Lucrarea 9

Compresia imaginilor

BREVIAR TEORETIC

Termenul de compresie se referă la totalitatea metodelor ce au drept scop reducerea cantităţii de date necesare pentru reprezentarea unei imagini. Compresia este folosită în special pentru stocarea sau transmiterea imaginilor.

Să considerăm cazul unei imagini de dimensiune 512×512 pixeli. Dacă aceasta este o imagine în tonuri de gri, iar fiecare pixel este codat cu 8 biţi, atunci cantitatea de date necesară pentru a reprezenta această imagine este:

$$512 \times 512 \times 8 = 2^9 \times 2^9 \times 2^3 = 2^{21} \approx 2\ Mb$$

Din acest calcul ne putem da seama că pentru a stoca o imagine avem nevoie de spaţiu considerabil, iar pentru transmiterea ei avem nevoie de un canal de transmisiune de bandă largă, de care nu dispunem întotdeauna.

9.1 Clasificarea metodelor de compresie

Metodele de compresie se pot clasifica astftel:

- **Metode de compresie la nivel de pixel**
  Aceste metode nu ţin cont de corelaţia care există între pixelii vecini, codând fiecare pixel ca atare. Acest tip de compresie este fără pierdere de informaţie, adică imaginea iniţială poate fi refăcută perfect din imaginea comprimată. Exemple de astfel de metode:
  - codarea Huffman
  - codarea LZW (Lempel-Ziv-Walsh)
  - codarea RLE (Run Length Encoding)
- **Metode de compresie predictive**
  Aceste metode realizează compresia folosind corelaţia care există între pixelii vecini, dintr-o imagine. Exemple de astfel de metode:

  - codarea cu modulaţie “delta”
  - codarea DPCM (Differential Pulse Code Modulation)

- **Metode de compresie cu transformate**
  Aceste metode se bazează pe scrierea imaginii într-o altă bază, prin aplicarea unei transformări unitare, atfel încât energia imaginii să fie concentrată într-un număr cât mai mic de coeficienţi.

- **Alte metode de compresie**

  - cuantizarea vectorială
  - codarea folosind fractali
  - codarea hibridă

9.2 Algoritmul Huffman

Să presupunem că valorile pixelilor unei imagini sunt simboluri ale unei surse S:

$$[S] = \{S_1, S_2, ..., S_N\} \tag{9.1}$$

pentru care se cunosc probabilităţile de apariţie:

$$[P] = \{p_1, p_2, ..., p_N\} \tag{9.2}$$

$$P(S_1) = p_1 \quad P(S_2) = p_2 \quad P(S_N) = p_N \tag{9.3}$$

Aceste probabilităţi nu reprezintă altceva decât frecveţele relative de apariţie ale simbolurilor într-un şir de simboluri emise de sursa S.

Entropia sursei S care generează simbolurile se calculează cu formula:

$$H(S) = -\sum_{i=1}^{N} p_i \cdot log p_i \tag{9.4}$$

Ne propunem să codăm simbolurile sursei S cu simboluri ale unei alte surse (de exemplu o sursă care generează doar două simboluri: 0 şi 1), astfel încât entropia noii surse să fie maximizată.

În continuare este prezentată o metodă care maximizează această entropie, metodă elaborată de Huffman în 1952:

Pasul 1. Se ordonează descrescător probabilităţile p_i.

Pasul 2. Se formează un arbore binar, având ca frunze valorile celor mai mici probabilităţi din şirul de probabilităţi. Rădăcina acestui arbore va conţine suma probabilităţilor celor două frunze ale sale. Se etichetează muchia stângă cu 1 şi muchia dreaptă cu 0.

Pasul 3. Din şirul P se elimină cele două probabilităţi care au fost alese ca fiind cele mai mici. În şirul P se introduce valoarea conţinută de rădăcina arborelui binar format.

Pasul 4. Dacă în şirul P există mai mult de un element, atunci se reia algoritmul, de la **Pasul 1**.

