# 差分チェインコードの統計的性質を利用した輪郭線画像の効率的な圧縮符号化

木村　彰男† (正員)　　遠藤　卓弥†*
渡辺　孝志† (正員)

An Efficient Coding Scheme for a Bi-level Boundary Image Using Statistical Characteristics of Subtracted Chain Codes

Akio KIMURA†, *Member*, Takuya ENDO†*, *Nonmember*, and Takashi WATANABE†, *Member*

† 岩手大学工学部電気電子・情報システム工学科，盛岡市
Department of Electorical Engineering and Computer Science, Faculty of Engineering, Iwate University, Morioka-shi, 020-8550 Japan
* 現在，(株) イーアールアイ

**あらまし**　本論文では，例えば地形図等高線画像のような2値の輪郭線画像に対して効果的に機能する，画像の圧縮符号化方式を提案する．従来，画像の圧縮を目的とした符号化方法はいくつか提案されているが，本論文で提案する手法は，輪郭線から得られる差分チェインコードに対して適切な符号置換を施し，その結果にハフマン符号化を適用して圧縮を図るという，ごくシンプルな方式であることが特長である．評価実験では，提案手法によって様々なタイプの2値輪郭線画像が効率的に符号化され，従来手法に比べてより圧縮効率が高められることを示す．

**キーワード**　2値輪郭線画像，ロスレス圧縮，チェインコード，ハフマン符号化

## 1. まえがき

ディジタル画像の保存や転送を行う上で，画像データを効率良く圧縮する手法は重要である．例えば地形図処理システム [1], [2] などでは，大量の等高線画像 (2値輪郭線画像) を効率良く蓄積保存しておき，必要に応じて高速に処理することが求められるため，性能の良い圧縮符号化技術を確立することが重要である．特に，圧縮された等高線データは完全に復元できなければならないので，ロスレスタイプのものが不可欠である．

2値画像をロスのない形で圧縮符号化する方法としては，国際標準である JBIG2 [3] が広く利用されている．JBIG2 は，算術符号化やパターンマッチング技術を駆使した圧縮率の高い符号化方式として知られているが，筆者らが対象としているような輪郭線画像では必ずしも最適な圧縮結果が得られるわけではない．詳細については後の章で述べるが，このような場合にはチェインコード [4] を利用した方がよいこともある．チェインコードに基づいた圧縮符号化方式としては，従来からいくつか提案されており，例えば生成されたチェインコードの統計的分布から内容木を構築し，その内容木を用いた表現に変換してデータ圧縮を図る手法 [5] がある．しかしながらこの手法では，内容木の生成やそれに基づいた符号化のために過去のチェインを複数個参照しながら処理を進める必要があり，更には事前に内容木を最適化しておくために再帰計算を行う必要があるなど，原理がやや複雑で計算量も多い．このため，大量の画像を一括して処理するのには向いていないように思われる．一方で，生成されたチェインの進行方向に対する角度変化に応じて新たな符号を割り当て，更にその符号にハフマン符号化を適用してデータ圧縮する手法 [6] が提案されている．この手法は，前者の手法に比べれば原理が簡単で計算量もそれほど多くはなく，効率的な手法であるといえるが，文献 [7] の報告によると，処理対象画像のチェイン数が増えたり個々のチェインが長くなったりすると圧縮率が低下してしまう (JBIG2 などの一般的な2値画像圧縮法に劣る場合がある)，といった問題があることも示されている．したがって，輪郭線画像の圧縮符号化問題に対しては，よりシンプルで圧縮効率の高い手法を新たに検討する余地が残されているといえる．

そこで本論文では，文献 [6] の手法を基本として，更に効率的な圧縮符号を生成するための改良アルゴリズムを提案する．提案する手法は，輪郭線画像から得られた差分チェインコードに対して適切な符号置換を施して，その結果に改めてハフマン符号化を適用するという，極めてシンプルなものである．輪郭線画像の差分チェインコードを注意深く観察すると，ある特定のパターンが多く発生していることが分かるので，提案手法ではその統計的な性質を積極的に利用することで効率的な圧縮符号化を実現している．以下では，具体的なアルゴリズムを述べるとともに，評価実験によってその有効性を検証する．

