LZW 符号語列への情報埋め込み手法 — GIF 画像データを例として

荻 原 剛 志†

LZ77 系のデータ圧縮手法で圧縮されたデータに情報を埋め込む深層暗号手法がすでに提案されている．一方，LZ78 系の圧縮手法，特に LZW 法で圧縮されたデータへの情報埋め込みは困難であった．これは，生成される符号語がアルゴリズムによって一意に決定されてしまうため，情報埋め込みに利用できる冗長性が存在しないためである．本論文では，生成される符号語の系列が何らかの情報を表現するように，圧縮されるデータそのものを操作する手法について述べる．情報を埋め込むカバーデータの例としては GIF 形式の画像を用いる．GIF 形式の画像は，最大 256 色の代表色をパレットとして持ち，代表色の色番号の列が LZW 法で圧縮された形式となっている．本論文では，GIF 画像の LZW 符号語列に情報を埋め込む実験の結果についても示す．

Data Embedding Method into LZW Code Words — Focusing on GIF Image Data

Takeshi Ogihara†

This paper shows a data embedding method into the sequence of code words made by LZW compression method. Data embedding methods for LZ77 compression methods were already proposed. However, with LZ78 compression methods, such as LZW, it is difficult to embed data into the sequence of code words. In this paper, GIF image format is taken up as cover data. GIF is a kind of palette-based image format and it uses LZW compression. Proposed method properly modifies the image data itself so that the sequence of code words would have secret information. This paper also shows the result of embedding into GIF images.

1. まえがき

深層暗号[1)]の一手法として，圧縮されたデータに情報を埋め込む方法が提案されている．

圧縮手法の中には，データを圧縮した結果である符号語列の生成方法に何らかの自由度があるものが存在し，符号語を生成する過程において符号化の方法が 1 通りとは限らない場合がある．たとえば，文字列置換に基づく圧縮手法では，すでに出現した文字列を表す方法が何通りか存在する場合がある．このような圧縮手法においては，選択可能な複数の方法のどれを選ぶかを，埋め込みたいデータの内容に従って決定することによって情報の埋め込みを実現できる．

これらの手法では，圧縮されるデータ自体には変更を加えないため，復元されたデータを調べても埋め込まれた情報の手がかりがないという特徴がある．これまでに，LZ77[2)] 系のデータ圧縮手法で圧縮されたデータに情報を埋め込む手法が提案されている[3)]．

一方，LZ78[4)] 系の圧縮手法，特に LZW 法[5)] で圧縮されたデータへの情報埋め込みは困難であった．これは，生成される符号語がアルゴリズムによって一意に決定されてしまうため，情報埋め込みに利用できる冗長性が存在しないためである．

しかし，生成される符号語の系列が何らかの情報を表現するように，圧縮されるデータそのものを操作することは可能であると考えられる．この方法は，元のデータに変更を加えないという特徴は失っているが，同じ圧縮法を適用すれば再び情報を取り出すことができるという新たな特徴を持つ．

本論文では，圧縮されるデータ自体を操作することによって，LZW 法の符号語列に情報を埋め込む手法を提案する．しかし，データへの変更がどの程度まで許容されるかはデータの種類によって異なる．そこで，本論文では情報を埋め込むカバーデータとして GIF 形式の画像を用い，提案手法が実際に適用可能であることを示すこととする．GIF 形式は限定色画像（パレット画像）の一種であり，最大 256 色の代表色をパレットとして持ち，代表色の色番号の列が LZW 法で圧縮された形式となっている．本論文では色番号を操

† 高知工科大学工学部情報システム工学科
Faculty of Engineering, Kochi University of Technology

作することでLZW法の符号語列に情報を埋め込む．

ただし，この埋め込み手法は限定色画像以外のデータに対しても有効である．限定色画像に適用するうえで採用した固有の実現方法については文中で明示する．

2. 圧縮されたデータへの情報埋め込み

2.1 LZ77系の圧縮手法

まず，LZ77系の圧縮手法が生成する符号語に情報を埋め込む方法の概要について述べる．

LZ77系の圧縮手法では，各符号語は文字列または単一の文字を表す．符号語が文字列を表す場合，その文字列はすでに読み込まれて符号化が済んだ入力列の中に存在する．たとえば，次のような文字列がすでに読み込まれているとする．

```
yabbaabaaabba
```

いま，この直後に続く入力列Aが“abaa”だったとする．この文字列は先行する文字列中の，6文字目から始まる部分文字列に一致することが分かるので，位置と長さを使って(5, 4)と表現できる（1文字目を0から数えるとする）．この表現が元の文字列よりも短いビット長で符号化できれば，データは圧縮できたことになる．LZ77系の手法では，このような原理に基づいて圧縮を行っている．

ところで，文字列の表現方法にはいくつかの冗長性が存在する．上記の例で，入力列Aが“abaabaaabb”だったとする．グリーディなアルゴリズムで，先行する文字列に最も長く一致する文字列を探索すると，この入力列は文字列(5, 4)と(6, 6)の組合せとして表現できる．一方，単一の文字‘a’と文字列(3, 9)の組合せとしても表現でき，こちらの方が生成される符号語のビット長の合計は短くなる可能性がある．

