広島経済大学研究論集
第39巻第３・４号　2016年12月
http://dx.doi.org/10.18996/kenkyu2016390303

複体の描画点数低減アルゴリズム

田中章司郎*・坂口　隼**

1. はじめに

地図情報は，グラフとしての頂点・辺・面などの位相幾何学的な情報（複体，complex）とそれに付随した位置・長さ・面積などの計量的な情報からなる[1)]。複体とは，一般に単体を組み合わせてできる図形のことである[2)]。ここで単体とは，点，線分，三角形，四面体等のことをいい，これらはそれぞれ０次元単体，１次元単体，２次元単体，３次元単体とも呼ばれる。地理情報の表現方式の１つである複体方式とは，平面を多角形に分割して作った複体によって地図上の行政区画等の境界線の地理情報を表現するものである[1)]。

地図境界線の簡略化によく用いられる方法に，Douglas-Peucker のアルゴリズム[3)] によって地理情報を表わす複体の多角形ごとに近似曲線を引くものがあり，この方法は統計分析の地図として広く用いられている SAS/GRAPH の最新版である Ver. 9.4 の PROC GREMOVE プロシジャ[4)] でも採用されている。しかし，この方法には簡略化の結果，多角形間に隙間や多角形同士で交差が生じるといった問題がある。

そこで本研究では，上記の問題点を持たない新たな描画点間引きアルゴリズムを提案し，国土地理院により整備された基盤地図情報である島根県行政区画の境界線データ[5)] に対して，このグラフ理論を応用したアルゴリズムを検証する。

* 広島経済大学経済学部教授
** 島根大学総合理工学部卒

2. グラフに対する描画点の間引きアルゴリズムの試行

この節では第４節の地図グラフデータ構造に先立って，グラフ理論の隣接リスト・隣接行列に基づく描画点の間引きアルゴリズムを検討する。

2.1 境界線データの変換

本研究では，行政区画境界線データ[5)] をグラフとして扱うためにグラフを表わすデータ構造へと変換を行ない，変換後のデータに対して描画点の間引きを行なう。

2.1.1 行政区画境界線データ

行政区画の境界線データは図2.1.1のように区画ごとに“<gml:exterior>”の下にその周囲の点の座標を時計回りの順に記述することによって各区画が多角形として表現されている。もし区画内に他の区画の飛び地や湖などの穴が存在する場合は区画の周囲の座標の後の“<gml:interior>”の下に穴の周囲の座標を時計回りに記述してある。座標記述部分の後には市町村名，市町村コード等が記述されており，その下からは別の区画の記述となっている。

2.1.2 グラフを表わすデータ構造

境界線データの多角形，つまり面の情報は用いずに，点の座標と辺の接続関係のみを抜き出して境界線データをグラフとして表わす方法として，本研究では隣接リスト表現と隣接行列表現の２通りを使用した。データ変換を行なうプログラムでは元データに現れる順に点に ID 番号を付与し，この ID 番号を元に辺の両端点を

```
<AdmArea　gml:id="K4_1">
<fid>fgoid:10-00100-7-10-3569</fid>
<lfSpanFr gml:id="K4_1-1">
<gml:timePosition>2008-03-31</gml:timePosition>
</lfSpanFr>
<devDate gml:id="K4_1-2">
<gml:timePosition>2008-03-31</gml:timePosition>
</devDate>
<orgGILvl>25000</orgGILvl>
<orgMDId>0-8</orgMDId>
<area>
<gml:Surface gml:id="K4_1-g" srsName="fguuid:jgd2000.bl">
<gml:patches>
<gml:PolygonPatch>
<gml:exterior>
<gml:Ring>
<gml:curveMember>
<gml:Curve gml:id="K4_1-3">
<gml:segments>
<gml:LineStringSegment>
<gml:posList>
 35.074251 133.136321
 35.074343 133.136346
          :
 35.074251 133.136321
</gml:posList>
</gml:LineStringSegment>
</gml:segments>
</gml:Curve>
</gml:curveMember>
</gml:Ring>
</gml:exterior>
<gml:interior>
<gml:Ring>
<gml:curveMember>
<gml:Curve gml:id="K4_1-4">
<gml:segments>
<gml:LineStringSegment>
<gml:posList>
 35.234903 132.968642
 35.235336 132.967642
          :
 35.234903 132.968642
</gml:posList>
</gml:LineStringSegment>
</gml:segments>
</gml:Curve>
</gml:curveMember>
</gml:Ring>
</gml:interior>
</gml:PolygonPatch>
</gml:patches>
</gml:Surface>
</area>
<type>その他</type>
<name>島根県 仁多郡奥出雲町</name>
<admCode>32343</admCode>
</AdmArea>
```

区画の周囲の座標記述部分

穴の周囲の座標記述部分

<name>島根県 仁多郡奥出雲町</name> ← 市町村名

<admCode>32343</admCode> ← 市町村コード

図2.1.1　境界線データ

表わすデータと点ごとの座標を表わすデータに変換する。点の識別は座標で行なうので変換後のデータでは異なる点同士で座標が一致することはない。間引きを行なうプログラムではデータ変換プログラムの結果として得られる２つのデータを入力データとし，同じ構造のデータを出力する。グラフは無向グラフとして扱った。

