NEIGHBORHOOD CODING FOR BILEVEL IMAGE COMPRESSION AND SHAPE RECOGNITION

Tiago B. A. de Carvalho[1], Denise J. Tenório[1], Tsang Ing Ren[1], George D.C. Cavalcanti[1], Tsang Ing Jyh[2]

[1] Center of Informatics, Federal University of Pernambuco
Recife, PE, Brazil - www.cin.ufpe.br/~viisar
[2]Alcatel-Lucent, Bell Labs, Belgium
[1]{tbac,djt,tir,gdcc}@cin.ufpe.br, [2]ing-jyh.tsang@alcatel-lucent.com

ABSTRACT

Neighborhood coding was proposed to encode binary images. Previously, this coding scheme presented good results in the problem of handwritten character recognition. In this article, we extended this coding scheme so that it can be applied as an image shape descriptor and in a bilevel image compression method. An algorithm to reduce the number of codes needed to reconstruct the image without loss of information is presented. Using the exactly same set of reduced codes, a lossless compression method and a shape recognition system are proposed. The reduced codes are used with Huffman coding and RLE (Run-Length Encoding) to obtain a compression rate comparable to well-known image compression algorithms such as LZW and CCITT Group 4. For the shape recognition task we applied a template matching algorithm to the set of strings generated by the coding reduction procedure. We tested this method in the MPEG-7 Core Experiment Shape 1 part A2 and the binary image compression challenge database.

Index Terms— Image coding, compression, shape analysis, classification, matching.

1. INTRODUCTION

Image representation is a fundamental concept in image processing. Bilevel image representation is the basic form to code image data. We extended the neighborhood coding scheme proposed in [1] to obtain a shape descriptor features and a compression method. In [1-3] the method was used for handwritten digits recognition via neural networks, clustering algorithms and chi-square analysis, respectively. This coding scheme was also used as neighborhood operators in a manner similar to morphological operators [2, 3].

In this paper, we reformulate the neighborhood coding representation reducing the total number of codes necessary to describe an image; consequently a compression scheme is obtained. The same reduced set of codes is also used successfully as features to characterize image shape, which shows to be attractive for the problems related to image retrieval systems.

The structure of this paper is as follows: In Section 2, we describe in details the neighborhood coding, using three different directions for the neighborhood coding. In Section 3, the code reduction algorithm is described. The compression procedure is explained in Section 4. The shape recognition proposal is described in Section 5. In Section 6, we describe the experiments and results of the proposed method, comparing the results to other methods. Finally, in Section 7, we present some discussion and conclusions.

2. NEIGHBORHOOD CODING

In a binary image, the neighborhood coding procedure transforms the image into a set of neighborhood codes that represent each pixel of the image. A neighborhood code is a vector that represents the foreground pixels. Each vector is composed of two parts: the center position and the segment. The center position is the location (x_i, y_i) of a pixel and the segment is defined by the distance of the center pixels to the border of the connected component in a specific direction. We used three types of neighborhood coding: (1) segment in the horizontal and vertical direction, (2) segment in the diagonal direction, and (3) the combination of segments (1) and (2). As an analogy, we can compare these neighborhood types to the moving direction of the chess pieces: (1) tower, (2) bishop and (3) queen, respectively.

The tower coding contains four segments, in the horizontal and vertical direction: (*north*, *east*, *south*, *west*), or (*n*, *e*, *s*, *w*), this generates, for the neighborhood code φ_i, the following sets of positions, respectively: N_i, E_i ,S_i and W_i, where $N_i = \{(x_i, y) \mid y = y_i - k, k = 0, 1, 2, ..., n_i\}$, $E_i = \{(x, y_i) \mid x = x_i + k, k = 0, 1, 2, ..., e_i\}$, $S_i = \{(x_i, y) \mid y = y_i + k, k = 0, 1, 2, ..., s_i\}$, $W_i = \{(x, y_i) \mid x = x_i - k, k = 0, 1, 2, ..., w_i\}$, and (x_i, y_i) is defined as the center position of the pixel in consideration.