さらに，入力列Aが“abba”である場合を考えてみる．この文字列は，先行する文字列中の2カ所にあり，(1, 4)とも(9, 4)とも表現できる．

LZ77系手法のこのような性質を利用して，生成される符号語に情報を付加することができる[3]（公開されているソフトウェア☆も存在する）．

たとえば，グリーディなアルゴリズムで得られた最長一致文字列を採用した場合にビット0，それ以外の文字列を採用した場合にはビット1を表すように決めておけばよい．また，先行する文字列中の2カ所以上に同じ部分文字列が存在した場合，どれを採用するかによって情報を表すこともできる．

☆ Brown, A. and Pizzini, K.: Gzsteg (1994). http://www.funet.fi/pub/crypt/steganography/

```
w = read a byte;
while ( ( k = read a byte ) != EOF ) {
    if ( [w, k] exists in the dictionary )
        w = [w, k];
    else {
        add [w, k] to the dictionary;
        output code(w);
        w = k;
    }
}
output code(w);
```

図1 LZWアルゴリズム
Fig. 1 LZW algorithm.

2.2 LZ78系の圧縮手法

LZ78系の手法も文字列の置換によって圧縮を実現するが，文字列は辞書と呼ばれるデータ構造に蓄積されていく．

図1に，LZ78系の代表的な手法であるLZW法のアルゴリズムの概要を示す．辞書内の文字列（以下，フレーズと呼ぶ）には正整数値（ID値）が一意に対応づけられており，変数wはこのID値を保持している．ただし，1文字のみからなるフレーズは既知であり，その文字符号がID値であるとする．$[w, k]$という記法は，wの表すフレーズに文字kを連接した新たなフレーズを表すとする．

このアルゴリズムを適用した場合に生成される符号語列は一意に決定され，冗長な符号化が行われる余地がない．実際には，このアルゴリズムは入力列に一致する最長のフレーズをグリーディな方法で辞書内から探しているため，前節で示したように，最長のフレーズを使わないことによって埋め込みが可能であるように思われる．しかし，最長のフレーズを採用しないで符号化を進めた場合，辞書内にまったく同じフレーズが重複して登録される．LZW法は復元の際にも同じ辞書を構成しなければならないため，フレーズの重複登録は容易に検知され，情報埋め込み操作の可能性があることが知られてしまう．

しかし逆に，符号語の作り方を工夫するのではなく，圧縮されるデータそのものを操作し，生成される符号語の系列が何らかの情報を表現するようにできる．次節では，本論文でカバーデータとして利用するGIF画像形式と，GIF画像に対する従来の情報埋め込み手法について述べた後，画像データを操作することによってLZW法の符号語に情報を埋め込む手法について説明する．

3. GIF 画像の圧縮データへの埋め込み

3.1 GIF 画像形式

GIF（Graphics Interchange Format）[6] は，現在最も一般的に利用されている画像形式の 1 つであり，限定色画像（パレット画像）の一種である．

GIF 形式の画像は，最大 256 色の代表色をパレットとして持ち，各画素の色は代表色の色番号で表現される．色番号は各画素 1 バイトで，走査線順に並べられ，LZW 法で圧縮されて画像ファイルに格納されている．なお，GIF 画像には透明色や，アニメーションの機能もあるが，本論文ではこれらについては触れない．

3.2 限定色画像への情報埋め込み

GIF 形式のような限定色画像は，各画素の色を代表色の番号で表すため，フルカラー画像に対する情報埋め込み手法のように，画素値の末尾ビット（LSB）や画像の周波数成分を変更して埋め込みを行うことができない．そのため，限定色画像に特有の情報埋め込み手法がいくつか提案されている[7)~10)]．

最も基本的な方法は，代表色を 2 つのグループ A，B に分けておき，埋め込みたいビットが 0 ならグループ A，1 ならグループ B から，元の画素に最も近い色を選択し，元の色と置き換えるという方法である．元の画素の色を含むグループが選択された場合には，置き換えの必要はない．代表色のグループ分けの方法が分かっていれば，埋め込まれた情報を取り出すのは容易である．

この方法では，色の置き換えにともなって画質が劣化する．情報埋め込み後の画像の画質は，代表色をどのようにグループ分けするかに大きく依存するが，代表色のグループ分けを適切に行えば，この方法でも画質が著しく劣化することはない[7),8)]．

なお，フルカラー画像から減色処理によって限定色画像を生成する過程で，同時に情報埋め込みを行う方法も提案されている[9),10)]．この手法は，与えられた限定色画像への埋め込みに比べ，埋め込み後の画像が高画質で，しかもより多くの情報を埋め込み可能である．ただし以下では，分かりやすさのため，埋め込みの対象となる限定色画像はあらかじめ与えられたものであると仮定する．

3.3 LZW 法の符号語列への埋め込み

前節で述べたように，限定色画像への情報埋め込みでは，ある画素の色を別の色に置き換えることが基本的な手段である．本論文でも，LZW 法が生成する符号語を変更するために画素の置き換えを利用する．

LZW 法の符号語によって情報を表現する方法はいくつか考えられる．LZW 法の符号語はフレーズの ID 値を表す正整数であることから，符号語が偶数・奇数のいずれであるかを使って埋め込みを行うことができる．符号語に対応するフレーズの長さを利用する方法もあろう．また，代表色をいくつかのグループに分割しておき，フレーズの最後にどのグループの色があるかによって情報を埋め込むなどの方法も考えられる．