## 2. 輪郭線画像の改良符号化方式

### 2.1 差分チェインコード

チェインコードは，2値画像における輪郭線パターンをより簡潔に表現するための方法であり，Freeman [4] が提唱した8連結方式のものが代表的である．チェインコードでは，輪郭線の連結成分を8連結で追跡し，注目画素の次に追跡が行われる画素への方向を8種類の数値 (符号) で表現する (図1，図2参照)．このため，輪郭線パターンを画像の形式で画素単位に保存

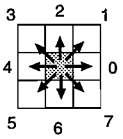


図 1　8 連結チェインコード
Fig. 1　8-connected chain code.

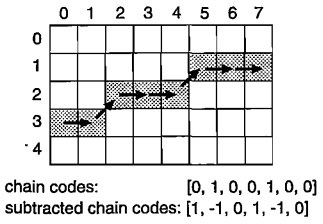


図 2　チェインコードと差分チェインコード
Fig. 2　Chain codes and subtracted chain codes.

しておくよりも効率的な記述が可能とされている．例えば図 2 のような輪郭線パターンは，始点画素位置 (0,3)，[0,1,0,0,1,0,0] のようにチェインコード表現できる．

本論文では，この Freeman のチェインコードを直接使うのではなく，差分チェインコードに変換してから利用する．今，$i$ 番目のチェインコードを $c_i$ $(0 \leq c_i \leq 7)$ と書くことにすると，差分チェインコード $x_i$ は，隣接するチェインコード $c_i$ と $c_{i-1}$ の差分 $k_i = c_i - c_{i-1}$ を用いて次のように書くことができる．

$$x_i = \begin{cases} k_i + 8 & (k_i < -3) \\ k_i - 8 & (k_i > 4) \\ k_i & (\text{その他}) \end{cases} \qquad (1)$$

こうすると，差分チェインコード $x_i$ のとり得る値は $\{-3, -2, -1, 0, 1, 2, 3\}$ となり，図 2 の輪郭線パターンの例では，差分チェインコードは [1,−1,0,1,−1,0] と表現される．一般的に，差分チェインコードでは，通常のチェインコードよりも出現符号に偏りができ，エントロピーが小さくなるといわれている．

### 2.2　差分チェインコードの符号置換

本論文では，主として地形図等高線画像を処理対象ととらえているが，等高線画像に限らず，一般的な輪郭線画像では生成される差分チェインコードに [...,1,−1,...] や [...,−1,1,...] という符号が多く現れる傾向がある（図 3 参照）．また，ある一定の傾きをもったディジタル直線は，隣接画素に進む際の方向変化が規則的になりやすいので，結果的に，差分チェインコード中に似たような符号列パターンが繰り返して現れる傾向がある（図 4 参照）．

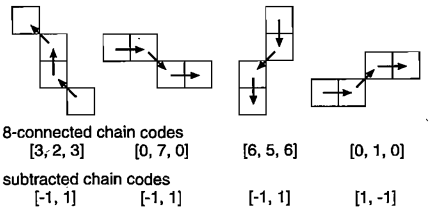


図 3　輪郭線によく現れるパターンの例
Fig. 3　Examples of boundary patterns.

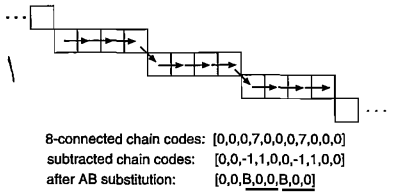


図 4　特定の符号列が繰り返される輪郭パターンの例
Fig. 4　An example of boundary patterns that the specific codes repeatedly appear.

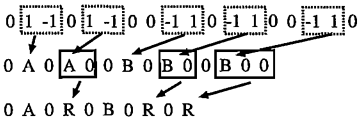


図 5　差分チェインコードの符号置換例
Fig. 5　An example of code substitutions for subtracted chain codes.