Pasul 5. Codarea binară a fiecărui element se obţine prin parcurgerea arborelui ce s-a format, de la rădăcină spre fiecare frunză.

Eficienţa codificării Huffman este dată de lungimea medie $\overline{l}$ a cuvintelor de cod, care se calculează folosind formula:

$$\overline{l} = \sum_{i=1}^{N} l_i \cdot p_i \tag{9.5}$$

unde l_i este lungimea codului alocat simbolului S_i.

Exemplu:
Fie o sursă S care generează 4 simboluri, $[S] = \{a, b, c, d\}$, care au următoarele probabilităţi de apariţie: $[P] = \{0.2; 0.4; 0.1; 0.3\}$. Arborele codării Huffman se construieşte conform etapelor prezentate în Figurile 9.1, 9.2, 9.3 şi 9.4.

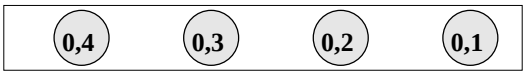

Figura 9.1: Algoritmul Huffman: etapa 1.

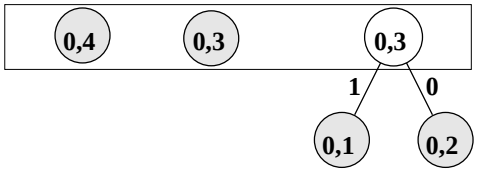

Figura 9.2: Algoritmul Huffman: etapa 2.

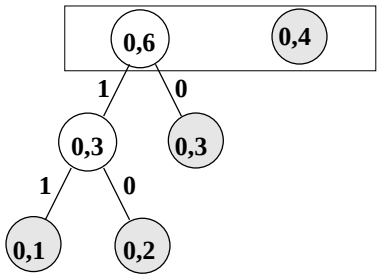

Figura 9.3: Algoritmul Huffman: etapa 3.

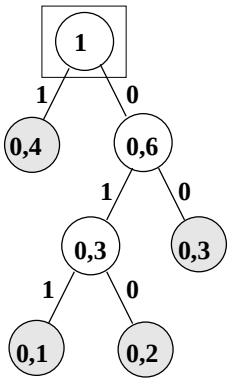

Figura 9.4: Algoritmul Huffman: etapa 4.

Pentru decompresie este necesară o tabelă în care să se memoreze corespondenţele între simboluri şi cuvintele de cod. Fără aceasta decompresia este imposibil de realizat.

Simbol	**Cuvânt de cod**
a	”010”
b	”1”
c	”011”
d	”00”

Lungimea medie a cuvintelor de cod, pentru acest exemplu, este:

$$\overline{l} = \sum_{i=1}^{4} p_i \cdot l_i = 0,2 \cdot 3 + 0,4 \cdot 1 + 0,1 \cdot 3 + 0,3 \cdot 2 = 1,9\, bits/simbol$$

Dacă nu am fi codat simbolurile, în vederea maximizării entropiei sursei, ar fi fost nevoie de 2 biţi/simbol pentru codare.

Pentru imagini, probabilităţile de apariţie ale nivelelor de gri se obţin prin calcularea histogramei imaginii. Dacă histograma este uniformă, atunci algoritmul Huffman de codare nu este eficient, nerealizând nici o îmbunătăţire a lungimii cuvintelor de cod.

9.3 Algoritmul RLE

9.3.1 Algoritmul RLE pentru imagini binare

Vom considera valorile pixelilor (0 sau 255) ca fiind simbolurile 0 şi 1 generate de o sursă binară. În vederea codării imaginea este transformată într-un şir unidimensional, prin concatenarea liniilor sau a coloanelor, ca în Figura 9.5.