次章では，符号語が偶数・奇数のどちらであるかを利用して情報を埋め込む手法について説明する．

4. 提案する埋め込みアルゴリズム

4.1 入力列の変更による符号語の変化

LZW 法では，入力列を変更すると生成される符号語列に予想外の変化が発生することがあり，制御するのが難しい．偶数・奇数を利用して埋め込みを行おうとする場合の簡単な例を示す．

いま，埋め込みたいビット $b = 0$ で，偶数の ID 値を符号語として出力したいとする．現在の入力列は以下のようであるとする（整数値は色番号）．

```
01 01 33 11 02 55
```

ここで，列 “01 01 33 11 02” を表すフレーズ w が辞書に登録されており，フレーズ $[w, 55]$ は登録されていないとする．通常の圧縮ではこの時点でフレーズ w の ID 値が符号語として出力される．埋め込み処理においても，ID 値が偶数の場合はそのまま出力すればよい．しかし，w の ID 値が奇数であったとすると，いずれかの色番号を置き換えてフレーズを再構成しなければならない．仮に，02 に最も近い色が 44 で，これと置き換えたとする．しかし，列 “01 01 33 11 44” を表すフレーズ w' が辞書に登録されており，かつ対応する ID 値が偶数であるとは限らない．この条件が満たされなかった場合，元の 02 を 44 以外の色で置き換えるか，あるいはフレーズ w の 02 以外の色を置き換えるかして，フレーズの再構成をやりなおさなければならない．しかし，そのような条件にあてはまる色自体が存在しない可能性もある．もし，44 が条件を満たしていたとしても，次の 55 を加えたフレーズ $[w', 55]$ が辞書に登録されていたとしたら，フレーズ w' は符号語として出力されない．その場合，フレーズ $[w', 55]$ を改めて置き換えの対象とするか，あるいはフレーズ w' を改めて再構成しなければならない．

このように，フレーズの末尾の 1 バイトを置き換える方法を考えただけでも，考えうる可能性は非常に多岐にわたることが分かる．そこで以下では，色番号の置き換えによって LZW 符号語を操作するための発見

的な手法を提案する．

4.2 提案手法の概要

まず，通常どおりに LZW 法でフレーズを構成し，出力すべき符号語の ID 値を得る．その値の偶数・奇数が埋め込みたいビットと一致していればそのまま符号語を出力する．一致していなければフレーズに含まれる画素値を変更し，別の符号語を構成する．符号語の値が埋め込みたいビットと一致するまで，画素値の変更を繰り返す．

ただし，フレーズの先頭バイトは，直前のフレーズとの組合せですでに辞書に登録されているために変更できない．フレーズ長が 2 バイトしかない場合には，2 バイト目の色の置き換えだけでフレーズの ID 値を操作しなければならないが，可能なすべての置き換えについて偶数（または奇数）の ID 値しか得られない可能性がある．

そこで，フレーズ長が 2 バイト以下の場合には埋め込みを行わないこととする．これによって，つねに複数個のバイトが置き換えの対象となり，置き換え可能な色の組合せ数が増大する効果がある．また，埋め込みが困難な場合にはフレーズ長が 2 バイトまで短縮されて，埋め込みをせずに次の処理に移行することができるようになる．この下限のフレーズ長は 2 バイトより長くてもよいが，埋め込みに寄与しないフレーズが増加するため，画像全体に埋め込める情報量は減少する．

フレーズを再構成する際には，まずフレーズの末尾の画素について，いくつかの候補の色との置き換えを試みる．適切なフレーズが構成できなかった場合には末尾から 2 番目の画素，末尾から 3 番目，というように置き換えを行う画素をフレーズの前方に移してゆく．ただし，色の置き換えをする前のフレーズより，置き換えの結果として構成されるフレーズの方が長くなる場合もある．上の例ではフレーズ $[w', \mathtt{55}]$ が辞書に登録されていた場合に相当する．この場合，LZW 法に基づいてさらに長いフレーズの構成を試み，辞書に登録されていないフレーズが得られた時点で上記の手順を適用する．

辞書に登録されているフレーズの長さは有限であるため，フレーズが際限なく長くなることはない．ただし，フレーズの伸長，短縮が繰り返される過程で同じフレーズが出現してしまうと手続きが止まらなくなるおそれがある．これを防ぐために，同じ置き換え候補の組合せを作らないようにする必要がある．

次節で手続きの詳細な定義を示す．

```
w = S[0];
x = 1;
while ( x < data size ) {
    k = S[x];
    if ( [w, k] exists in the dictionary )
        P[x] = [w, k];
    else{
        add [w, k] to the dictionary;
        output code( w );
        P[x] = k;
    }
    w = P[x];
    x++;
}
output code( w );
```

図 2 LZW アルゴリズム：入力列は $S[]$ とする
Fig. 2 LZW algorithm: Byte stream is in $S[]$.