本論文では，これらの性質に特に注目し，以下に述べるような簡単な符号の置換えによって圧縮率の向上を図る．

#### 2.2.1　特定パターンの再符号化

例えば，図 3 に示したようないくつか輪郭線パターンは，チェインコードこそ異なるものの，差分チェインコードはいずれも [1,−1] あるいは [−1,1] である．そこで，差分チェインコード中で [1,−1] または [−1,1] と続けて現れる部分にそれぞれ新しく [A]，[B] という新しい符号を割り当てることにする．具体的な割当例を，図 5 の上段部に示す．

#### 2.2.2　繰返し符号列の再符号化

続いて，先の手順で [A]，[B] を含めて再符号化が行われた差分チェインコードに対し，もう一つ別の符号置換を施すことを考える．図 4 に示したような，ある一方向に直線的に長く延びる輪郭線の差分チェインコードには，“進行方向の変化を表す [0] 以外の符号一つ（図中では [B]）と，複数個の連続した [0] で構成される符号列パターン” が，繰り返し続けて現れる傾

向がある（図 4 の例では，[B,0,0] という符号列が 2 回繰り返して現れている）．この傾向は，特に等高線画像を対象とした場合に頻発するので，本手法では，[*,0,0,...,0,*,0,0,...,0]（*は [0] 以外の符号）といった，特定符号と連続した [0] からなる符号列が 2 回続けて現れた場合には，後者を新たな符号 [R] で置き換えることにする．図 4 の例でいえば，[0,0,B,0,0,B,0,0] には 2 回続けて [B,0,0] が現れているので，後の方を [R] で置き換えて [0,0,B,0,0,R] と表現する．

以下では，この符号 [R] で置換えを行うための詳細手順について示す．

[繰返し符号列の再符号化手順]

Step 1. 差分チェインコードを一つずつ順に調べていき，[0] 以外の符号が見つかるまでは調べた符号 [0] をそのまま出力していく．[0] 以外の符号が見つかったら，それを $S_1$ に保存する．

Step 2. 次の [0] 以外の符号が現れるまでの [0] の総数をカウントして $\alpha_1$ とし，その現れた [0] 以外の符号を $S_2$ に保存する．そして，符号 $[S_1]$ と $\alpha_1$ 個の [0] を出力する．

Step 3. 更に，次の [0] 以外の符号が現れるまでの [0] の総数をカウントして $\alpha_2$ とし，現れた [0] 以外の符号を $S_3$ に保存する．そして以下の処理を施す．

- $S_1 = S_2$ かつ $\alpha_1 \leq \alpha_2$ のときは，新符号 [R] と $(\alpha_2 - \alpha_1)$ 個の [0] を出力する．
- それ以外の場合には，符号 $[S_2]$ を出力し，更に $\alpha_2$ 個の [0] を出力する．

Step 4. $S_1 \leftarrow S_2$，$S_2 \leftarrow S_3$，$\alpha_1 \leftarrow \alpha_2$ とし，更に $\alpha_2 = 0$ として Step 3 を繰り返す．チェインコードの終わりにきたら処理を終了する．

図 5 の下段部に，この [R] による再符号化の例を示した．この図から分かるように，[R] は，直前のチェインコード数個分を一つの符号で表現するものであるが，画像によってはこの [R] がかなり多く出現するので，後の **2.3** に示すハフマン符号化によって更に効率良く符号化できる可能性がある．

なお，上記の手順で再符号化が行われた（[R] を含んだ）コードから，もとの（[R] を含まない）コードを復元するには，以下の手順に従えばよい．

[繰返し符号列の復号手順]

Step 1. 再符号化コードを一つずつ順に調べていき，[0] 以外の符号が現れるまで調べた符号（[0]）をそのまま復号結果として出力する．[0] 以外の符号が見つかったら，その符号を出力してから $S_1$ に保存する．

Step 2. 続けて再符号化コードを調べていき，次の [0] 以外の符号が現れるまで調べた符号（[0]）を復号結果として出力する．この際，この [0] の総数をカウントして $\alpha_1$ としておく．[0] 以外の符号が見つかったら，その符号を $S_2$ に保存し，以下の処理を施す．

- $S_2$ が [R] の場合，符号 $[S_1]$ を出力し，更に加えて $\alpha_1$ 個の [0] を出力する．
- $S_2$ が [R] 以外の場合，符号 $[S_2]$ を出力し，$S_1 \leftarrow S_2$ とする．

