AN ALGORITHM USING WALSH TRANSFORMATION FOR COMPRESSING TYPESET DOCUMENTS

Attila Fazekas and András Hajdu

fattila@math.klte.hu hajdua@math.klte.hu

Lajos Kossuth University
4010, Debrecen PO Box 12, Hungary

Dedicated to L. Hajdu on the occasion of his 30th birthday.

Abstract. In this paper the authors present an algorithm which can be used for compressing text documents, principally. The algorithm allows some loss of information, but the original digital image is compressed in a rather efficient way, so the result compressed data structure is suitable to be transmitted through some kind of telecommunication channel. The original document is assumed not to contain sophisticated typographical details, but text, and some simple graphics. The compression algorithm tries to recognize the text parts of the document and the result of a character recognition process is stored, instead of the graphic representation of the text. This character recognition part is based on Walsh transformation. The algorithm was tested in several cases, and proved itself to be pretty efficient and reliable for simple documents.

Keywords. Image data compression, optical character recognition, Walsh transformation
AMS Subject Classification. 68U10 Image Processing

1. INTRODUCTION

One of the basic problems in digital image processing is to use minimal memory to store the data structure which represents the digital image. An algorithm is called compressive if it assigns a data structure P' to the original digital image P so that less memory is required to store P' than P.

Several compressive algorithms are known which are based on different methods. Beside the ordinal data compressors, when we do not care what sort of information the data structure represents, there are some algorithms which were developed especially for compressing digital images (or a special type of digital images).

The efficiency of the compression is usually defined by the value $\frac{|P|}{|P'|}$, where $\mid P \mid$ is the size of the data structure P. An algorithm is more efficient if the compressed data structure reserves less memory than the original image.

In our terminology, the best algorithm is a not necessarily unique one which gives the most efficient compression for a given digital image. Obviously, it is not sure that the best algorithm remains the most efficient one if we consider another image. It is extremely difficult to find out which compressive algorithm is the best for the digital images we are about to process. The authors in the literature propose several algorithms according to the characteristics of the digital images. For example, an algorithm for compressing line drawings can be found in [3], or if you are looking for a survey on compression algorithms, see [2].

A possible way to enhance the efficiency of the compression is to allow information loss. A good example for this kind of compression is JPEG. We can define some distance function between the original and the decompressed digital images to measure the information loss of the compression. For binary images we can obtain a very simple distance function by summing the pixel-to-pixel differences. In this case the maximal distance value appears between the digital image and its inverse. This example illustrates well that information loss should be separated from visual information loss. Intuitively, the latter would reflect the distortion of the visual information.

In this paper we present an information loss compressive algorithm which can be applied mainly to text documents. During the compression our main purpose is to preserve the visual information about the document which is the most important for a human reviewer.

This algorithm can be used successfully in practice, when the main goal is to transmit the compressed document through some telecommunication channel. The original digital images are supposed to contain basically text information, which is recorded in a typographically fixed form without using sophisticated structures. For example, fax documents usually have these properties.

The theoretical scheme of the algorithm can be seen in Figure 1.1.

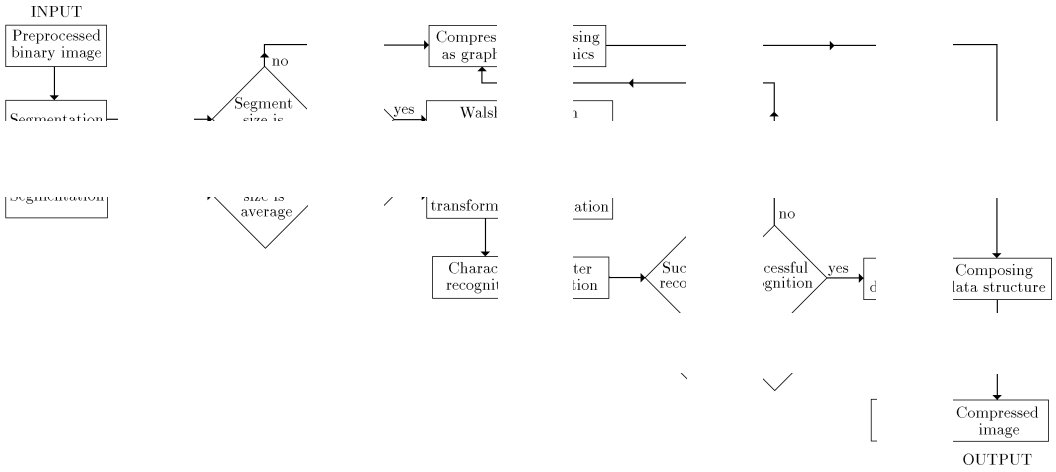

Figure 1.1. Theoretical scheme of the algorithm

A special segmentation algorithm divides the original binary image into smaller ones, and these smaller segments are stored in a chained list. These segments are actually rectangles, and beside the image information, the coordinates of the upper left pixel and the size of the rectangle are also stored for every segment. The detailed description of this segmentation algorithm can be found in Chapter 2. If we know the original image size then the original image can be restored from this data structure with some information loss.

