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

情報工学専攻　千原 良太
Ryota CHIHARA

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合せて区割を構成し，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．

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許容格差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消費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．

1　はじめに

日本の衆議院議員選挙の小選挙区制における区割画定問題 (以下「選挙区割問題」と称する) とは，各都道府県にあらかじめ定められた選挙区数に対し，市区町村を複数組合せて条件を満たす選挙区を構築する問題である．選挙区割問題の解法の一つとしてゼロサプレス型二分決定グラフ (ZDD) による列挙法 [1] がある．本研究では，ZDD による解法にヒューリスティクスによる近似解法を組み合わせることで，従来より少ないメモリ使用量と実行時間で解を列挙する手法を提案する．

2　選挙区割問題の定義

選挙区割問題を考えるにあたって，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．

3　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) [2] は，組合せ集合を表現する非巡回有向グラフ (DAG) を用いたデータ構造である．ZDD の大きな特徴は，指数的な個数の組合せを効率的に表現できる点で，二分決定木から不要な節点の削除・共有を行うことで，疎な組合せ集合を高速に列挙することができる．

また，ZDD をトップダウンに構築する手法としてフロンティア法がある．この手法では，根から順に節点を作成し，処理中のラベルをもつ節点集合の中で，共通する節点はその場で共有を行い，制約を満たさない枝はその場で枝刈りすることで，高速に ZDD を構築すること

ができる．

4 選挙区割列挙アルゴリズム

選挙区割問題では，ZDD の節点のラベルに辺集合 E をそれぞれ割り当てることで，選んだ辺を選挙区の連結部分とし，連結部分からなる部分グラフで選挙区を表現する．節点には辺のラベルの他に，フロンティア上の市区町村の接続関係，確定した部分グラフの個数，部分グラフ中の各連結成分の重みの情報をもつ．また，制約として選挙区の人口下限 L と人口上限 U を最適な選挙区割を含むような値に定めておく．一般的には，許容格差定数 r と呼ばれるものを用いる．市区町村の人口の和を W として，$L = \frac{W}{r\cdot(d-1)+1}$, $U = \frac{r\cdot W}{r+(d-1)}$, を求める．これらの情報をもとにフロンティア法により ZDD を構築し，確定した部分グラフの個数が m を超えた場合や L や U の制約を満たさない部分グラフが存在すると確定した場合に枝刈りを行い，最適な選挙区割の解を含む ZDD を得ることができる．

5 ヒューリスティクスを用いた手法

選挙区割問題の場合，ZDD の節点に持つ情報が複数あり，節点の共有が行うのが難しい構造となる．そこで，ヒューリスティクスでできる限り最適な選挙区割に近い実行可能解を求め，この解の選挙区人口を用いて U と L の値を定めることで，ZDD の節点の削除を効率よく行い，実行時間の短縮と省メモリ化を目指す．

本研究では，ヒューリスティクスとして焼きなまし法を実装する．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．近傍操作は，次の 3 つを用いる．各頂点には，どの部分グラフに属しているかを示すラベルが割り振られている．

- **シフト近傍**：1 頂点のラベルを隣接している部分グラフのラベルに変更する．
- **スワップ近傍**：2 つの部分グラフ間で頂点のラベルを入れ替える．
- **2 連鎖シフト近傍**：ある部分グラフにおいて，頂点を 1 つ渡すシフト近傍と頂点を 1 つ受け取りシフト近傍を同時に行う．

選挙区割問題におけ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 $w'_1, \ldots, w'_n$ 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 $\sum_{i=d}^{n} w'_i$，最小重みは w'_1 であるから，求めるスコアの最大値は，$(\sum_{i=d}^{n} w'_i)/w'_1$ となる．よって，初期温度は

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

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．

6 計算機実験

6.1 実験概要

令和 2 年度の国勢調査のデータを利用し，各都道府県ごとにグラフを作成した．本研究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．各インスタンスのパラメータを表 1 にまとめた．