Step 3. 再符号化コードの終わりがくるまで Step 2 を繰り返す．

これらの符号 [R] による再符号化，符号 [R] の復号化手順は，対象コードをただ一度走査するだけで処理が終了するので，いずれも速く実行できることを付け加えておく．

### 2.3 ハフマン符号化

圧縮の最終段階として，ハフマン符号化 [8] を適用する．本手法では，1 枚の輪郭線画像に含まれるすべてのチェインに対して **2.2** で述べた二つの置換え処理をそれぞれ施した後，得られた（すべてのチェインの）再符号化コードを対象としてハフマン符号を生成することにする．そして，このハフマン符号で表現し直したものをバイナリーデータに変換して，最終的な圧縮符号を得る．

### 2.4 提案アルゴリズムの手順詳細

提案アルゴリズムをまとめると次のようになる．

[輪郭線画像の圧縮アルゴリズム]

Step 1. 輪郭線追跡などによって処理対象画像のチェインコードを抽出し，式 (1) に従って差分チェインコードを算出する．この際，各チェインの始点座標（BOC: Beginning Of Chain）も別途保持しておく．

Step 2. 算出されたそれぞれの差分チェインコードに対し，先に述べた **2.2.1** と **2.2.2** の方法に従って新しい符号 [A]，[B]，[R] を割り当てて新たなコードを得る．

Step 3. 先のステップで置換え処理が行われたコードのすべてを対象として，それらの中に現れている各符号の出現頻度をそれぞれ調べる．そして，その頻度に基づいて各々のコードをハフマン符号化して圧縮コードを得る．

Step 4. ハフマン木，及び各チェインの BOC と圧縮済コードをバイナリーデータに変換した上でそれぞれファイル出力し，処理対象画像の圧縮結果とする．

なお，圧縮されたデータから元の輪郭線画像を復元

表 1 テスト画像の諸元
Table 1 Specifications of tested images.

| | 画像の大きさ | チェイン数 | PBM (byte) |
|---|---|---|---|
| 画像 1 | 1024×648 | 139 | 83,002 |
| 画像 2 | 1055×622 | 80 | 82,148 |
| 画像 3 | 1068×625 | 223 | 83,794 |
| 画像 4 | 1010×868 | 67 | 110,292 |
| 画像 5 | 997×746 | 1 | 93,303 |
| 画像 6 | 1000×1000 | 1 | 125,059 |
| 画像 7 | 256×256 | 1 | 8,249 |
| 画像 8 | 256×256 | 2 | 8,249 |
| 画像 9 | 700×700 | 135 | 61,657 |

するには以下の手順に従えばよい．

[圧縮画像の復元アルゴリズム]

Step 1. バイナリーデータからハフマン木，及び各チェインの BOC と圧縮済コードを読み出す．

Step 2. 読み出したそれぞれの圧縮コードに対し，**2.2.2** で示した復号手順に従って再符号化コードを復元し，続いて符号 [A] を [1,−1]，符号 [B] を [−1,1] と置き換えてもとの差分チェインコードを復元する．

Step 3. 復元されたそれぞれの差分チェインコードからもとのチェインコードを求め，先に読み出しておいた BOC の位置から各々チェインをたどることでオリジナルの輪郭線画像を復元する．

## 3. 評価実験

### 3.1 ファイルサイズによる圧縮効果の比較

本論文で提案した手法の圧縮効果を検証するために，数多くの 2 値輪郭線画像に対して実験を試みた．紙面の都合上，ここでは図 6 に示した 9 種類の画像に対する結果について述べる．各画像の大きさとチェイン数，及び PBM（Portable BitMap）形式によるファイルサイズは表 1 にまとめてある．ここで，PBM 形式は，ヘッダ部を除けば 1 画素 1 ビットで保存されるので，ファイルサイズの数値がほぼ画像の大きさを示す指標になっていることに注意されたい．