During the processing of the chained list we judge whether a given segment contains text information (letter, number, etc.), or a graphic part of the document. The analysis of the stored image sizes helps us to make this decision: if the segment size is close to the average segment size then we consider it as a character.

We apply a standard graphics compressor algorithm to those elements of the list which contain graphic parts with high probability. The binary images which contain text information, are recognized by a character recognition program. Our character recognition method is described in Chapter 3. If the recognition is successful then we replace the binary image with its character code in the chained list. If the recognition is not successful then we deal with that segment as if it contained a graphic part of the document (and apply the above explained compression algorithm to it). You can find more about character recognition in [4].

This chained list is the actual compressed data structure, and we obviously lose information, since the character attributes disappear during the character recognition step.

The binary image is restored (usually with some information loss) from the compressed data structure in the following way. First we create an empty binary image (in background color) with the size of the original one. The size of the original image can be stored in the head of the list, or can be regarded as fixed. The elements of the list are reconstructed by sequently processing the disjoint parts of the image. If an element is a compressed graphic detail then according to its coordinates it is restored by decompressing. If the element is a recognized symbol then according to the coordinates and size of its storing rectangle, we put a character to the correct position with the given size from a character set. If this character description corresponds to the most often used font style of the documents then the quality of the reconstruction can be enhanced.

Chapter 4 contains the result of our experiments, where we tried to use a character set applied commonly.

2. SEGMENTATION

As we mentioned in the introduction, the first step is to run a segmentation algorithm on the digital image we would like to compress. Our purpose is to determine minimal storing rectangles to those subsets of the image foreground which can be separated by horizontal and vertical lines. The determination of such a rectangle means that we calculate its size and the coordinates of its upper left pixel. Sometimes we will refer to these storing rectangles as segments. It is important to note that it is not the connected foreground components that our segmentation procedure results. The main reason why we choose this type of segmentation method is that we would like to manage ligature appearances in text which depend on the applied character set. In these situations the recognition is obviously unsuccessful, as the picture of the letters change drastically, see Figure 2.2, where a ligature "fi" appears in the Hungarian word "figyelő" (observer). Our algorithm is preconditioned for this phenomenon, and it can be trained to recognize ligatures. In our experiments beside the ligature-free character sets OCR-A and OCR-B (which were composed for optical character recognition) we also used a set which allows ligatures.

Our segmentation algorithm can be made parallel easily, and the whole procedure can be performed as an alternating recursive sequence of horizontal and vertical segmentation steps. The horizontal segmentation steps are followed by vertical segmentation steps and vice versa. With these steps each of the existing segments is divided into smaller segments. The procedure stops if the number of the segments remains the same during a segmentation

step. The following description explains briefly in meta language format the operation of the horizontal segmentation algorithm. The procedure `CutPicture` defines a new segment whose first row will be the row under `SY` and last row will be the row above `EY`. The current element of the list will describe this new segment, so its record fields are changed according to this modification. The new element of the list will store the information about that part of the image which is not processed yet.

```
Type SP=^Segment;
     Segment=Record Of
         X,Y:Word; {*Upper left pixel coordinates*}
       LX,LY:Word; {*Size of the segment*}
        Code:Byte; {*Code of the recognized character*}
     Picture:Pointer; {*The address of the segment*}
        Next:SP;   {*Next element of the chained list*}
     End;
     :
     :
Function HSegmentation(Var Head:SP):Boolean; {*True if a new segment is defined*}
  Var NewHead:SP; {*New segment*}
  Var SY,EY:Word; {*Beginning and end of the segment*}
  Begin
    SY:=NextEmptyLine;
    EY:=NextEmptyLine;
    If (SY=Head^.Y) And (EY=Head^.Y+Head^.LY-1) Then
      HSegmentation:=False; {*Image can not be segmented any more*}
      Exit
    End;
    While (EY-SY<>1) Do Begin {*Ignoring pairs of empty lines*}
      SY:=EY;
      EY:=NextEmptyLine
    End;
    New(NewHead); {*Creating a new element in the list*}
    CutPicture(Head,NewHead,SY,EY); {*Current segment is defined by SY,EY*}
    NewHead^.Next:=Head^.Next;
    Head^.Next:=NewHead;
    Head:=NewHead;
    HSegmentation:=HSegmentation(Head) Or True {*Processing the image part remained*}
  End;
```

Figure 2.1. Description of the segmentation algorithm in meta language format

Horizontal segmentation step: We have a pixel running from left to right, starting from the upper left corner of every segment we already have. If we find an object (foreground) point in the given row then we go down one pixel and start to run a pixel from the beginning of this row. We go on with this procedure till the running pixel reaches the right side of the segment (we find a row which does not contain object points). In this case we obtain a new segment. The top row of the new segment will be the top row of the original segment which contains an object point. The bottom row of the new segment will be the bottom row of the original segment which is already processed and contains an object point. After defining the new segment, we go on with processing the original segment, starting from that row which did not contain object points.