4.3 提案アルゴリズム

準備として，図 1 の LWZ 法のアルゴリズムを別の書き方で記述しなおしたものを**図 2** に示す．ここでは，入力バイト（色番号）列はすべて配列 $S[]$ に格納されていると仮定する．手続きの中で構成されたフレーズの ID 値は，配列 $P[]$ の対応する位置に格納される．

次に，**図 3** に提案する埋め込みアルゴリズムを示す．配列 $D[]$ には，入力列を操作して生成される埋め込み済みの画像を表すバイト列が格納される．

この手法では，ある代表色を別の色で置き換えるとき，できるだけ似た色から順に置き換えを試みるようにしている．色の類似度は色空間内の距離で比較でき，たとえば 5.2 節の実験では RGB 表色空間内でのユークリッド距離を用いている．限定色画像以外に適用する場合は，別な尺度を用いなければならない．配列 $A[]$ は各画素について，現在何番目に似た色まで置き換えを試みたかを覚えておくために使う．初期値は 0 で，置き換えをしていないことを示す．

図 3 を順に説明する．外側のループの最初（9，10 行目）では，新しくフレーズ $[w, k]$ を作り，辞書に存在するかどうかを調べる．存在していれば，さらに長いフレーズを作るために繰返しの先頭に戻る．変数 len はフレーズ w に対応するバイト列の長さを示す．

13 行目から 35 行目までの部分は，フレーズ $[w, k]$ が辞書に存在しなかった場合で，通常の LZW 法ではフレーズ w を出力する部分に相当する．

14 行目から 23 行目の部分は，フレーズ w の長さが 2 以下か，またはフレーズ w の ID 値の偶数・奇数が埋め込みビット b と一致した場合の処理である．この場合にはフレーズを作りなおす必要はないので，フレーズ $[w, k]$ を辞書に登録し，w を出力する．フレーズ w の長さが 2 より長ければ埋め込みができている

```
 1  for (i = 0; i < data size; i++) {
 2     D[i] = S[i];  A[i] = 0;
 3  }
 4  w = D[0];
 5  x = 1;
 6  b = read embedded bit;
 7  len = 1; /* Length of phrase */
 8  while ( x < data size ) {
 9     k = D[x];
10     if ( [w, k] exists in the dictionary ) {
11        w = P[x] = [w, k];
12        x++;  len++;
13     }else if ( len <= 2 || (w & 1) == b ) {
14        add [w, k] to the dictionary;
15        output code( w );
16        w = P[x] = k;
17        if (len > 2)
18           b = read embedded bit;
19        for (i = x + 1; i < data size; i++) {
20           D[i] = S[i];  A[i] = 0;
21        }
22        x++;
23        len = 1;
24     }else {
25        for ( ; ; ) {
26           x--;  len--;  A[x]++;
27           k = get_replacement( S[x], A[x] );
28           D[x] = k;
29           if ( near( S[x], k ) || len <= 1 )
30              break;
31           D[x] = S[x];
32           A[x] = 0;
33        }
34        w = P[x - 1];
35     }
36  }
37  output code( w );
```

図 3 提案アルゴリズム
Fig. 3 Proposed algorithm.

ので，新しい埋め込みビットを得る．また，埋め込みにともなう色の置き換え処理の結果，配列 $D[]$ および $A[]$ の，位置 $x+1$ 以降の値が変化している可能性があるため，これを初期値に戻しておく．ただし，位置 x の情報は新しい w で使うため，元に戻してはならない．

25 行目から 34 行目の部分は，フレーズ w の長さが 2 より長く，かつフレーズ w の ID 値の偶数・奇数が埋め込みビット b と一致しなかった場合の処理である．この場合はフレーズを再構成しなければならず，ループ内で置き換えの位置と候補となる色を探す．

色の置き換え方法の概要は前節で述べたとおりであり，置き換え可能な適切な色が見つかるまで，フレーズ長を短くしながら処理を繰り返す．ここで，関数 `get_replacement(S[x], A[x])` は，すべての代表色の中から，画素 `S[x]` に `A[x]` 番目に近い色を取り出すものとする．また，関数 `near()` は 2 つの色番号を引数とし，色空間内でのそれらの距離があらかじめ決められたしきい値以下の場合に真を返す．置き換えの候補となった色が元の色とかけ離れていると，画質が大きく劣化するため，そのような色を選択しないようにしている．この部分も，限定色画像以外に適用する場合には別な尺度を用いなければならない．

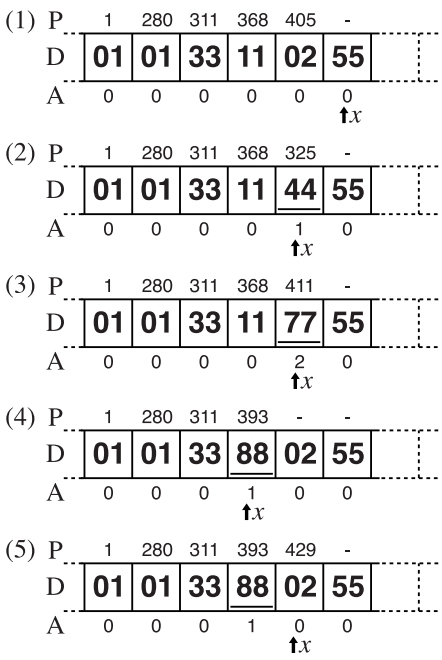

図 4 アルゴリズムの動作例
Fig. 4 Example of the proposed algorithm.