2.1.2.1　隣接リスト表現

隣接リスト表現とは，各点に対して，その点と隣接する全ての点（または接続する全ての辺）を隣接リストに列挙することでグラフを表わす方法である[6)]。隣接リスト表現に必要な記憶領域は点と辺の数に比例する。要素を隣接する点の ID 番号とした隣接リストを用いてグラフを表現した例を図2.1.2.1に示す。なおプログラム上では隣接リストは一方方向線状リストで表現し，点を表わす構造体にリストの先頭を指すポインタを持たせた。

2.1.2.2　隣接行列表現

隣接行列表現とは，グラフ上の点の隣接関係を表わす隣接行列を用いてグラフを表わす方法である。隣接行列は点の ID 番号を添字とし隣接している点同士を表わす成分を１，隣接していなければ０とすることでグラフを表現する。隣接行列に必要な記憶領域は $S(|V|^2)$（V：点の集合）となる。隣接行列表現によってグラフを表わした例を図2.1.2.2に示す。

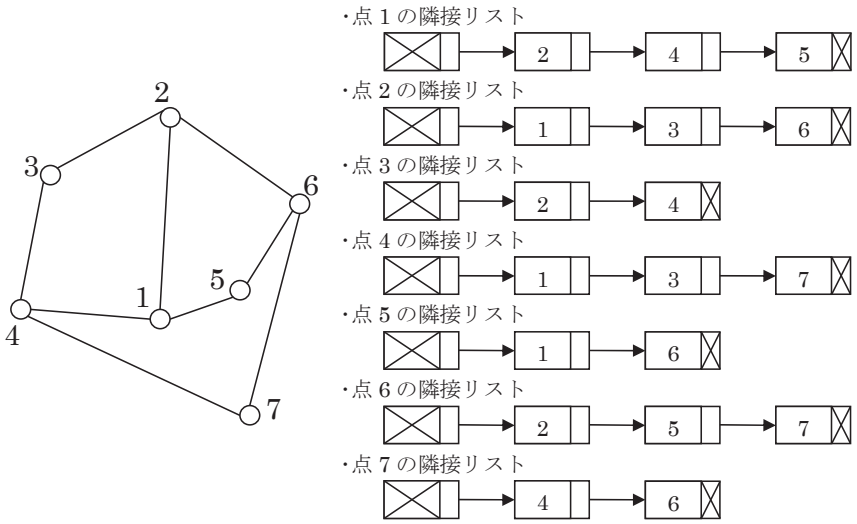


図2.1.2.1　隣接リスト表現の例

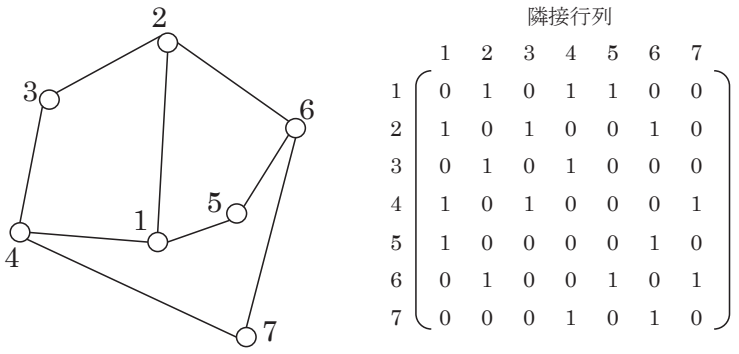


図2.1.2.2　隣接行列表現の例

2.2　間引きの手順

間引きによって面の数が変化することを避けるために，辺の短絡除去を用いて描画点の間引きを行なった。辺の短絡除去とは，グラフから辺を取り除き，その辺の両端を同一視することをいう[2)]。

図2.2の左側のグラフにおいて点１を間引く場合，点１に接続している辺（今回は辺 {1, 5} とする）の短絡除去を行なう。手順としては，まず短絡除去を行なう辺 {1, 5} 以外で点１に接続している辺 {1, 2}，{1, 4} を点１の代わりに点５に接続させる。その上で点１と辺 {1, 5} をグラフから取り除く。以上の操作によって点１を間引いたものが図2.2の右側のグラフである。

2.3　短絡除去を行なう辺の選択方法

前節でも説明したが，短絡除去とは辺の両端点を１つの点にまとめ，グラフからその辺を取り除くことをいう。間引きの際に短絡除去を行なう辺は以下の４つの条件を満たすものとする。

条件１．間引く点に接続している
条件２．長さが一定値未満である
条件３．短絡除去をすることによって面の数の変化や，辺同士の交差が生じない
条件４．条件１～３を満たす辺のうち，最も短い辺である

条件１では短絡除去することによって間引く点に接続している辺を他の点に接続しなおすことになる辺が選ばれる。条件２では短絡除去する辺を，その長さが簡略化の精度として設定する閾値未満となるものに限定する。条件３では３つの辺から成る面を構成する辺が除去されると面の数が変化するのでそのような辺を候補から外し，また短絡除去に伴なう辺の置き換えによって辺同士の交差が生じても境界線として描画した際に面の数が変化したように見えるのでこれも候補から外す。そして，いくつかの候補があった場合に短絡除去による境界線の変形が最も小さくなる辺を条件４で選択する。

作成した間引きプログラムでは点の ID 番号が若い順に点を見てゆき，以上の条件に合う辺の端点となっているものを全て間引いてゆく。

3.　グラフに対する間引き実行結果

3.1　隣接リスト表現を用いた場合の実行結果

点の数51,819，辺の数51,835（異なる面同士で一致する点と辺について重複する部分はカウントしない）である島根県行政区画の境界線データ（図3.1.1）を，隣接リスト表現を用いてグラフとして表わした上で間引きを行なった結果である境界線を図3.1.2に，比較のために

