837 | 643865 | N/A | N/A | N/A | N/A |
| $G_8$(Osaka) | 337316 | 637772 | N/A | N/A | N/A | N/A |

node,solve,time,memory に値があるものは計算が出来ており，N/A となっているものは，メモリ不足により計算が行えなかったことを表している．$G_1, G_2, G_3, G_5$ では解を求められたが，$G_4, G_6, G_7, G_8$ では解を求められなかった．

### 5.4.3 人口制約あり：ヒューリスティクスの結果を用いる場合

本節の実験では，第 4 章で述べたヒューリスティクスを用いて，各インスタンスにて人口上下限制約 $L, U$ を求め，それを利用して ZDD の構築を行なう．ヒューリスティクスでの計算時間は，各 120 秒とし，10 回実験を行った中で最良の値を採用した．各インスタンスの実験結果を表 5.4 に示す．

**表 5.4:** ヒューリスティクスを用いた区割列挙

| Name | $L$ | $U$ | node | solve | time(sec) | memory(MB) |
|---|---|---|---|---|---|---|
| $G_1$(Aomori) | 226194 | 555698 | 34609 | 2001248 | 2.77 | 460 |
| $G_2$(Miyagi) | 451162 | 467561 | 55 | 2 | 0.01 | 4 |
| $G_3$(Yamagata) | 355396 | 356505 | 4416 | 541 | 0.08 | 19 |
| $G_4$(Fukushima) | 459096 | 460480 | N/A | N/A | N/A | N/A |
| $G_5$(Ibaraki) | 391937 | 419212 | 1340 | 390 | 0.02 | 6 |
| $G_6$(Nagano) | 408772 | 410752 | 10320 | 17657 | 2.30 | 471 |
| $G_7$(Aichi) | 359399 | 553700 | 1847085 | 1.29E+14 | 89.02 | 12607 |
| $G_8$(Osaka) | 424530 | 569011 | N/A | N/A | N/A | N/A |

$G_1$ を除いて，$L$ と $U$ の値の範囲を縮小することに成功した．例えば，$G_6$ において $r = 1.4$ を用いた場合は $U - L = 220662$ であるが，ヒューリスティクスを用いた場合 $U - L = 1980$ となり，1/100 程度の圧縮を行うことに成功した．ZDD の構築においては，新たに $G_6, G_7$ で解を求めることが出来た．既存手法で解が求められているものについても，$G_1$ 以外は 4 つの指標の値が大幅に削減されている．

## 5.5 補足実験

本節では，許容格差の値の変化によって解の個数がどのように減るのかを確認するため，第 5.4.2 項の実験から $G_3, G_5$ において $r$ の値を細かく変化させ，区割列挙を行った場合について検証を行った．表 5.5 は $G_3$ について，表 5.6 は $G_5$ についての実験結果である．

どちらのインスタンスについても，解が存在する限り $r$ の値を小さくすればするほど，4 つの指標の値が削減されることがわかった．

## 5.6 考察

人口制約のない区割列挙は，実行時間やメモリ使用量のパフォーマンスが行った実験の中で最もよく，全てのインスタンスにおいて解を得ることが出来た．$G_4$ の実行が遅い点については，インスタンスのグラフのサイズが大きいことに加え，グラフの形状上，フロンティアのサイズが大きくなってしまったことが原因だと考えられる．この手法は解の個数が非常に多く，ノードを参照しても各部分グラフの重みを保持していないため，ZDD

**表 5.5:** $G_3$ で $r$ を変化させた列挙

| $r$ | node | solve | time(sec) | memory(MB) |
|---|---|---|---|---|
| 1.40 | 319171 | 7493473 | 33.89 | 6702 |
| 1.35 | 289106 | 5873171 | 27.25 | 5164 |
| 1.30 | 255214 | 4403896 | 22.02 | 3972 |
| 1.25 | 222857 | 3111414 | 15.99 | 2791 |
| 1.20 | 188371 | 2014623 | 10.92 | 1896 |
| 1.15 | 152860 | 1146740 | 6.96 | 1133 |
| 1.10 | 111220 | 527439 | 3.64 | 570 |
| 1.05 | 67194 | 135951 | 1.36 | 212 |
| 1.025 | 40548 | 36118 | 0.61 | 106 |
| 1.010 | 19378 | 6503 | 0.27 | 54 |

**表 5.6:** $G_5$ で $r$ を変化させた列挙

| $r$ | node | solve | time(sec) | memory(MB) |
|---|---|---|---|---|
| 1.40 | 1077156 | 36745326 | 212.75 | 30690 |
| 1.35 | 687919 | 17015647 | 101.19 | 16703 |
| 1.30 | 405073 | 7114993 | 49.12 | 7968 |
| 1.25 | 229884 | 2594285 | 19.28 | 3287 |
| 1.20 | 104224 | 746808 | 6.15 | 1079 |
| 1.15 | 39771 | 138923 | 1.64 | 284 |
| 1.10 | 10311 | 8305 | 0.26 | 43 |
| 1.05 | 1154 | 169 | 0.03 | 8 |
| 1.025 | 0 | 0 | 0.02 | 6 |