元の色に近い置き換え候補が見つかるか，あるいはフレーズ長が 1 以下になる場合（29 行目）は for ループから抜けて外側のループの先頭（8 行目）に戻り，LZW のフレーズを構成しなおす．

4.4 例によるアルゴリズムの説明

ここでは 4.1 節と同じ簡単な例を使ってアルゴリズムの動作を説明する．

図 4 の (1) は，列 “01 01 33 11 02” を表すフレーズ 405 が辞書に登録されており，フレーズ [405，55] は登録されていない状態である．埋め込みたいビットが 0 だとすると，フレーズ 405 は奇数なので出力できない．そこで，(2) では添字 x を 1 つ減らして，02 を別の色に置き換える．プログラムでは 26 行目からの部分である．02 を 44 に置き換えてループの先頭の 8 行目に戻る．

列 “01 01 33 11” を表すフレーズ 368 と 44 からフレーズ [368，44] が構成され，これが辞書に存在して ID 値は 325 であるとする．さらに，フレーズ [325，

55] は登録されていないとする．325 は奇数なのでやはり出力できず，(3) では先ほどと同様の手順で 44 を 77 に置き換えて，フレーズ 411 を得る．フレーズ [411，55] が登録されていないとすると，411 は出力できない．

この時点で，02 と置き換え可能な色がもう存在しないとする．この場合，プログラムの 31 行目以降が実行され，置き換えられていた 02 が元に戻され，添字 x はさらに 1 つ減って 11 の置き換えを試みる．(4) は 11 が 88 に置き換えられ，列 “01 01 33 88” を表すフレーズの ID 値が 393 であったことを示す．

次の (5) はフレーズ [393，02] の ID 値が 429 であったことを示している．フレーズ [429，55] が辞書に存在しなければ，再び 02 が置き換えられるが，今度は直前の色が 88 に置き換えられているので (2)，(3) とは別の組合せについて調べることになる．

提案アルゴリズムはこのように色の置き換えの組合せを調べ，埋め込みたいビットに合致するフレーズのうちで最初に見つかったものを採用する．どのような組合せでも適切な組合せが見つからない（上記の例ならばつねに ID 値が奇数になってしまう）場合には，添字の x が減らされてゆき，フレーズ長が 2 以下になる．

4.5 同一色が連続するフレーズの除外

図 3 で提案したアルゴリズムをそのまま実装すると，画像中で同じ色が連続した部分にも埋め込みをしてしまうため，周囲とは異なる色が不自然に点在する結果となる．そこで，同じバイトの繰返しだけからなるフレーズには埋め込みをしないように改良を加えると画質が改善される．

具体的には，図 3 の 13 行目の if 文の条件に，フレーズ w が同じ色番号の連続であるという条件を論理和で追加し，このフレーズを埋め込みの対象としないようにすればよい．

このような場合を排除することは，今回対象としている画像データだけではなく，音声データなどにおいても不自然なノイズの混入を防止するという意味があると考えられる．

4.6 埋め込みデータの取り出し

提案手法で情報を埋め込んだ GIF 画像から埋め込まれた情報を取り出すには，LZW 法の符号語が偶数か奇数かによってビットの値を決定すればよい．ただし，符号語に対応するフレーズの長さが 2 バイト以下の場合には情報は埋め込まれていない．**図 5** に LZW 法の復元手順をもとにした取り出しアルゴリズムを示す．

配列 $P[\,]$ には，埋め込み済みの LZW の符号語（ID 値）列が格納されているとする．最初の符号語 $P[0]$ は必ず単一のバイトを表している（1 行目）．変数 *fst* はその符号語が表すバイト列の先頭のバイトを表す．外側のループで符号語を順番に調べるが，5 行目以降の else 節は符号語 $P[x]$ が 2 バイト以上を表すフレーズだった場合の処理である．LZW 符号においては，その時点で辞書に登録されていない ID 値を持つフレーズが入力列に現れるという特殊なパターンが存在する[5]．8 行目から 11 行目はこのパターンに対応するためのものである．

```
 1  fst = P[0];  /* P[0] is a single byte */
 2  for (x = 1; x < phrase number; x++) {
 3      if (P[x] is a single byte) {
 4          fst = P[x];
 5      }else {
 6          len = 0; /* Length of bytes */
 7          code = P[x];
 8          if (code is not in the dictionary)
 9              str[len++] = fst;
10              code = P[x-1];
11          }
12          while (code is not a single byte) {
13              get_entry( code, &w, &k );
14              /* code is [w, k] */
15              str[len++] = k;
16              code = w;
17          }
18          str[len++] = fst = code;
19          if (len > 2)
20              output bit(P[x] & 1);
21      }
22      add [P[x-1], fst] to the dictionary;
23  }
```

図 5 情報取り出しアルゴリズム
Fig. 5 Extracting algorithm.