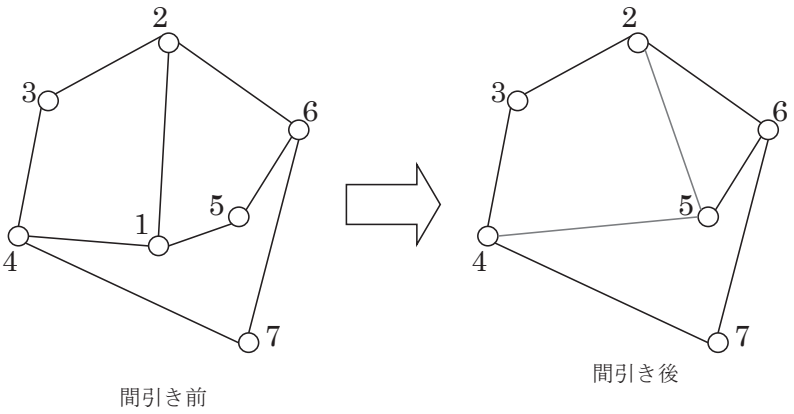


図2.2　点１の間引き

Douglas-Peucker のアルゴリズムを用いた間引きプログラム実行の結果である境界線を図3.1.3に示す。なお表示の際に少しでも境界線を見やすくするために隠岐の表示は省いてある。間引き前の松江市周辺の境界線を拡大したものが図3.1.4，グラフ理論に基づく間引き後の境界線が図3.1.5，Douglas-Peucker のアルゴリズムを用いた簡略化後の境界線が図3.1.6である。またそれぞれの方法での簡略化の精度，間引き後の点と辺の数，辺の長さと座標の最小値，最大値を

図3.1.1　間引き前の島根県行政区画の境界線

図3.1.2　グラフ理論に基づく間引き後の島根県行政区画の境界線

図3.1.3　Douglas-Peucker のアルゴリズムを用いた簡略化後の島根県行政区画の境界線

図3.1.4　間引き前の松江市周辺の境界線

表3.1に示す。

簡略化の精度は2.3節の短絡除去を行なう辺の条件２に設定する短絡除去する辺の長さの閾値であり，緯度経度に基づいて計算する。この値が大きくなるほど間引く描画点の数は増えてゆく。隣接リスト表現を用いた方法の簡略化の精度0.001506は51,000番目に短い辺の長さであるが，簡略化の精度が50,000番目に短い辺の長さである0.001181以下では間引きによって境界線がほとんど変化しなかったため境界線の変化がひと目でわかる0.001506を選択した。一方，Douglas-Peucker のアルゴリズムでは，多角形の近似曲線を引く際，近似曲線から一定距離内に元の多角形を構成する点が全ておさまった場合にアルゴリズムが終了し，最終的な近似曲線が決定される。Douglas-Peucker を用いた方法の簡略化の精度は近似曲線からの距離を表わし，その値は隣接リスト表現を用いた方法の点と辺