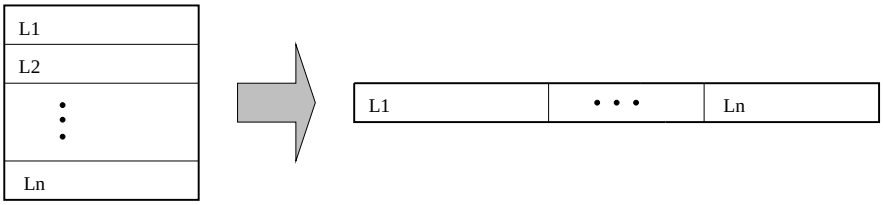

Figura 9.5: Transformarea imaginii într-un şir unidimensional, prin concatenarea liniilor.

Acest şir de elemente 0 şi 1 va fi codat, codarea realizându-se astfel: primul element al şirului codat este primul element din şirul de codat; apoi se scrie în şirul codat lungimea fiecărui subşir constant din şirul de codat.

Exemplu:
şirul de codat: 000000011111000110000000001010001111111111111
şirul codat: 0 7 5 3 2 9 1 1 1 3 12

Acest tip de codare se foloseşte în special pentru comprimarea imaginilor transmise prin fax.

Decompresia se face similar cu compresia, parcurgând şirul codat şi generând şiruri alternate, de simboluri 0 sau 1, începând cu primul element din şirul codat, şi de lungimi indicate de valorile întâlnite în şirul de decodat.

9.3.2 Algoritmul RLE pentru imagini în tonuri de gri

Pentru imagini în tonuri de gri, algoritmul RLE se aplică pentru plane formate din biţii de pe aceeaşi poziţie, din reprezentarea binară a valorilor pixelilor. De exemplu, dacă imaginea în tonuri de gri, are 256 de nivele de gri, corespunzător la o cuantizare pe 8 biţi, atunci din această imagine se

construiesc 8 plane (sau 8 imagini binare) astfel: o imagine binară formată din biţii b_0, o altă imagine binară din biţii b_1, ş.a.m.d. (vezi Figura 9.6).

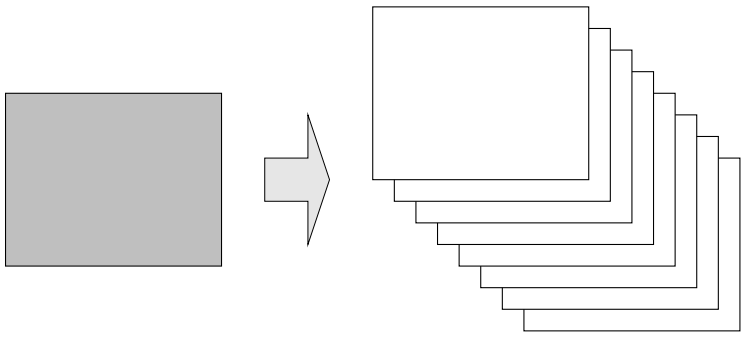

Figura 9.6: Transformarea unei imagini cu 256 nivele de gri, în 8 imagini binare.

Valoarea pixelului (i, j) din imaginea în tonuri de gri va fi reprezentată pe 8 biţi astfel:

$$val(i, j) = [b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7]$$

unde b_0 este cel mai semnificativ bit (MSB[1]), iar b_7 este cel mai puţin semnificativ bit (LSB[2]).

Imaginea binară formată din biţii cei mai semnificativi va fi comprimată cel mai bine cu algoritmul RLE. Imaginea binară formată din biţii cei mai puţin semnificativi va fi o imagine cu "purici", pentru care se poate lua decizia de a nu mai fi codată şi deci ignorată.

9.4 Compresia cu transformate

Compresia cu ajutorul transformatelor se bazează pe proprietatea acestora de a compacta energia imaginii într-un număr redus de coeficienţi, cât mai decorelaţi, repartizaţi neuniform în spaţiul transformării.

Formula care defineşte o transformarea directă este următoarea:

$$V = A \cdot U \cdot A^T \tag{9.6}$$

unde A este matricea ce defineşte o transformare unitară, separabilă.

Pentru compresia imaginilor, transformarea cea mai apropiată din punct de vedere al performanţelor de transformarea optimă Karhunen-Loéve, este transformarea cosinus. Coeficienţii de energie mare sunt situaţi în colţul

[1]Most Significant Bit.
[2]Least Significant Bit.