12 行目からのループでフレーズ *code* をバイト列に変換する．LZW 法の辞書には，各フレーズごとに，そのフレーズの末尾バイトを取り除いたフレーズと末尾バイトが登録されている．手続き`get_entry()` は，フレーズ *code* を構成しているフレーズ w とバイト k を辞書から取り出すものである．このループの処理が終わると，配列 $str[\,]$ にフレーズ $P[x]$ に対応するバイト列が逆順に格納されている．

変数 *len* はバイト列の長さを表す．長さが 2 より長い場合には，フレーズ $P[x]$ が奇数の場合に 1，偶数の場合に 0 を出力すればよい．前節で述べた改良を付け加えた場合，同一色の連続を表すフレーズかどうかもチェックする必要があるが，このためには配列 $str[\,]$ の内容がすべて同じバイトかどうかを調べればよい．

最後に，直前の符号語 $P[x-1]$ と $P[x]$ の先頭バイトである *fst* からなる新しいフレーズを辞書に登録


(a) n1a


(b) n2a


(c) Lenna


(d) map

図 6 実験用画像
Fig. 6 Test images.

する．

このように，埋め込まれた情報を取り出すには，LZW の符号語列のみが与えられていればよく，パレットや色の類似を判断するしきい値などの情報は必要ない．

さらに，辞書を構成しながら情報を取り出すのであるから，復元の過程ではなく，復号後の限定色画像を LZW 符号に再圧縮する過程でも情報を取り出せることに注意したい．

5. 実　　験

5.1 実験対象の画像と埋め込みデータ

実験では，**図 6** の 4 つの画像に対して埋め込みを行った．画像 (a) n1a，(b) n2a は JIS 標準画像データ☆1の画像を 1/3 に縮小して GIF 形式画像としたものである．(c) Lenna は USC-SIPI 画像データベース☆2の画像を GIF 形式にしたものである．(d) map は情報処理学会の Web ページ☆3に含まれていた GIF 画像である．いずれの画像も，含まれている色の数（パレット内の代表色数）は 256 色である．

埋め込みデータとして，英文の小説☆4を gzip で圧縮したものを利用した．圧縮によって 0 と 1 のビットがほぼランダムな割合で出現する．

実験では，4.5 節で述べたように，同一色が 2 バイト以上連続するフレーズは埋め込みの対象としていない．また，2 つの代表色の RGB 表色空間内でのユークリッド距離 d がしきい値 $T = 20$ より小さい場合に関数 `near()` は真となり，2 つの色を置き換えて使うようにしている．このしきい値は文献 9) の実験で用いられている最も小さな値であり，画質への影響が小さいことが分かっている．

5.2 実 験 結 果

実験結果を**表 1** に示す．表中のサイズは，圧縮されていない状態のパレットと色番号の列の大きさを示す．これを通常の GIF 形式に変換したときの大きさと，埋め込みをして作成した GIF 画像の大きさ，およびそれぞれの圧縮率を示している．ここでの圧縮率は，圧縮前の大きさに対する圧縮後の大きさを表す．

埋め込みビット数は，その画像に埋め込むことができた情報をビット数で表している．括弧内の数値は，全画素数に対する埋め込みビット数の割合を示す．以下ではこの数値を埋め込み率と呼ぶことにする．埋め込み率が高いほど，画素数あたりに多くの情報の埋め込みができたことを意味する．

PSNR（ピーク SN 比）は埋め込み前と埋め込み後の画像の比較から算出した．なお，PSNR の計算には以下の式を用いた．ここで，M と N は画像の横と縦のピクセル数，$p_k(i,j)$ および $p_k{}'(i,j)$ は，それぞれ埋め込み前と後の画像における位置 (i,j) のピクセルの k（R,G,B のいずれか）色成分を表す．

$$PSNR = 10\log_{10}(255^2/MSE)$$

$$MSE = \frac{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\sum_{k=r,g,b}(p_k(i,j) - p_k{}'(i,j))^2}{3MN}$$

提案手法では，フレーズ長を短くしながら色の置き換えを試みるが，LZW 法ではフレーズ長の平均が短いほど圧縮率は低くなる．このため，埋め込み後の画像は埋め込みのない画像に比べて圧縮後のサイズが数パーセント程度大きくなる傾向がある．

表 1 の n1a，n2a，および Lenna の結果から，圧縮率の良い画像ほど埋め込み率が高いという傾向が読みとれる．提案手法では 2 バイトより長いバイト列に対応するフレーズに埋め込みを行うため，このような傾向が見られると考えられる．

一方，map は圧縮率が良いにもかかわらず埋め込み率は高くない．この画像には同一色で塗りつぶされた

☆1 ISO/JIS-SCID: Standard colour image data, ISO/DIS 12640, JIS X 9201-1995.
☆2 http://sipi.usc.edu/services/database/Database.html
☆3 http://www.ipsj.or.jp/03somu/map.html
☆4 L. M. Montgomery, “Anne of Green Gables”, 1908.

表 1 実験結果
Table 1 Results of embedding.