In the same way, we can define the bishop coding with the diagonal segments: (*northeast, southeast, southwest, northwest*) or (*ne, se, sw, nw*) generating, for the neighborhood code φ_i, the sets NE_i, SE_i, SW_i and, NW_i, where $NE_i = \{(x,y) \mid x = x_i + k, y = y_i - k, k = 0, 1, 2, ..., ne_i\}$, $SE_i = \{(x,y) \mid x = x_i + k, y = y_i + k, \ k = 0, 1, 2, ..., se_i\}$, $SW_i = \{(x,y) \mid x = x_i - k, y = y_i + k, k = 0, 1, 2, ..., sw_i\}$, $NW_i = \{(x,y) \mid x = x_i - k, y = y_i - k, k = 0, 1, 2, ..., nw_i\}$.

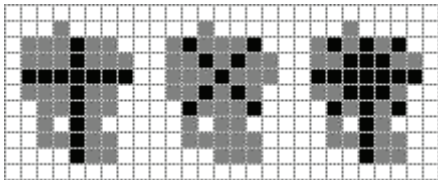

Figure 1. Example of three different type of coding: tower (left), bishop (center) and queen (right). The dark pixels represent the neighbor regions.

The third type of coding, the queen coding is defined as the combination of the tower and the bishop coding, and contains, for the neighborhood code φ_i, the set of pixels: N_i, E_i, S_i, W_i, NE_i, SE_i, SW_i, NW_i. Therefore, tower and bishop coding forms a 6-tuple: (*x, y, n, e, s, w*) and (*x, y, ne, se, sw, nw*) and the queen coding forms a 10-tuple (*x, y, n, e, s, w, ne, se, sw, nw*).

Figure 1 shows an example of each one of these coding, they all have the same central position. On the left the tower coding which can be written as a vector (4, 4, 2, 3, 5, 3). The center image shows the bishop coding that can be described as a vector (4, 4, 2, 2, 2, 2). On the right, the queen coding, represented as a vector (4, 4, 2, 3, 5, 3, 2, 2, 2, 2).

3. CODE REDUCTION

Given a binary image, the neighborhood vector coding can be used to represent the whole image. However, the generated set of codes can be extremely redundant, since the information contained in one code can be already coded in another vector. This redundancy can be eliminated consequently creating a more compact representation of the image. The problem is how to obtain the subset of codes that are sufficient to reconstruct the binary image without loss of information. For this task, we proposed an algorithm to select the codes that have the most relevant information:

1. Create C a variable that measures if all pixels have been encoded. C starts as C = m, where m is the number of foreground pixels of the image.
2. Create $\Phi = \{\varphi_i \mid i = 1, \ldots, m\}$, where φ_i is the neighborhood code $(x_i, y_i, n_i, e_i, s_i, w_i)$ that codes the pixels in $P_i = \{(x, y) \mid (x, y) \in N_i \cup E_i \cup S_i \cup W_i\}$.
3. Create $A = \{a_i \mid i = 1, \ldots, m\}$, where a_i is a flag, $a_i = 0$ if the center (x_i, y_i) of φ_i is not yet encoded and $a_i = 1$ otherwise. A starts as A = 0 (no center has been encoded).
4. Create the output array $O = \{o_i \mid i = 1, \ldots, m\}$, where o_i is a flag, $o_i = 1$ if the code φ_i has been selected and $o_i = 0$ otherwise. O starts as O = 0 (no code has been selected yet).
5. Create $T = \{t_i \mid i = 1, \ldots, m\}$, where t_i starts as $t_i = n_i + e_i + s_i + w_i$.
6. While C > 0 do *// have all pixels been encoded?*
   Choose $\varphi_l \mid t_l \geq t_v, \forall \varphi_l, \varphi_v \in \Phi$ *// code with maximum t*
   $C = C - t_l$
   $o_l = 1$
   For each $\varphi_k \in V_l \subseteq \Phi$ with $a_k = 0$, where V_l is the set of all neighborhood codes with center $(x, y) \in P_l$ do
      $a_k = 1$
      For each $\varphi_w \in V_k \subseteq \Phi$ do $t_w = t_w - 1$ *// update T*
7. Return O