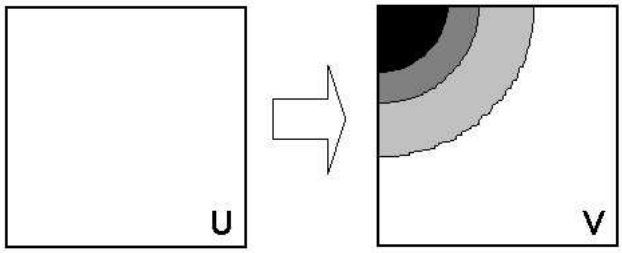

Figura 9.7: Transformarea directă.

din stânga-sus al imaginii transformate, în cazul în care se foloseşte pentru compresie transformarea cosinus (vezi Figura 9.7).

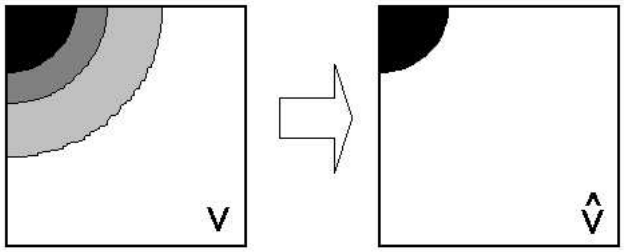

Figura 9.8: Anularea coeficienţilor de energie mică.

Pentru a obţine o rată de compresie mai mare, vor fi anulaţi coeficienţii de energie mică (vezi Figura 9.8). Anularea acestor coeficienţi va duce, însă, la scăderea calităţii imaginii după decompresie. Adică, prin transformare inversă (vezi Figura 9.9), imaginea obţinută din imaginea $\hat{V}$, nu va fi exact imaginea originală U.

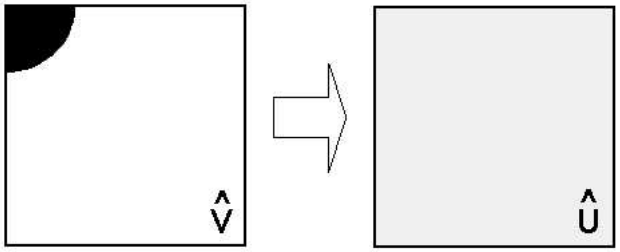

Figura 9.9: Transformarea inversă.

$$\hat{U} = A^{*T} \cdot \hat{V} \cdot A^{*} \tag{9.7}$$

Compresia cu transformarea cosinus stă la baza algoritmului JPEG[3] de compresie a imaginilor.

DESFĂŞURAREA LUCRĂRII

Problema 1. Implementaţi în C unul dintre algoritmii prezentaţi (Huffman sau RLE). Pentru simplitate, imaginea citită în format BMP, va fi scrisă într-un format simplificat (vezi funcţia `write_naked_image`, din meniul `Algortimi`). Imaginea comprimată va fi scrisă într-un fişier cu exensia `.huf` sau `.rle`.

```
void ImageViewer :: write_naked_image( void )
{
   int i, j;
   int w, h;
   FILE *file;

   w = image.width();
   h = image.height();

   file = fopen( "naked.img", "w" );
   if( file != NULL )
   {
      for( i = 0; i < w; i++ )
      {
         for( j = 0; j < h; j++ )
         {
            int niv = qRed( image.pixel( i, j ) );
            //nivelul de gri al pixelului

            fprintf( file, "%3d ", niv );
         }
         fprintf( file, "\n" );
      }

      fclose( file );
   }
}
```

Problema 2. Calculaţi raportul de compresie obţinut, ca raport dintre dimensiunile celor două fişiere: cel original (`naked.img`) în formatul simplificat şi cel comprimat.

[3]Joint Photographic Experts Group.