画像	横 × 縦	サイズ (byte)	GIF - 埋め込みなし	GIF - 埋め込みあり	埋め込みビット数	PSNR (dB)
n1a	512×640	328,482	160,578 (48.88%)	173,084 (52.69%)	43,705 (13.34%)	45.09
n2a	512×640	328,482	203,518 (61.96%)	210,783 (64.17%)	30,295 (9.25%)	42.21
Lenna	512×512	262,946	213,186 (81.08%)	220,204 (83.74%)	18,049 (6.89%)	48.21
map	576×477	275,537	29,430 (10.68%)	34,014 (12.34%)	11,562 (4.21%)	51.33

領域が多い．また，見やすさのために互いに異なる色を使って作図されているため，相互に置き換え可能な似た色があまり存在しない．そのため，風景の画像などと比較して埋め込める箇所が少ないと考えられる．

5.3 画質と埋め込み可能な情報量

埋め込み前の画像と埋め込み後の画像を直接比較しても，肉眼では区別が難しい．しきい値 T が大きいほど，かけ離れた色の間の置換が起きる可能性が高くなるが，文献 9) では本実験（$T = 20$）より大きなしきい値に基づく色の置換によって限定色画像へ情報を埋め込み，主観評価実験を行っている．その結果，人間の目には画像の変更が分からないという結果が示されていることから，本実験で作成した画像も，改ざんに気づかれる可能性は低いと考えられる．

限定色画像を対象とした埋め込み手法では，画質の劣化を問題にしなければ，基本的にすべての画素に対して 1 ビットの情報を埋め込むことができる．文献 9) の方法では，しきい値などのパラメータを調整して，埋め込み率で 70%以上の情報を埋め込んでも画質が著しく劣化することはない．提案手法では数バイトのフレーズあたりたかだか 1 ビットしか埋め込むことができないため，表 1 に示すように埋め込み率は 10%程度である．ただし，本手法は LZW 法の符号語への埋め込みを目的としているため，限定色画像に特化した手法に埋め込み率で及ばないのはむしろ当然である．LZ77 系の圧縮手法を用いて符号語列に情報を埋め込む手法[3] でも，入力バイト数に対する埋め込みビット数の割合は，テキストの場合で約 15%，画像の場合で 4.7%となっている．

5.4 色の置き換えについて

提案手法では，フレーズを再構成するためにフレーズの末尾から手順に従って色の置き換えを試みる．元の画素値から何番目に近い色で置き換えを行ったのかは，図 3 における配列 $A[]$ で分かる．

画像 n2a と Lenna の全画素について，何番目に近い色で置き換えを行ったかをまとめた．**表 2** において，「色の近さ」が 1 の場合，元の画素に一番近い代表色で置き換えたことを示し，以下，2 の場合は 2 番目に近い色での置き換えの回数を示す．ただし，「色の近さ」が 0 の場合は置き換えが発生していないことを表す．また，括弧内は全画素数に対する割合である．

表 2 置き換えられた色の傾向
Table 2 A trend of replaced colors.

色の近さ	n2a	Lenna
0	308,628 (94.19%)	249,829 (95.30%)
1	16,176 (4.94%)	9,744 (3.72%)
2	2,354 (0.72%)	1,884 (0.72%)
3	338 (0.10%)	493 (0.19%)
4	94 (0.03%)	118 (0.05%)
5	45 (0.01%)	50 (0.02%)
6	14 (0.00%)	18 (0.01%)
7	7 (0.00%)	8 (0.00%)
8	9 (0.00%)	0 (0.00%)
≥9	15 (0.00%)	0 (0.00%)

この結果から，何度も色の置き換えを繰り返さなければならない場合が確かにあるものの，全体からみればごく少数にとどまっていることが分かる．他の画像についても追試を行い，同様の傾向が認められた．

6. 議　　論

6.1 提案手法の特徴

情報埋め込み手法のいくつかは，埋め込んだ情報を取り出すために付加的な情報を必要とする．3.2 節で説明した限定色画像に対する埋め込み手法でも，代表色のグループ分けについての情報が必要となる．たとえば文献 10) の方法では数百バイトの付加情報が必要であり，文献 8) の方法ではパレットの色は正確に同じに保持されていなければならない．

一方，提案手法は LZW 法の符号語列のうち，フレーズ長が 2 バイトより長いもののみから情報を取り出す．この符号語列は，画素を表すバイト列だけから構成できるため，パレットも含め，付加情報は必要ない．また，ある画像に対してパレットをどのように構成しても，画素値と色番号が 1 対 1 に対応する限り，LZW 圧縮を行ったときに辞書に登録されるフレーズの系列は必ず同じものになる．したがって，情報を埋め込んだ画像を，たとえば PNG や TIFF，JPEG2000 のような歪みのない圧縮が可能な画像形式に保存しても，これらの形式の画像を再び GIF 形式に変換すれば埋め込まれたデータを取り出すことができる．

限定色画像だけではなく，他の種類のデータに対して本手法を適用した場合にも，情報を取り出すために付加情報が必要ないという性質と，データが変形しない限り埋め込まれたデータが失われないという性質は同じである．画像や音声のデータでは，元のデータに変更を加えずにファイル形式（圧縮方式やバイトの順序）だけが変更されることがあるが，このような操作は改ざんとはいえない．したがって，伸長後のデータからも情報が取り出せるという特徴は，「壊れやすい電子透かし[1)]」などの用途に対して有用であると考えられる．

反面，情報の取り出しは非常に容易であるため，情報を秘匿する深層暗号の目的で利用しようとする場合には，通常の暗号化手法で処理したデータを埋め込むなどの対策が必要である．これによって，暗号鍵を持たなければ復元はもちろん，情報が埋め込まれているかどうかの判断も困難にすることができる．