The value t indicates the coding capacity of the neighborhood vector, which means the total number of pixels that the code is able to represent. Initially t is defined as the summation of all the coding segments, for the tower coding $t = n + e + s + w$, for the bishop coding $t = ne + se + so + no$ and for the queen coding $t = n + l + s + o + ne + se + so + no$. Even though, the update of the coding capacity t is an exhaustive task, it does the algorithm convergence faster and assures that the final set of selected codes has low redundancy. At the end of the process just the relevant codes remain.

The reduction code algorithm starts with the initialization of the variables (step 1-5). Following, all the pixels are transformed into the neighborhood coding format (step 2). The code with biggest t value is chosen, and each of the affect codes, that are codes that represent pixels that were associated with the selected code, is identified. The value of t for each affected code is updated by subtracting the number of pixels that it does not code anymore (step 6). Finally the procedure returns the output, the selected codes (step 7).

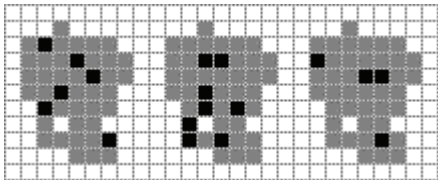

Figure 2. The selected code centers are shown in black. The images show the results using tower (left), bishop (center) and queen (right) coding.

Figure 2, shows the results of the code reduction process. The center of each selected code is shown in black. The complete image is shown in gray. The left image shows the tower coding result, the center image shows the bishop coding and the right image the queen coding. After the coding reduction, for the same image 6 neighborhood vectors codes are necessary to represent the whole image using the tower coding, 9 codes for the bishop coding and 4 for the queen coding.

The code reduction method aims to obtain the least number of neighborhood vector codes necessary to represent the whole image without loss of information. In this way, it is possible to obtain a very compact representation of the image.

4. COMPRESSION METHOD

Once the reduced codes are obtained the rest of the compression method is straightforward. The process can be divided in 5 stages: (1) coding reduction; (2) grouping of the codes that have the same vector segments; (3) applying run-length encoding (RLE) [4] for the vector segments; (4) applying RLE for the run counts of the first RLE and (5) Huffman coding. The code reduction consists in generating the neighborhood coding as described in the last section. The next step is to group the codes according to the segments. The vector segment of a code is defined as the code vector without the central position value, x and y. For the tower coding the code segment is represented by (*n, e, s, w*), for the bishop coding (*ne, se, sw, nw*), and for the queen coding (*n, e, s, w, ne, se, sw, nw*). This grouping is important to obtain a maximum information reduction using the RLE procedure. The list of codes is divided in two parts: vector segment and center position, both listed in an equivalent order. The next step of the process is to apply RLE to the list of vector segment, resulting in a list of run counts and run values, the run counts indicates the number of times each run value repeats. In this specific case, the run counts represent the number of times each vector segment repeats in the total coding procedure. Since there are many repeated run counts, RLE is applied to the run count of the first RLE. Finally, the results are stored in a file using Huffman coding. The decompression procedure is also straightforward; the original image can be obtained simply by applying the reverse procedure.

5. SHAPE RECOGNITION

Shape descriptors can be used to compare the outlines of objects in an image to determine their similarity. They are important and useful for applications such as image database retrieval systems. An indication of the importance of such descriptors is the fact that the MPEG-7 group incorporates such shape descriptors into the MPEG-7 standard [5].

One neighborhood code describes not just a few pixels but part of one connected component of the image. Thus neighborhood

coding can be used as shape descriptor. The method used for shape recognition is obtained by calculating a distance measure between two strings of neighborhood codes. A string is defined as the set of selected codes, obtained through the code reduction algorithm, in the order they were selected, as explained in Section 3. One string is generated per object.