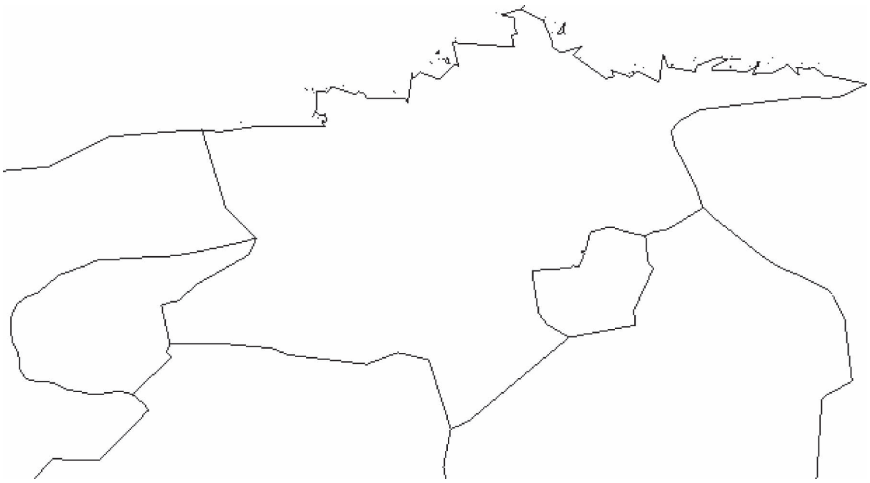


図3.1.5　グラフ理論に基づく間引き後の松江市周辺の境界線

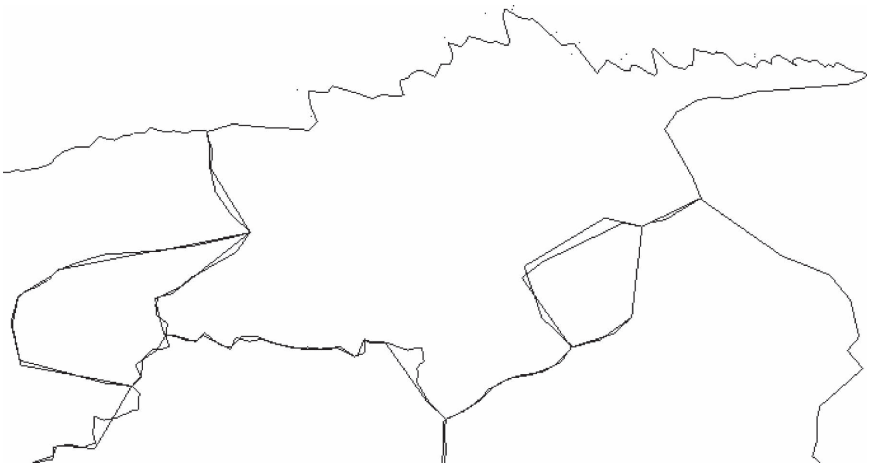


図3.1.6　Douglas-Peucker のアルゴリズムを用いた簡略化後の松江市周辺の境界線

表3.1　アルゴリズムごとの各数値

アルゴリズム	簡略化の精度	点の数	辺の数	辺の長さの最小値	辺の長さの最大値	左上の点	右下の点
間引き前		51,819	51,835	0.000003	0.026412	(37.2478, 131.6679)	(34.3024, 133.3870)
隣接リスト表現を用いた方法	0.001506	1,856	1,872	0.000029	0.118640	(37.2470, 131.6756)	(34.3078, 133.3866)
Douglas-Peucker を用いた方法	0.050	2,200	1,997	0.000017	0.064998	(37.2471, 131.6692)	(34.3055, 133.3864)

※点の座標は（緯度, 経度）となっている

の数になるべく近くなる0.050を選んだ。

図3.1.1 ～ 図3.1.6 から，従来の手法である Douglas-Peucker のアルゴリズムを用いた簡略化では面の間に隙間や交差が生じているのに対し，境界線データをグラフとして表わし間引きを行なうことで隙間や交差を生じさせることな

く描画点数を減少させられることがわかる。

3.2　隣接行列表現を用いた場合の実行結果

隣接行列表現を用いて境界線データをグラフとして表わそうとしたところ，51,819×51,819行列を記憶するメモリ領域を確保できなかった（今回用いたC言語では，boolean型はint型で代用されるため，約10.8 GBのメモリが消費される）。しかし，よりサイズの小さい河川流域境界線データに対して，隣接リスト表現と隣接行列表現の２通りの方法を用いて境界線データをグラフとして表わし，間引きを行なったところ，実行時間に違いは見られたものの実行後のデータは同様のものが得られた。以上から隣接行列表現を用いた場合は辺の数に関わらず|V|×|V|行列を記憶するメモリ領域が必要となるものの，隣接リスト表現を用いた場合と同様に面の間に隙間や交差を生じさせることなく描画点を間引くことができることを確認した。このメモリの問題を解決するためには，有限差分法，有限体積法，有限要素法などで一般的に用いられているスパース行列技法の適用などを検討する必要があろう。

以上から複体方式で表わされた地理情報をグラフとして扱い，描画点の間引きを行なうことで従来の簡略化が持つ問題点を克服することができるということがわかった。しかし，これまでに用いたグラフを表現するデータ構造では境界線データの面の情報を記憶させることができないので，境界線データが持っていた面の情報が失われてしまう。そこで，本研究では次に地図グラフ表現の標準的データ構造[1)]によって境界線データを表現し描画点の間引きを行なった。

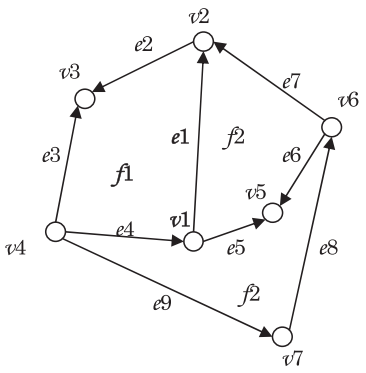
4.　地図グラフ表現の標準的データ構造を用いて面情報を保持した間引き

前章で用いたデータ構造では面情報を保持できないため，変換後のデータにおいて面単位での処理や，元の境界線データへの再変換を行なうことができない。以下では面情報を保持するデータ構造として，地図グラフ表現の標準的データ構造[1)]を用いて描画点の間引きを行なう。

4.1　地図グラフ表現の標準的データ構造

地図グラフとは平面上に辺を交差させることなく描画されたグラフのことで，これを表現する標準的データ構造とは，辺，点，面ごとに以下の情報を持たせたものである[1)]。

(a) 各辺 e ごとに
　（ｉ）e の始点，（ii）e の終点，（iii）e の左面，（iv）e の右面
(b) 各点 v ごとに
　（ｉ）v に接続する辺の集合（辺の向きを



・辺 $e1$ に持たせる情報
　始点　$v1$
　終点　$v1$
　右面　$f2$
　左面　$f1$
・点 $v1$ に持たせる情報
　$v1$ に接続する辺の集合
　$\{e1, -e4, e5\}$
・面 $f1$ に持たせる情報
　$f1$ の周囲の辺の集合
　$\{e1, e2, -e3, e4\}$

図4.1　地図グラフの標準的データ構造の例

判別するために v を終点としている要素にはマイナス符号を付ける）

(c) 各面 f ごとに

（i）f の周囲3の辺の集合（辺の向きを判別するために f の境界を反時計回りに巡るとき，負方向に出合う要素にはマイナス符号を付ける）

地図グラフ表現の標準的データ構造を用いて図4.1の地図グラフの辺 $e1$，点 $v1$，面 $f1$ をあらわしたものを例として以下に示す。

4.1.1 境界線データから地図グラフ表現の標準的データ構造への変換

境界線データ[5] から地図グラフ表現の標準的データ構造への変換を行なうプログラムでは境界線データに現れる順に辺，点，面に ID 番号を付与し，その ID 番号に基づき4.1節の (a)～(c) の各データを作成する。なおプログラム上では (b) の（i）と (c) の（i）は反時計回りの順に記憶した双方向循環リストによって表現する（図4.1.1.1）。さらにこれらに加えて，間引き処理において使用する各点の次数と各面を構成する辺の数，境界線の描画の際に使用する各点の座標，各面の市町村コードとセグメントも記憶する。