実験では，以下の (1)〜(3) に示す 3 種類の手法を用いて各画像をそれぞれ圧縮符号化し，得られたファイルサイズをバイト単位で比較した．

（1） **2.2** で述べた二つの符号置換処理を行わず，差分チェインコードに直接ハフマン符号化を適用して圧縮符号化する方法（文献 [6] の方法(注1)）

（2） 差分チェインコードに **2.2.1** の置換処理（符号 [A][B] 置換）だけを施してハフマン符号化する方法

（3） 差分チェインコードに **2.2.1** と **2.2.2** の両方の置換処理（符号 [A][B][R] 置換）を施してからハフマン符号化する方法

比較結果を表 2 に示す．表中では，上記 (2) の方法を提案手法 1，上記 (3) の方法を提案手法 2 として表している．また，表中には三つの手法の最終的な平均符号長（bit）もそれぞれ示しており，更に，提案手法 1，2 のファイルサイズ欄には，括弧書きで文献 [6] の手法のファイルサイズに対する割合を併記した．

まず，提案手法 1 の結果から見ると，対象画像の大小，チェインの多少にかかわらず，すべての画像に対して文献 [6] の手法より高い圧縮性能を示していることが確認できる．特に，画像 1〜4 については，ファイルサイズを 75 %程度にまで圧縮することができており，等高線画像に対して **2.2.1** の [A][B] 置換がうまく機能したことが分かる．

更に提案手法 2 では，一部を除いてほぼ全体的に提案手法 1 よりも高い圧縮性能を示していることが見てとれる．特に画像 1〜3 に対しては，ファイルサイズが（文献 [6] の手法の）68 %程度にまで圧縮されており，その性能の高さを示すことができた．これは，これらの等高線画像中に符号 [R] で置き換えられる輪郭パターンが多く存在したことでハフマン符号化が効果的に機能した結果といえる．しかしながら，逆に画像 5〜7 のように，符号 [R] に置き換えられるパターンがあまり多くない場合には，提案手法 1 に比べてファイルサイズが若干増えてしまうこともあった．ただしその差はわずかであり，総合的に判断すれば提案手法 2 の方が有用性が高いものと考えている．

参考までに，各画像を 2 値画像圧縮の国際標準である JBIG2 を利用して圧縮した場合のファイルサイズも表中に示しておいた．これをみると，確かに JBIG2 はいかなる種類の 2 値画像に対しても安定に圧縮することが可能で，画像 4,9 などでは提案手法 2 よりも高い性能を示していることが分かる．しかしながら，画像 5〜8 のように，画像に含まれるチェイン数が極端に少ない場合には，先に示した 3 種類の（チェインコードを利用する）圧縮法の方が良い結果となっており，JBIG2 は輪郭線画像に対しては必ずしも最適な圧縮方法ではないことが改めて確認された．

### 3.2 検　　討

評価実験の結果から，本論文で提案した手法の妥当性と有効性は一応示されたものと考えるが，ここでは

(注1)：本論文で取り扱っている差分チェインコードは，チェインの進行方向変化を表す符号としてもとらえることができるので，この方法は実質的に文献 [6] の方法と等価である．