Using the template matching procedure, the distance measured between these two strings is defined as the minimum cost of matching the strings. This measure is also known as *Levenstein Distance* as described in [6]. Given two strings ***t*** (test string) and ***r*** (template string), the *Levenstein Distance* calculates the minimum cost to transform ***t*** into ***r*** through insertions, deletions or changes of items of ***t***. For the tower coding, the cost of inserting a neighborhood code φ_i from ***r*** into ***t*** is equals to $n_i^2 + e_i^2 + s_i^2 + w_i^2$. The cost of deleting a neighborhood code φ_j from ***t*** is equals to $n_j^2 + e_j^2 + s_j^2 + w_j^2$. And the cost for changing a neighborhood code φ_i from ***r*** to another code φ_j from ***t*** is equals to $2 \times (x_i - x_j)^2 + 2 \times (y_i - y_j)^2 + (n_i - n_j)^2 + (e_i - e_j)^2 + (s_i - s_j)^2 + (w_i - w_j)^2$. The template matching runs over all possible combinations of symbols in order to obtain ***t*** from ***r***.

The *Levenstein Distance* between the strings is inversely proportional to the similarity of the objects. In this way, a queried image is associated to the class of images with lowest distance.

6. EXPERIMENTS AND RESULTS

The experiments are divided in three parts: code reduction, compression and shape recognition. For the code reduction and compression experiments a set of selected binary images were used. Figure 3 shows the images used in the experiments. The images *0, 1, 2, 3, 4, 5, 6, 7, 8* and *9* were obtained using an Arial font with centered window size of 128x128 pixels. The *bat* and *bell* images are from the MPEG-7 Core Experiment Shape 1 part A2 [5]. The images *courier12, oldeng16, ouster, times12i* and *cat* were obtained from the binary image compression challenge [7]. *ouster* and *cat* are halftone images and *courier12*, *oldeng16* and *times12i* are short texts written in Courier, Old English and Times New Roman having size 12, 16 and 12, respectively. The images in Figure 3 are not in the original size and the text images are partially shown.

Figure 3. Test images compilation. The text images are partially shown and all the images are reduced from its original size

6.1. Code Reduction

Table 1 shows the dimensions of the images, the total number of neighborhood codes before code reduction (Total) and the number of codes after reduction using Tower, Bishop and Queen coding. The images *0* through *9*, *bat-12* and *bell-2* are all solid images, the code reduction over these images, using Tower coding, reduced the number of codes to approximately 3%. The Bishop coding achieve greater reduction than Tower coding just for the halftone images. The Queen coding achieve the best reduction in all cases because it codes as much as using Tower and Bishop coding together. Even though the Queen coding generates less number of codes, it takes a lot more storage space since each code is associated with a longer vector segment.

Table 1. The table shows the image dimension (width × height), the total number of codes required before reduction and after reduction using Tower, Bishop and Queen coding.

Image	Dimensions	Total	Tower	Bishop	Queen
0	128×128	1,349	39	88	38
1	128×129	756	16	62	12
2	128×130	1,313	50	75	33
3	128×131	1,222	59	79	47
4	128×132	1,328	33	83	23
5	128×133	1,376	48	98	41
6	128×134	1,524	57	96	50
7	128×135	910	33	69	24
8	128×136	1,536	60	87	54
9	128×137	1,498	54	93	48
bat	474×216	58,903	328	807	303
bell	59×64	1,558	38	150	35
cat	380×469	73,145	50,420	15,712	12,008
courier12	374×46	1,111	314	717	233
oldeng16	476×55	3,515	628	1,201	447
ouster	108×144	6,880	2,324	1,835	1,494
times12i	278×46	1,179	435	618	314

6.2. Compression

To evaluate the proposed method, we compared the results of neighborhood coding compression with other image file formats that have special algorithms for binary image compression. Table 2 shows the size of the compressed image files using TIFF, GIF, JBIG and the proposed method using Tower, Bishop and Queen coding. The files were obtained using ImageMagick [8], optimizing the parameters for best lossless compression result. The compression algorithm used in the GIF was LZW, in the TIFF format was CCITT Group 4, which is also used in the transmission of facsimiles [4]. The best results were obtained using JBIG format, the state of the art in binary image compression.