変換後のデータは，各辺ごとに図4.1.1.2のように (a) の（i）～(iv) の各点と各面の ID 番号と，各点と各面が保持する辺のリストの前後の辺の ID 番号を記述し，また各点ごとに図4.1.1.3のように (b) の（i）を表わすリストの先頭の辺の ID 番号と，座標，次数を記述し，また各面ごとに図4.1.1.4のように (c) の（i）を表わすリストの先頭の辺の ID 番号と，周囲の辺の数，市町村コード，セグメントを記述することでファイルに保存する。

間引きを行なうプログラムは以上の構造を持つ3つのファイルを入力データとして読み込み以下の操作で点を間引いてゆく。

ID 番号が若い順に各点ごとに手順1～3を行なう

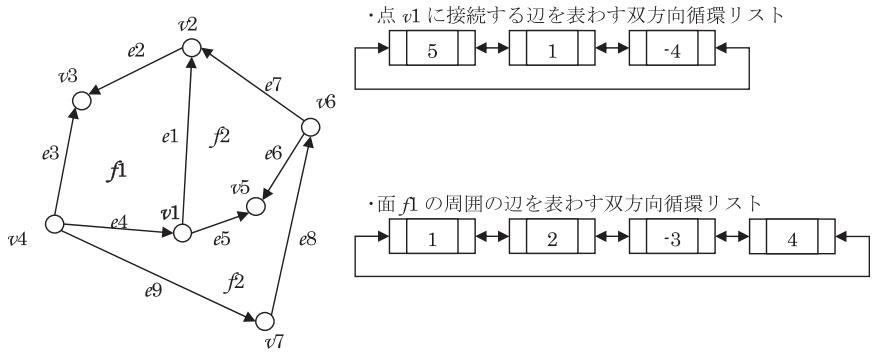


図4.1.1.1　$v1$に接続する辺の集合と面 $f1$の周囲の辺の集合を反時計回りに双方向循環リストで表現した例

2	1	1	0	-2	-2	1909	1909	1909	2		
3	2	1	0	-3	-3	1	1	1	3		
					⋮						
3014	3013	2	121	-3015	-3015	3013	3013	3013	3015	-3015	-3013
3015	3014	2	21	-3016	-8811	3014	3014	3014	3016	-8811	-3014
始点	終点	左面	右面	始点が持つリストで直後の辺	始点が持つリストで直前の辺	終点が持つリストで直後の辺	終点が持つリストで直前の辺	左面が持つリストで直後の辺	左面が持つリストで直前の辺	右面が持つリストで直後の辺	右面が持つリストで直前の辺

図4.1.1.2　辺を表わすファイルの内容

-1	35.074251	133.136321	2
-2	35.074343	133.136346	2
	⋮		
51833	37.238929	131.867445	2
51834	37.238940	131.867344	2

隣接リストの先頭の辺　緯度　経度　次数

図4.1.1.3　点を表わすファイルの内容

32343	1	1909	1909
32206	1	-653	1739
	⋮		
32528	61	51781	43
32528	62	51835	54

市町村コード　セグメント　周囲の辺を表わすリストの先頭の辺　周囲の辺の数

図4.1.1.4　面を表わすファイルの内容

手順１．短絡除去することで辺同士の交差や面の数の変化が生じず，なおかつ長さが簡略化の精度未満の辺が処理中の点に接続しているならば手順２～３を行ない，さもなければ次の点の処理を手順１から行なう

手順２．手順１の条件に合う辺のうち最も短い辺を，その辺を境界線とする面の周囲の辺の集合から取り除く

手順３．手順１の条件に合う辺のうち最も短い辺の短絡除去を行なうことで，処理中の点を取り除き，その点に接続している辺を短絡除去する辺のもう片方の点に接続し直す

以上の手順で間引きを行ない，その結果として入力データと同じ構造のファイルを出力する。間引きによる辺の置き換えで辺同士の交差が生じるか否かの判定は，短絡除去する辺が構成する面が飛び地のように周囲に面が無い場合は置き換える辺ごとにその他全ての辺について交差判定を行ない，周囲に面がある場合は置き換える各辺に周囲の面を構成する辺と飛び地を構成する辺について交差判定を行なった。

4.2　地図グラフに対する間引き実行結果

島根県行政区画の境界線データを，地図グラフ表現の標準的データ構造を用いて地図グラフとして表現した上で間引きを行なった結果，点の数は51,819から1,856，辺の数は51,835から1,872となり，面の数は282のまま変化せずという結果になった。隠岐を除く境界線は図4.2.1，松江市周辺の境界線は図4.2.2のように変化した。

以上から地図グラフ表現の標準的データ構造を用いた間引きでも，３節で示した実行結果と同様に面の間に隙間や交差を生じさせずに間引きを行なうことができるということがわかる。また３節では記さなかったが，Douglas-Peucker のアルゴリズムを用いた簡略化後の島根県行政区画の境界線データの面（構成する点の数が３つ以上である区画）の数は21となっており，従来の手法では面の数が変化しているのに対し，今回行なったグラフ理論に基づく間引きでは面の数が変化していないということもわかる。