Vertical segmentation step: The vertical segmentation procedure is analogous to the horizontal one, but here the pixel runs from top to bottom, starting from the upper left corner of a segment. We go right one pixel till a column is found which does not contain object points. In this case a new segment is defined.

In the first step of the segmentation procedure we consider the whole original binary image as the initial segment, and first a horizontal segmentation is performed. The segmentation procedure stops if during a horizontal or vertical segmentation step we do not

find a row or a column which does not contain an object point – in other words, new segments cannot be defined.

The steps of the above described segmentation algorithm can be seen in the following figure, where the segments are represented by rectangles. The result of the first (horizontal) step is a text line if we assume that the original binary image is a text document.

Figure 2.2. The result of the segmentation after the second and fourth steps

It is worth noticing that the top of the line is determined by the highest character ("f"), and the bottom is determined by the lowest character ("g"). The second (vertical) segmentation procedure divides the text line into characters, but the rectangles that store the characters contain relatively large background components which can be eliminated with the following (horizontal) segmentation step. The number of the segmentation steps which are necessary to segment a graphic part of the document depends on the complexity of the graphics.

The segmentation of the accented characters is an interesting and difficult procedure. In the case of English text after three segmentation steps (horizontal – vertical – horizontal) the storing rectangles cannot be reduced any more, while in the case of Hungarian text which contains accented characters, we have the same situation only after the fourth segmentation step. Figure 2.2 illustrates this case as well, where the segmentation of the Hungarian accented character "ő" is performed in four steps. The reconstruction of an accented character is quite difficult, since the accent is segmented separately. It does not mean a problem in our algorithm, but it turns to be crucial, when the recognition of the character is important. The recognition can be performed by analyzing the placement of the segments with small size. For example, it can be useful to examine if a vowel takes place under a segment with small size.

The segments which are obtained during a segmentation step are organized into a chained list according to their creation in the recursive segmentation procedure. An element of this list contains the image information of the given segment (as a binary image), the coordinates of the upper left pixel, and the size of the segment. A pointer is also included which points to the following element of the list.

3. COMPRESSION AND CHARACTER RECOGNITION

In our algorithm we use two basically different compression methods. The graphic parts of the document are compressed by an information preserving compression program, while the text information (letters, numbers, etc.) is recognized and the characters are encoded with their code from a character map which obviously means a more efficient, but information loss compression.

Processing an element of the chained list, according to the average size of the stored binary images it can be decided whether the given element stores text information (character) or a graphic part. If it is a character according to its size, but the character recognition is not successful then we consider this segment as a graphic part and compress it that way.

3.1. The Walsh transformation

We use the Walsh transformation in our character recognition process. The Walsh transformation $W(u, v)$ is a separable and symmetric one which has the following form in 2D:

$$W(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\ g(x,y,u,v),$$

where $f(x, y)$ is the intensity of the pixel with the coordinates (x, y) in the original binary image. The size of the image f is $N \times N$, and $u, v = 0,\ \ldots,\ N-1$, thus we compute N^2 Walsh transforms altogether that we can organize into an N^2 dimensional feature vector:

$$(W(0,0),\ W(0,1),\ W(0,2),\ \ldots,\ W(0,N-1),\ W(1,0),\ W(1,1),\ \ldots,\ W(N-1,N-1)).$$

Function g is the so called kernel function and has the following form:

$$g(x,y,u,v) = \frac{1}{N} \prod_{i=0}^{n-1} (-1)^{b_i(x)b_{n-i-1}(u)+b_i(y)b_{n-i-1}(v)},$$

where $b_i(x)$ is the ith bit in the binary expansion of x (so it is equal to 0 or 1), and $N = 2^n$. A more detailed description about the Walsh transformation can be found in [2]. The Walsh transform is unique in the sense that if we consider two different binary images then the corresponding feature vectors are also different. If we compose a feature vector which does not contain all the Walsh transform values then this vector can be the same for two different original binary images, see Section 3.2. The Walsh transformation is separable, as its kernel function can be separated in the following way:

$$g(x,y,u,v) = g_1(x,u)\ g_2(y,v).$$

Moreover, we have the property

$$g(x,y,u,v) = g_1(x,u)\ g_1(y,v),$$

(the factors are functionally equivalent), thus the Walsh transformation is symmetric. These two properties make the computation of the 2D transforms considerably faster, since it can be reduced to the computation of two 1D Walsh transformations, and the symmetric property makes the computation even more faster.