の終端から列挙した解の精度を測ることが不可能である．よって，選挙区割を見つける用途では実用的ではないことがわかった．

人口制約付きの区割列挙だが，許容格差定数 $r = 1.4$ としたときには，ノード数が大幅に増加することからメモリ不足になり，頂点数や辺数，分割数が大きいインスタンスでは解を得られなかった．ただし，今回解を導出できた $G_5$(Ibaraki) より頂点 (市区町村) の数が少ない都道府県は 47 個中 30 個存在することから，$r = 1.4$ のままで少なくとも過半数の都道府県で区割の列挙ができると考えられる．また，$r$ の値を小さくすると，いくつかの都道府県では解が 0 個になるものの補足実験でメモリが削減されることがわかったため，列挙可能な都道府県は増えると考えられる．

ヒューリスティクスでは，$G_2, G_3, G_5, G_6$ で導出した $L$ と $U$ の値がほぼ一致しており，解の個数も少ないことから最適解に近い区割を導出できたことがわかる．$L$ と $U$ の値の差がないとき，ZDD の枝刈りがよく働くため実行時間，メモリ使用量の双方で非常に良い結果が得られている．しかし，$G_1$ と $G_7$ は $L$ と $U$ の差が大きく，解の個数も大きいことから，質の悪い局所解に陥っている可能性が高い．また，例外として $G_4$ は $U - L = 1384$ と非常に小さい値であるにも関わらず，解を得ることが出来なかった．これも先述した通りグラフの形状からフロンティアのサイズが大きくなることが原因である．さらに，$G_8$ では，最適解がおよそ $r = 1.34$ の範囲内にあることが知られており，最適解で $L$ と $U$ を導出したとしても，ZDD を構築することはできないと考えられる．このようなインスタンスで ZDD で列挙をする場合は，$U$ と $L$ の値を変化させる以外の手法を考える必要がある．

# 第6章　おわりに

## 6.1　まとめ

本研究では，選挙区割問題に対して，ヒューリスティクスとして焼きなまし法を実装し，得られた解を利用して効率的に ZDD を構築する手法を提案した．許容格差 $r$ で制約を決める既存手法では，人の手で ZDD が構築可能な値を探索しなければいけなかったが，提案手法によりその作業を不要にし，さらに ZDD 構築において計算時間と消費メモリを大幅に削減する効果が得られることを確認した．

## 6.2　今後の課題

ヒューリスティクスの解は，一部のインスタンスで最適解から大きく離れたものであったため，アルゴリズムを改善する必要がある．また，今回用いたフロンティア法では，ヒューリスティクスを改良しても解を得られない都道府県が存在するため，全ての都道府県を列挙するためには，別の ZDD 構築アルゴリズムが必要となる．関連する研究として，ZDD を反復的にトップダウンで構築するサブセッティング法 [8] を選挙区割問題に適用した事例 [9] が存在する．サブセッティング法にヒューリスティクスで求めた重み上下限制約を組合せることで，全ての都道府県について選挙区をより高速に列挙できる可能性が考えられる．

# 謝辞

本研究を進めるにあたり，大変多くのご指導，ご助言を頂いた中央大学大学院理工学研究科情報工学専攻の今堀慎治教授，ZDD の実装にあたりライブラリの提供やご相談に快く応じて頂いた京都大学大学院情報学研究科通信情報システム専攻の川原純准教授に深く感謝いたします．また，多大なるご助言，ご協力を頂いた今堀研究室の皆様には大変お世話になりました．心から感謝いたします．

最後に，大学 4 年間に加え，大学院に 2 年間通わせていただいた両親に深く感謝いたします．

# 関連発表

1. 千原良太，今堀慎治：選挙区割問題に対するヒューリスティクスを用いたZDD構築の効率化，日本オペレーションズ・リサーチ学会 2023年春季研究発表会，2023年3月8日

# 参考文献


[1] 一森哲夫：議席配分の数理-選挙制度に潜む 200 年の数学-，近代科学社 (2022).

[2] 根本俊男，堀田敬介：区割画定問題のモデル化と最適区割の導出，オペレーションズ・リサーチ，vol. 48，no. 4，pp. 300-306 (2003).

[3] Kawahara, J. et al.: Generating All Patterns of Graph Partition Within a Disparity Bound, WALCOM, pp. 119-131 (2017).

[4] Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems, Proceedings of the 30th international Design Automation Conference, pp. 272-277 (1993).

[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] Sekine, K., Imai, H. Tani, S.: Computing the Tutte polynomial of a graph of moderate size, Proceedings of the 6th International Symposium on Algorithms and Computation (ISAAC), LNCS-1004, pp. 224-233 (1995).

[7] 梅谷俊治：しっかり学ぶ数理最適化　モデルからアルゴリズムまで，講談社 (2020).

[8] Iwashita, H. and Minato, S.: TCS Technical Report Efficient Top-Down ZDD Construction Techniques Using Recursive Specifications, TCS Technical report (2013).

[9] 山崎宏紀：部分グラフ列挙問題に対する反復的トップダウン ZDD 構築手法の研究，京都大学大学院情報学研究科 修士課程通信情報システム専攻 修士論文 (2022).