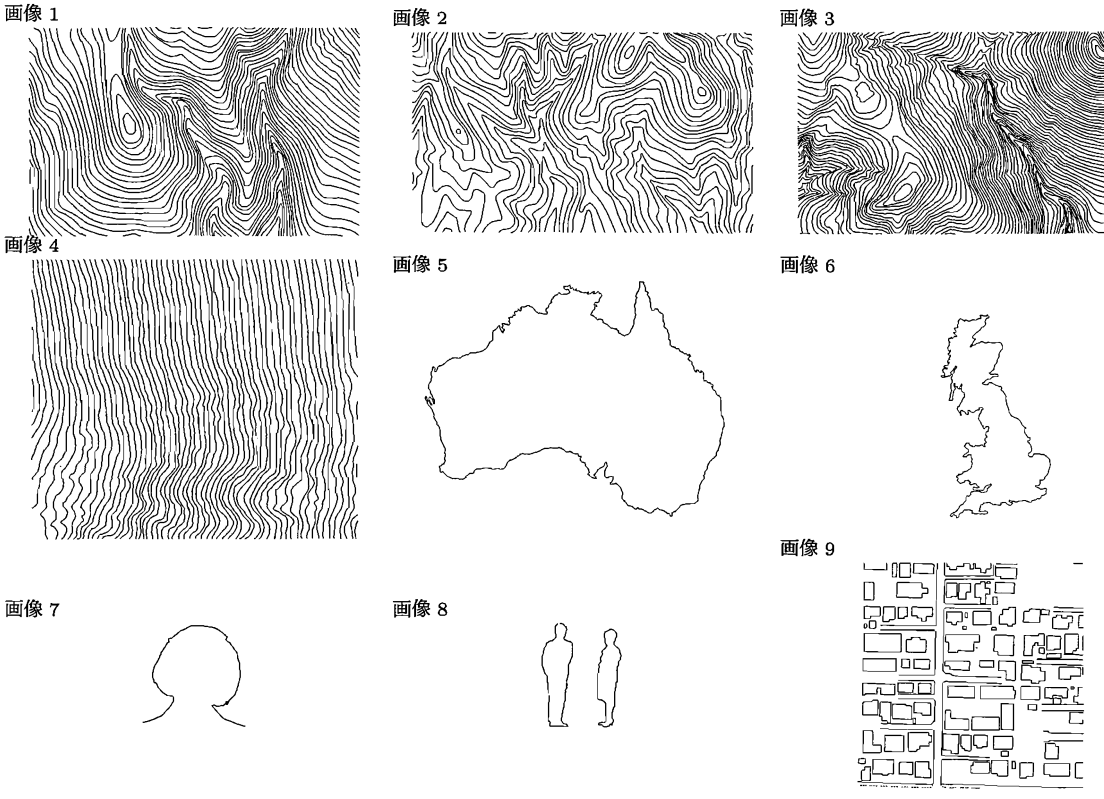


図 6 テスト画像
Fig. 6 Tested images.

表 2 ファイルサイズ (bytes) の比較結果
Table 2 Comparison results of file sizes (bytes).

| | PBM | 置換なし (文献 [6] に相当) | | 提案手法 1 ([A][B] 置換のみ) | | 提案手法 2 ([A][B][R] 置換) | | JBIG2 |
|---|---|---|---|---|---|---|---|---|
| | | サイズ | 平均符号長 | サイズ | 平均符号長 | サイズ | 平均符号長 | |
| 画像 1 | 83,002 | 10,071 | 1.70 | 7,260 (72.1 %) | 1.20 | 6,469 (64.2 %) | 1.06 | 7,123 |
| 画像 2 | 82,148 | 8,185 | 1.77 | 5,914 (72.3 %) | 1.25 | 5,634 (68.8 %) | 1.19 | 6,143 |
| 画像 3 | 83,794 | 14,508 | 1.85 | 10,335 (71.2 %) | 1.28 | 9,910 (68.3 %) | 1.22 | 9,921 |
| 画像 4 | 110,292 | 10,109 | 1.58 | 7,593 (75.1 %) | 1.17 | 7,423 (73.4 %) | 1.15 | 5,880 |
| 画像 5 | 93,303 | 982 | 2.07 | 959 (97.7 %) | 2.01 | 998 (101.6 %) | 2.10 | 1,234 |
| 画像 6 | 125,059 | 916 | 1.73 | 865 (94.4 %) | 1.63 | 901 (98.4 %) | 1.70 | 1,112 |
| 画像 7 | 8,249 | 192 | 1.76 | 173 (90.1 %) | 1.51 | 174 (90.6 %) | 1.52 | 311 |
| 画像 8 | 8,249 | 240 | 1.43 | 234 (97.5 %) | 1.35 | 232 (96.7 %) | 1.33 | 309 |
| 画像 9 | 61,657 | 3,692 | 1.20 | 3,647 (98.9 %) | 1.18 | 3,583 (97.0 %) | 1.16 | 1,271 |

提案手法の特徴や課題について考察する.

提案手法は, 輪郭線の差分チェインコードに現れる特徴的なパターンを見出して, そこに新たな符号を割り当ててからハフマン符号化するという, 極めてシンプルな方式で構成されている. 新たな符号の割当によって, 符号の種類がいったん増えてしまうことになるが, 出現頻度には大きく偏りが生じるので, 最終的にはハフマン符号化でかなり圧縮することができる. また, 提案手法は, 冒頭で述べた内容木に基づく手法 [5] や JBIG2 [3] などに比べて圧倒的に計算量が少ないため, 高速に処理することが可能である. これより, 大量の画像を一括処理する場合などに非常に有用であると思われる.