3.2. Application of the Walsh transformation

To perform the Walsh transformation, first we have to magnify the original image to the size of $2^n \times 2^n$ for some n which does not mean a considerable modification, since the Walsh transformation is invariant under magnification. We used the image size 32×32.

However, in our algorithm we compute only the following 64 Walsh transforms instead of the $32 \times 32 = 1024$ ones:

$$(W(0,0),\ W(0,1),\ W(0,2),\ \ldots,\ W(0,7),\ W(1,0),\ W(1,1),\ \ldots,\ W(7,7)).$$

There are two reasons for restricting the number of the Walsh transforms:

- These 64 values describe well the global features, and symmetric relations of the binary image;
- We have the opportunity to filter out some noise from the image, since the computation of less Walsh transforms results a blurring effect.

An example for the latter property can be seen in Figure 3.1, where the Walsh transforms $W(0,0),\ W(0,1),\ W(0,2),\ W(0,3),\ W(1,0),\ W(1,1),\ \ldots,\ W(3,3)$ are equal for the two original 8×8 binary images.

Figure 3.1. Different images with the same feature vector

Magnifying the segments to the same size (32×32) can cause a problem in character recognition. Namely, the characters whose lower and upper cases are similar (e.g. "w" and "W") become identical. In our algorithm it does not mean a problem, and in that case, when the recognition of the character is important, the proper case can be reconstructed by comparing the side lengths of the original storing rectangle.

In the previous section we mentioned that the 2D Walsh transformation can be performed as two 1D transformations. In our algorithm we prescind from this consideration and compute the transforms directly from tables, as this is the fastest way. The value

$$\prod_{i=0}^{n-1} (-1)^{b_i(x)b_{n-i-1}(u)+b_i(y)b_{n-i-1}(v)} \qquad (*)$$

in the kernel function can be ± 1. For example, let us consider the case $n = 1,\ N = 2^1 = 2$. The corresponding table traditionally has the form

Table 3.1. Values of the Walsh transformation kernel function in the case of $N = 2$

(u,v) \ (x,y)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	+	+	+	+
(0,1)	+	−	+	−
(1,0)	+	+	−	−
(1,1)	+	−	−	+

where the cells of the table contain + or − signs according to the value of $(*)$.

3.3. Character recognition – feature and etalon vectors

As it is described in the previous section, we obtain a 64 dimensional feature vector by computing some of the Walsh transforms for an element of the chained list. We compare this feature vector with etalon vectors which contain the same 64 Walsh transforms of etalon characters. The etalon characters are composed as the average of sample characters. For a given feature vector we find the closest etalon vector by using a suitable distance function (for example the Cartesian one). If the minimum distance between a feature vector and the etalon vectors is larger than a threshold value then the correspondent segment is considered as a graphic part of the original image, and not as a character. The algorithm can be trained to be able to recognize ligatures, too. A detailed description how the Walsh transformation is applied to character recognition can be found in [1].

4. CONCLUSION – EXPERIMENTAL RESULTS

We emphasized in the previous chapters that the algorithm gives the best result in the case of text documents. Moreover, if we use well-chosen documents and character set we can achieve very good results. For example, if the same character set is used for composing the document and for restoring it then the restored document perfectly fit (bit by bit) to the original one without any loss of information.

The segmentation procedure also means an efficient compression, since it eliminates (at least when the original image is not affected by noise corruption) the large background components from the compressed data structure.

Our experimental results are summarized in the following table. We used an LZW based compression program to compare our algorithm with the general compression programs. We tested our algorithm for a document which contains only text information and for a document with graphic parts. The entries of the table are given in kilobytes.

Table 4.1. Experimental results – the entries are given in kilobytes

	Original size	LZW compression	Our compression	Our compression together with LZW
Text document	966.8	39.3	76.8	9.1
Graphic document	24	1.2	2	0.7

REFERENCES

[1] A. Fazekas, T. Herendi, *Methods and applications of digital image processing*, Bulletins for Applied Mathematics, 778/91 (1991), pp. 641–662.

[2] R. C. Gonzalez, R. E. Woods, Digital Image Processing, Addison-Wesley, Reading, MA, 1992.

[3] L. Huang, A. Bijaoni, *An efficient image compression algorithm without distortion*, Pattern Recognition Letters, **12** (1991), pp. 69–72.

[4] O.D. Trier, A.K. Jain, T. Taxt, *Feature extraction methods for character recognition - A survey*, Pattern Recognition Letters, **29** (1996), pp. 641–662.