図4.2.1　間引き後の島根県行政区画の境界線

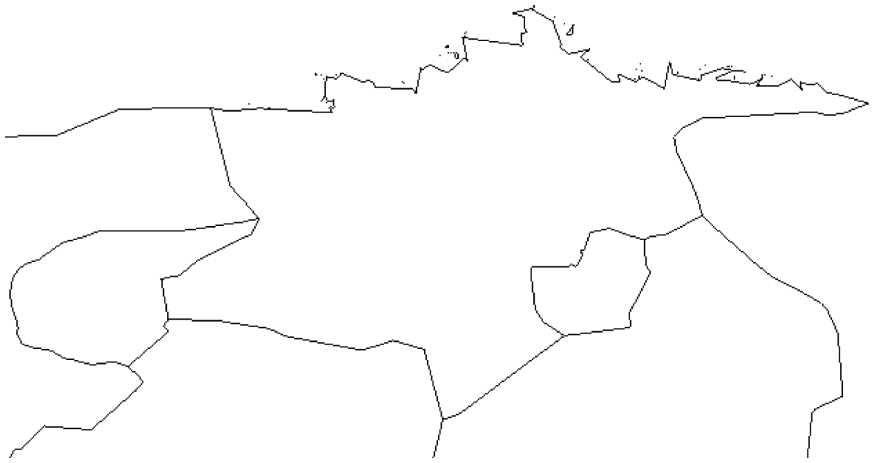


図4.2.2　間引き後の松江市周辺の境界線

4.3　アルゴリズムの改良

これまでに示したように境界線データをグラフとして表わし描画点を間引くことで面の間に隙間や交差を生じさせることなく描画点数を減少させることができるが，間引く点の選択条件が間引きによって隙間や交差，面の数の変化が生じないということのみなので間引きの結果，面が大きく変形してしまう。そこで，間引きによる面の変形を抑えるために，間引きアルゴリズムの改良を行なった。

4.3.1　次数情報を追加したアルゴリズム

次数が大きい点は多くの面を構成しているため，間引きを行なうと次数が大きい点ほど多くの面の変形を招いてしまう。そこで次数が3以上の点を間引かないようにアルゴリズムを変更した。その結果である隠岐を除く境界線を図4.3.1.1に示す。また，変更前のアルゴリズムでの結果との差異が特に見られた飯南町と出雲市と太田市付近の境界線の比較を図4.3.1.2に示す。なおアルゴリズム変更前後で間引き後の点と辺

図4.3.1.1　次数を考慮した間引き後の島根県行政区画の境界線

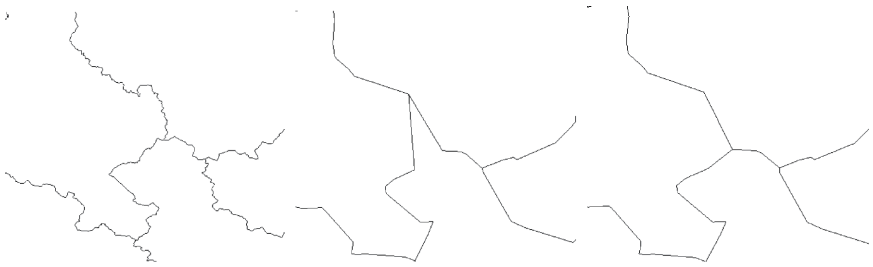


図4.3.1.2　飯南町と出雲市と太田市周辺の境界線の比較

の数が変化しないように簡略化の精度を調整し，0.001514とした。

図4.3.1.1と図4.3.1.2から次数が３以上の点を間引かずに残すことで，間引きによる変形が抑えられていることがわかる。

4.3.2　次数情報と中間点を考慮したアルゴリズム

ここでは前節の次数の考慮に加え，図4.3.2.1のように短絡除去の際に，除去する辺の両端点をそれらの中間の座標に新たな点を設け，同一視するようにアルゴリズムを変更した。その結果である隠岐を除く境界線は図4.3.2.2，松江市周辺の境界線は図4.3.2.3のようになった。またこれまでの各アルゴリズムごとの簡略化の精度，間引き後の点，辺，面の数，辺の長さと座標の最小値，最大値は表4.3.2のようになった。ここでもアルゴリズム変更前後で点と辺の数が変化しないように簡略化の精度を調整した。