列挙手法は次の 3 通りで実験を行った．

- **人口制約を考慮しない列挙** グラフの重みを考慮せず，L, U による制約を無視して ZDD を構築す

表 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

る．この手法のみ ZDD には部分グラフの重みに関する情報を持たない．

- **許容格差定数 r を用いた列挙** 4 章で述べた許容格差定数を用いて L, U を計算し，ZDD を構築する．
- **ヒューリスティクスを用いた列挙** 5 章で述べたヒューリスティクスを用いて L, U を求め，ZDD を構築する．

r はほぼ全ての都道府県の列挙が可能な値として 1.4 を採用した．実験環境は CPU が Intel Xeon E5-2687W-v4 (3.00GHz)，メモリが 512GB のマシンを利用し，プログラムは C++17，ライブラリとして SAPPOROBDD[*1] と TdZDD[*2] を使用した．

6.2　実験結果と考察

表 2　$r = 1.4$ での L, U の値

Name	L	U
G_1	325,785	509,759
G_2	348,787	596,814
G_3	281,059	439,776
G_4	353,649	585,129
G_5	305,000	542,408
G_6	310,304	530,966
G_7	342,837	643,865
G_8	337,316	637,772

表 2 では，既存手法の $r = 1.4$ を用いた U, L の値，表 3 では，ヒューリスティクスを用いて導出した U と L の値を示している．表 4，表 5，表 6 は各実験ごとの実行結果を示しており，node は ZDD のノード数，solve は解の個数，time は実行時間，mem はメモリ消費量を示している．

人口制約を考慮しない列挙は，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

[*1] https://github.com/Shin-ichi-Minato/SAPPOROBDD

[*2] https://github.com/kunisura/TdZdd

表 3　ヒューリスティクスから得た L, U の値

Name	L	U
G_1	226,194	555,698
G_2	451,162	467,561
G_3	355,396	356,505
G_4	459,096	460,480
G_5	391,937	419,212
G_6	408,772	410,752
G_7	359,399	553,700
G_8	424,530	569,011

表 4　人口制約を考慮しない列挙の結果

Name	node	solve	time(sec)	mem(MB)
G_1	5,196	10,452,641	0.02	6
G_2	2,891	1.98E+10	0.01	4
G_3	2,672	490,516,246	0.01	4
G_4	233,446	1.51E+14	1,108.69	16,859
G_5	10,757	3.24E+13	0.02	6
G_6	48,612	3.82E+17	1.48	472
G_7	1,145,106	3.02E+29	2.41	395
G_8	955,147	1.73E+30	0.5	92

表 5　$r = 1.4$ を用いた列挙の結果

Name	node	solve	time(sec)	mem(MB)
G_1	23,749	668,154	2.25	405
G_2	6,581	40,106	3.38	652
G_3	319,171	7,493,473	33.81	6,702
G_4	N/A	N/A	N/A	N/A
G_5	1,077,156	36,745,326	212.75	30,690
G_6	N/A	N/A	N/A	N/A
G_7	N/A	N/A	N/A	N/A
G_8	N/A	N/A	N/A	N/A

表 6　ヒューリスティクスを用いた列挙の結果

Name	node	solve	time(sec)	mem(MB)
G_1	34,609	2,001,248	2.77	460
G_2	55	2	0.01	4
G_3	4,416	541	0.08	19
G_4	N/A	N/A	N/A	N/A
G_5	1,340	390	0.02	6
G_6	10,320	17,657	2.30	471
G_7	1,847,085	1.29E+14	89.02	12,607
G_8	N/A	N/A	N/A	N/A

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 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．

許容格差定数 r を用いた列挙だが，$r = 1.4$ 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は区割の列挙ができると考えられる．

ヒューリスティクスを用いた列挙では，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．

7　おわりに

7.1　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．

7.2　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の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を選挙区割問題に適用した事例 [3] 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．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] Kawahara, J. et al.: Generating All Patterns of Graph Partition Within a Disparity Bound, WALCOM, pp. 119-131 (2017).

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

[3] 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).