Problema 3. Observaţi efectul suprimării coeficienţilor de energie joasă, la o compresie-decompresie folosind transformata cosinus discretă. (funcţia `compresie_decompresie_cu_DCT`, din meniul `Algoritmi`). Pentru aceasta vizualizaţi imaginea `decompresata.bmp`. Codul prezentat în continuare presupune că imaginea este pătrată:

```
void ImageViewer :: compresie_decompresie_cu_DCT( void )
{
   int w, h;
   int i, j, k;
   double pi = 3.1415926;

   w = image.width();
   h = image.height();

   int N = w;
   double  max;
   double  C[ N ][ N ]; //matricea transformarii cosinus
   double  U[ N ][ N ]; //imaginea in spatiul original
   double  V[ N ][ N ]; //imaginea in spatiul transformatei
   double  AUX[ N ][ N ];

   // COMPRESIA IMAGINII
   //formarea matricei C a transformarii cosinus discreta
   for( i = 0; i < N; i++ )
      C[ 0 ][ i ] = 1. / sqrt( N );

   for( i = 1; i < N; i++ )
      for( j = 0; j < N; j++ )
         C[ i ][ j ] = sqrt( 2./N ) *
            cos( pi * ( 2*j + 1 ) * i / ( 2*N ) );

   //formarea matricei U
   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
         U[ i ][ j ] = qRed( image.pixel( i, j ));

   //V = C*U*Ct
   //mai intii vom calcula AUX = C * U
   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
      {
         AUX[ i ][ j ] = 0;
         for( k = 0; k < N; k++ )
```

```
            AUX[ i ][ j ] += C[ i ][ k ] * U[ k ][ j ];
      }

   //apoi V = AUX * Ct
   max = 0;
   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
      {
         V[ i ][ j ] = 0;
         for( k = 0; k < N; k++ )
            V[ i ][ j ] += AUX[ i ][ k ] * C[ j ][ k ];

         if( V[ i ][ j ] > max )
            max = V[ i ][ j ];
      }

   //anularea coeficientilor
   //in vederea maririi factorului de compresie
   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
      {
         if( V[ i ][ j ] < 100 )  //pragul de anulare
            V[ i ][ j ] = 0;

         //alte valori prag: -500, -100, 0, 100, 500
      }

   // DECOMPRESIA IMAGINII
   //U = Ct * V * C
   //AUX = Ct * V
   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
      {
         AUX[ i ][ j ] = 0;
         for( k = 0; k < N; k++ )
            AUX[ i ][ j ] += C[ k ][ i ] * V[ k ][ j ];
      }

   //apoi U = AUX * C
   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
      {
         U[ i ][ j ] = 0;
         for( k = 0; k < N; k++ )
```

```
            U[ i ][ j ] += AUX[ i ][ k ] * C[ k ][ j ];
      }

   //pseudo-imaginea diferenta
   QImage diff( N, N, 32, 0, QImage:: IgnoreEndian );

   for( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ )
      {
         int dif = abs( qRed( image.pixel( i, j ) ) -
            (int)( U[ i ][ j ] ) );
         diff.setPixel( i, j, qRgb( dif, dif, dif ) );
      }

   iio.setImage( diff );
   iio.setFileName( "diferenta.bmp" );
   iio.setFormat( "BMP" );
   iio.write();
}
```

Problema 4. Calculaţi eroarea pătratică medie (ε) dintre imaginea originală şi imaginea obţinută prin compresia şi decompresia cu transformată cosinus discretă, folosind formula:

$$\varepsilon = \overline{(U-V)^2} = \frac{1}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}[u(i,j)-v(i,j)]^2$$

Pentru aceasta modificaţi funcţia `Compresie-decompresie cu DCT`. Observaţi valorile erorii medii pătratice pentru diverse valori ale pragului de anulare a coeficienţilor în spaţiul transformării.

Problema 5. Observaţi pseudo-imaginea diferenţă, dintre imaginea originală şi cea obţinută prin compresia şi decompresia cu DCT, pentru diferite valori ale pragului (vezi fişierul `diferenta.bmp`).