図4.3.2.2と図4.3.2.3より，短絡除去の際に中間点を設けることによって境界線は滑らかにはなるものの元の形状が完全に保たれるわけではないと考えられる。

4.3.3　計算量

これまでに用いた各アルゴリズムの漸近的計算量は表4.3.3.1のようになる。また各アルゴリズムでの実行時間を表4.3.3.2に示す。

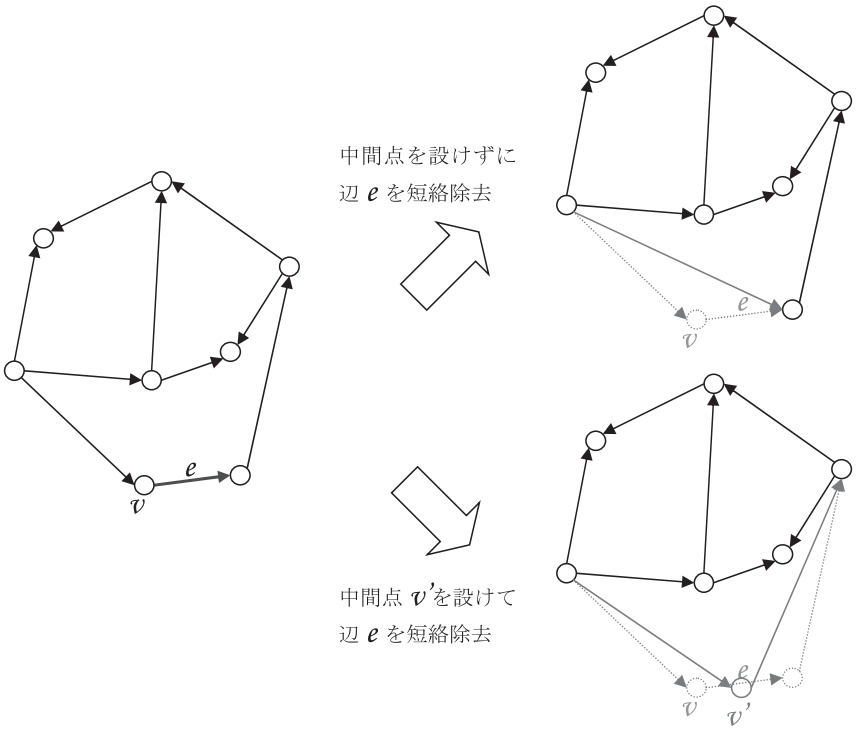


図4.3.2.1 点 v を間引く際に中間点 v' を設けて辺 e を短絡除去する例

図4.3.2.2 次数を考慮し中間点を設ける間引き後の島根県行政区画の境界線

領域計算量について，Douglas-Peucker を用いた方法は各面についてその周囲の点を読み込んで処理を行なうため各面ごとの周囲の辺の数に比例し，S(C) となる。その他のアルゴリズムでは地図グラフ全体を読み込んで処理をするため領域計算量は点，辺，面の数に比例し，S（|V|＋|E|＋|F|）となる。

時間計算量について，Douglas-Peucker を用

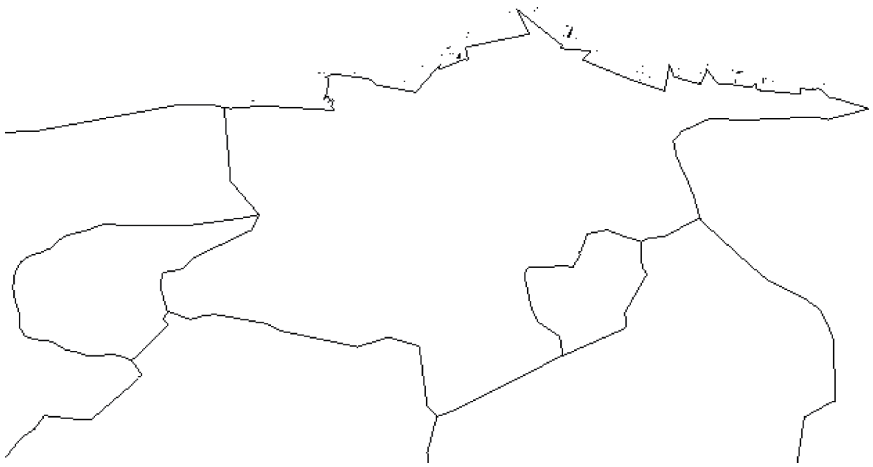


図4.3.2.3　次数を考慮し中間点を設ける間引き後の松江市周辺の境界線

表4.3.2　アルゴリズムごとの各数値

アルゴリズム	簡略化の精度	点の数	辺の数	面の数	辺の長さの最小値	辺の長さの最大値	左上の点	右下の点
間引き前		51,819	51,835	282	0.000003	0.026412	(37.2478, 131.6678)	(34.3024, 133.3870)
次数考慮無し	0.001506	1,856	1,872	282	0.000029	0.118640	(37.2470, 131.6756)	(34.3078, 133.3866)
次数考慮有り	0.001514	1,856	1,872	282	0.000029	0.118640	(37.2470, 131.6756)	(34.3078, 133.3866)
次数考慮有り 中間点有り	0.002022	1,856	1,872	282	0.000043	0.115350	(37.2471, 131.6819)	(34.3082, 133.3870)
Douglas-Peucker を用いた方法	0.050	2,200	1,997	21	0.000017	0.064998	(37.2471, 131.6692)	(34.3055, 133.3864)

表4.3.3.1　各アルゴリズムの漸近的計算量

アルゴリズム	領域計算量	時間計算量
次数考慮無し	S(|V|＋|E|＋|F|)	O(|F|･C＋|V|･|E|･D)
次数考慮有り	S(|V|＋|E|＋|F|)	O(|F|･C＋|V|･|E|･D)
次数考慮有り，中間点有り	S(|V|＋|E|＋|F|)	O(|F|･C＋|V|･|E|･D)
Douglas-Peucker を用いた方法	S(C)	O(|F|･C^3)

|V|, |E|, |F|…点，辺，面の集合の濃度
D…次数の平均，C…面の周囲の辺の数の平均

いた方法は，各面ごとに，特徴点を計算し近似曲線を引きつつ，近似曲線から処理中の面の各点との距離を求める部分が３重ループとなる。そのため時間計算量は面とその周囲の辺の数に比例し，O(|F|･C^3) となる。その他のアルゴリズムでは，まず交差判定に用いる飛び地の周囲の辺の特定のために各面の周囲の点の次数を調べるため，計算量が面とその周囲の辺の数に比