**2.3** で述べたように, 提案手法では対象画像ごとに

独立してハフマン符号化を行っている（つまり，対象画像の数だけハフマン木が存在している）が，試しにこれを，画像1〜9すべてをハフマン木生成の対象として取り扱い，ただ一つのハフマン木に基づいて符号化する形でも実験を行ってみた．詳細な数値等はここでは省略するが，すべての画像に対してファイルサイズが増加してしまう結果となった．特に画像7，8については，提案手法2におけるファイルサイズが2倍近くにまで増加してしまった．これは，これらの画像の輪郭点数が他の画像に比べて極端に少なく，自らの統計結果がハフマン木の構成にほとんど反映されなかったことに加えて，ハフマン木生成時の対象画像が増えたことで各符号の出現回数にあまり偏りがなくなり，ハフマン符号化の効果がさほど得られなかったことが要因と思われる．このことから，ハフマン符号化は画像ごとに独立して適用するのが妥当であると考えられる．

提案手法の課題としては，実験結果でも述べたように，符号 [R] でうまく置き換えられないパターンが多く存在する輪郭線画像（例えば画像9）に対する圧縮効率を改善すること，があげられる．現状の符号化アルゴリズムでは，例えば [..., A, 0, 0, B, 0, 0, A, 0, 0, B, 0, 0,...] と並ぶパターンを効率良く再符号化することができない．これについては，手法のシンプルさを損なわない形で更なる符号置換え処理を追加する，あるいは部分的にランレングス符号化を適用する，といった改良が考えられるが，詳細は今後の検討課題としたい．また，圧縮符号化の方式は大きく異なるが，文献 [5] のような内容木に基づいた手法等との性能比較も必要であろう．こちらについては稿を改めて議論したい．

## 4. むすび

本論文では，地形図等高線画像などに代表される2値輪郭線画像に対して効果的に機能する，圧縮性能の高い符号化アルゴリズムを提案した．提案手法は，輪郭線から得られた差分チェインコードに適切な符号置換を施し，その結果にハフマン符号化を適用する，という極めてシンプルな方法であるにもかかわらず，多くの輪郭線画像において効率的な符号化が行われ，従来の手法よりも高い圧縮性能を示すことを評価実験によって確認した．今後は，更なる圧縮率向上のためのアルゴリズム改良が必要である．

**謝辞** 本研究の一部は科学研究費補助金基盤研究(C) No.20500771 の助成によった．

## 文献

[1] 渡辺孝志，山本善一，阿部英志，木村彰男，“拡張ボロノイ線図を用いた等高線画像の大局的復元処理，” GIS-理論と応用，vol.6, no.2, pp.23–31, 1998.

[2] 古舘守通，渡辺孝志，阿部英志，横山隆三，“数値標高モデルの生成に用いる補間手法の性能評価，” GIS-理論と応用，vol.8, no.1, pp.29–38, 2000.

[3] F. Ono, W. Rucklidge, R. Arps, and C. Constantinescu, “JBIG2-the ultimate bi-level image coding standard,” Proc. International Conference on Image Processing 2000, vol.1, pp.140–143, 2000.

[4] H. Freeman, “On the encoding of arbitrary geometric configurations,” IEEE Trans. Electron. Comput., vol.EC- 10, issue 2, pp.260–268, 1961.

[5] A. Akimov, A. Kolesnikov, and P. Fränti, “Lossless compression map contours by context tree modeling of chain codes,” Pattern Recognit., vol.40, pp.944–952, 2007.

[6] Y.K. Liu and B. Žalik, “An efficient chain code with Huffman coding,” Pattern Recognit., vol.38, pp.553–557, 2005.

[7] H.S.-C.E. Bribiesca, and R.M. Rodríguez-Dagnino, “Efficiency of chain codes to represent binary objects,” Pattern Recognit., vol.40, pp.1660–1674, 2007.

[8] David A.Huffman, “A method for the comstruction of minimum-redundancy codes,” Proc. I.R.E., Sept. pp.1098–1102, 1952.

[9] 安居院猛，中嶋正之，画像情報処理，森北出版，1991.

（平成21年8月7日受付，11月5日再受付）