限定色画像の場合はさらに，画像の濃淡や色あいを変化させるといった操作を行っても，異なる代表色どうしが同じ色にならない限りは情報を取り出すことが可能である．

6.2 他の圧縮手法を用いた埋め込み

圧縮されたデータへの埋め込み手法として，LZ77 系の圧縮アルゴリズムを利用した例を 2.1 節で紹介した．この方法は圧縮対象のデータは変更しないため，どんな種類のデータに対しても適用可能であるという特長がある．当然，いったんデータを伸長すると埋め込まれたデータは失われてしまう．

この手法の欠点と考えられるのは，圧縮，伸長のアルゴリズムが公開されている場合，埋め込み対象となった同じデータに再び圧縮を適用し，元のデータと比較すれば，埋め込みの有無を容易に検出できることである．

一方，提案手法は圧縮対象のデータを操作するため，変更が難しいテキストなどのデータには適用できない．ある程度の誤差が許容される画像（限定色画像に限らない），音声などのデータには同様の手法が適用できると考えられる．

また，提案手法は LZW 法を利用したが，データ自体を変更することによって圧縮の符号語を操作するという手法は，LZ77 系をはじめ，他の圧縮手法にも適用が可能である．たとえば LZ77 系の手法では，一致する文字列の長さや，その文字列が以前に出現した位置などを操作可能である．

6.3 手法の改良

提案手法では，得られたフレーズの末尾から順番に色の置き換えを試みる発見的な手法を採用しているが，置き換え可能な組合せをこの方法ですべて調べられるわけではない．すべての組合せを列挙し，その中から最も画質を損なわない置き換え方法を探す方法もありうる．しかし，置き換えによってフレーズの長さが変化するため，組合せを効率良く探索する手法を提案するのは困難であると思われる．表 2 の実験の結果を見ても，ほとんどの場合についてたかだか数回以内の試行で埋め込みが可能となっていることから，計算コストの高い探索手法を採用する意義は低い．

7. むすび

本論文では，入力データ自体を変更することによって，そのデータを圧縮して得られる LZW 符号語列に情報を埋め込む新しい深層暗号手法を提案した．提案手法は，埋め込まれた情報を取り出すために付加情報を必要とせず，データが伸長された後でも取り出しが可能であるという特徴を持つ．

提案手法の中核となっているのは，LZW 圧縮手法のフレーズを構成するバイト列の一部を置き換えることによって，生成される LZW 符号語を操作する発見的なアルゴリズムである．本論文では，埋め込みの対象として GIF 画像に代表される限定色画像を用いて提案手法の実現例を示したが，ある程度の誤差が許容されるデータには提案手法を同様に適用可能である．本論文では実験により，提案手法によって実際に情報を埋め込み可能であることを示した．

これまでにも LZ77 系などの他の圧縮データへの情報埋め込み手法の提案はあったが，LZW 符号語への埋め込みは例がなかった．埋め込みが困難であると思われる箇所にあえて情報を埋め込むという点で深層暗号への利用が考えられるが，データに対する改ざんの検出などの用途への応用も検討して行きたい．

参考文献

1) 松本 勉：インフォメーションハイディングの概要，情報処理，Vol.44, No.3, pp.227–235 (2003).
2) Ziv, J. and Lempel, A.: A universal algorithm for sequential data compression, *IEEE Trans. Inf. Theory*, Vol.23, No.3, pp.337–343 (1977).
3) Atallah, M.J. and Lonardi, S.: Authentication of LZ-77 compressed data, *ACM Symposium on Applied Computing* (*SAC 2003*) (2003).
4) Ziv, J. and Lempel, A.: Compression of individual sequences via variable-rate coding, *IEEE Trans. Inf. Theory*, Vol.24, No.5, pp.530–536 (1978).
5) Welch, T.A.: A Technique for High-Perfor-

mance Data Compression, *IEEE Computer*, Vol.17, No.6, pp.8–19 (1984).
6) CompuServe Inc.: *Graphics Interchange Format, Version 89a* (1989–1990).
7) Fridrich, J. and Du, R.: Secure Steganographic Methods for Palette Images, *Intl. Workshop on Information Hiding 1999*, pp.47–60 (1999).
8) 井上光平ほか：限定色画像への電子透かし埋め込み，電子情報通信学会論文誌，Vol.J82-A, No.11, pp.1750–1751 (1999).
9) 陳　那森，荻原剛志，金田悠紀夫：限定色画像に対する画像劣化の少ない深層暗号手法，画像電子学会誌，Vol.31, No.3, pp.370–377 (2002).
10) 陳　那森ほか：限定色画像に対する深層暗号手法の改良について，情報処理学会論文誌，Vol.44, No.5, pp.1329–1332 (2003).

(平成 16 年 12 月 13 日受付)
(平成 17 年 10 月 11 日採録)

荻原　剛志（正会員）

平成 2 年大阪大学大学院基礎工学研究科博士後期課程修了．同年大阪大学情報処理教育センター助手．平成 5 年奈良先端科学技術大学院大学情報科学センター助教授．平成 7 年神戸大学工学部助教授．平成 9 年神戸大学大学院自然科学研究科助教授．平成 17 年 9 月より高知工科大学工学部教授．ソフトウェア工学，データ圧縮，深層暗号等の研究に従事．Mac OS X 上のソフトウェア開発に興味を持つ．工学博士．IEEE，電子情報通信学会，日本ソフトウェア科学会，情報理論とその応用学会各会員．