表4.3.3.2 各アルゴリズムでの相対実行時間

アルゴリズム	相対実行時間
次数考慮無し	27.2
次数考慮有り	25.8
次数考慮有り，中間点有り	103.1
Douglas-Peucker を用いた方法	1.0

```
for(i=0; i<nof; i++)       /*  全ての面について飛び地か否かを判定する  */
{
   lp0 = f[i].edgelist;    /* lp0->num:f[i]を構成する辺の番号  */
   while(1)
   {
      if(v[e[abs(lp0->num)-1].tail-1].deg>2 || v[e[abs(lp0->num)-1].head-1].deg>2) break;
      /*  次数が 2 より大の点があれば他の面と接していると判断  */

      lp0 = lp0->next;
      if(lp0==f[i].edgelist)              /*  周囲に次数が 2 以上の点が無いならば飛び地と判断  */
      {
         while(1)
         {
            insert_ascend(island,abs(lp0->num));              /*  周囲の辺を island に挿入  */

            lp0 = lp0->next;
            if(lp0==f[i].edgelist) break;
         }

         break;
      }
   }
}
```

図4.3.3.1 間引きプログラム（飛び地の周囲の辺の特定部分）

例する（図4.3.3.1）。そして間引き処理において，全ての点について，その点に接続する辺が間引きに伴なう置き換えによって他の辺と交差するかの判定部分が，点とその次数と辺の数に比例する（図4.3.3.2）。したがって時間計算量は O(|F|·C＋|V|·|E|·D) となる。ただし，中間点を設けるアルゴリズムの漸近的計算量は中間点を設けない場合と同じではあるものの，間引きによって置き換えられる辺が中間点を設けない場合の約 2 倍に増えるため交差判定に要する時間もほぼ 2 倍となり，実行時間に差が生じる。

以上のように，本研究の間引きアルゴリズムではグラフ全体に対して処理を行ない，さらに間引きの度に辺同士の交差判定をするため，各面に対して処理を行ない，交差判定をしない Douglas-Peucker のアルゴリズムを用いた間引きに比べて領域計算量と時間計算量の両方において劣ることは不可避といえる。

5. 考察と今後の課題

複体方式で表わされた境界線データをグラフとして表わし描画点を間引くことで従来の簡略化で見られた問題点を克服することができた。しかし，今回行なった間引き後の境界線は多少変形する。従って，今後の課題は，地図グラフの面を次数が 3 以上である点で区切った折れ線に分割し，各折れ線ごとに Douglas-Peucker のアルゴリズムを用いて近似曲線を引くなど，変形が小さくなるようにさらに間引きアルゴリズムを改良すること，処理に必要な計算機資源を低減させること，などを検討する必要があろう。

```
for(i=0; i<nov; i++)      /* i:間引き処理中の点番号 */
{
                                                    ⋮
  lp1 = v[i].adjlist;     /* lp1->num:v[i]に接続している(間引き処理によって置き換える)辺の番号 */
  while(1)
  {
    if(lp1->num!=lp0->num)        /* 除去する辺は判定しない */
    {
      if(lp1->num>0) v0 = &v[e[lp1->num-1].head-1];    /* v0:新たな辺の replace でない方の端点 */
      else v0 = &v[e[-lp1->num-1].tail-1];

      lp2 = temp2;     /* temp2:面を構成する辺の番号を記憶したリスト */
      /* 除去する辺が構成する面とその周りの面を構成する辺について交差判定を行なう */
      while(1)
      {
        if(e[lp2->num-1].tail!=v[i].num && e[lp2->num-1].head!=v[i].num)
        /* 間引きによって変化しない辺について判定を行なう */
        {
          if(intersection(replace,v0,&v[e[lp2->num-1].tail-1],&v[e[lp2->num-1].head-1])==1)
          {
            isc_flag = 1;
            break;
          }
        }

        lp2 = lp2->next;
        if(lp2==temp2) break;
      }
      if(isc_flag==1) break;

    /* 飛び地を構成する辺との交差判定 */
    lp2 = island;         /* lp2->num:飛び地を構成する辺の番号 */
    while(1)
    {
      if(e[lp2->num-1].tail!=v[i].num && e[lp2->num-1].head!=v[i].num)
      /* 間引き処理によって変化しない辺との交差判定を行なう */
      {
        if(intersection(replace,v0,&v[e[lp2->num-1].tail-1],&v[e[lp2->num-1].head-1])==1)
        {
          isc_flag = 1;
          break;
        }
      }

      lp2 = lp2->next;
      if(lp2==island) break;
    }
    if(isc_flag==1) break;
  }

  lp1 = lp1->next;
  if(lp1==v[i].adjlist) break;
}
```

図4.3.3.2　間引きプログラム（交差判定部分）

文　　献

1)　伊理正夫，腰塚武志（1993），計算幾何学と地理情報処理，第２版，共立出版

2)　小沢哲也（1997），平面図形の位相幾何，培風館

3)　David H. Douglas and Thomas K. Peucker（1973），Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, The Canadian Cartographer 10, pp. 112−122

4)　SAS/GRAPH(R) 9.4 Reference https://support.sas.com/documentation/cdl/en/graphref/67881/HTML/default/viewer.htm#titlepage.htm

5)　国土地理院 基盤地図情報ダウンロードサービス http://fgd.gsi.go.jp/download/

6)　R. セジウィック（原著），野下浩平，星　　守，佐藤　創，田口　東（邦訳）（1994），アルゴリズム C++，近代科学社