The best results using the proposed methods are in boldface. For the solids images the Tower coding achieved the greater compression, for halftone images the Bishop coding was more suitable and the compression using Queen coding had better results on text images.

Although the proposed method did not surpass the compression rate of JBIG, it showed a compression rate similar to well established compression algorithms (Group 4 and LZW).

6.3. Shape Recognition

The database used in the test experiments was the MPEG-7 Core Experiment Shape 1 part A2 [5]. A subset of this dataset A2 contains 420 images, of those 70 are original images, having 5 rotations (9°, 36°, 45°, 90° and 150°) for each of the original images. Therefore the problem of the A2 database is to recognize the independent rotation that the images were submitted to. Two examples of these images are shown in Figure 3, *bat* and *bell* images.

Table 2. Size in bytes of the images compressed through Group 4 used in the TIFF file format, LZW used in the GIF, JBIG and the proposed method using Tower, Bishop and Queen coding.

Img	TIFF	GIF	JBIG	Tower	Bishop	Queen
0	439	425	183	**277**	428	364
1	405	345	135	143	268	**141**
2	433	362	174	337	401	**326**
3	451	386	200	**384**	433	428
4	419	386	150	**251**	415	262
5	439	377	187	**328**	501	392
6	455	427	209	**377**	493	453
7	407	327	142	254	344	**246**
8	455	417	194	**389**	473	479
9	457	427	199	**360**	499	437
bat	817	1,433	404	**2,521**	4,483	3,406
bell	433	254	148	**232**	571	295
cat	69,781	10,540	6,310	115,158	**41,350**	43,056
cou.	979	823	412	977	1,712	**920**
old.	1,599	1,600	797	2,207	3,157	**2,013**
ous.	4,335	1,889	1,231	5,565	**4,865**	4,920
tim.	1,083	794	426	1,317	1,573	**1,179**

Each image from the dataset passed through a pre-processing phase before the extraction of the neighborhood codes. This pre-processing consist in the correction of the image rotation by moving them to a canonical position, which is obtained by rotating the images around its center of mass, and making the horizontal direction the one with the largest variance. Principal Component Analysis (PCA) is applied on the (x, y) positions of the foreground pixels to identify the direction of larger variance.

After the images rotations have been adjusted via PCA, a set of reduced codes are extracted using the Tower coding. The distances between individual codes are compared by template matching, following the procedure described in Section 5. For each of the 420 images, the six closest (lowest *Levenstein* distance) images are obtained. The correct retrieval for each of the 420 images is measured in the following way: for each tested image in the database, the sixth closest images are retrieved, and the number of correct images is defined as the total number of images correctly retrieved. The maximum number of correct image is 2520 (= 6 x 420). The score of correct answers is defined by *"100 x number of correct images / 2520"*.

Table 3. Recognition performance of the proposed method against five others shape descriptor used in MPEG-7 [5].

P298	P320	P517	P567	P687	Proposed
100.00	99.37	100.00	97.46	99.60	99.52

For this database, we obtained 99,52% of correct assigned images. This result is comparable to proprietary algorithms that are used in the MPEG-7 standard, as described in [5]. Table 3 shows the results for the shape recognition method comparing to other techniques.

7. DISCUSSIONS AND CONCLUSIONS

There are three main contributions presented in this paper: the neighborhood code reduction scheme, a lossless binary images compression method and a shape recognition method. The last two contributions are consequence of the first coding reduction scheme. The reduction of codes significantly decrease the number of codes needed to reconstruct the image without losing information of the original image. For that, the neighborhood codes proposed in [1] had to be modified, adding the center position (x, y) of the code.

The compression rates achieved by the proposed method are approximately the same as those obtained by well established compression algorithms such as Group 4 used for transmission of facsimiles and TIFF format and LZW used in the GIF image file format. Even though, the proposed method does not exceed the state-of-the-art in binary images compression JBIG. Some experiments were also performed, using arithmetic coding instead of Huffman coding to improve the compression rate.

The shape recognition procedure proposed here also showed comparable results to the best algorithms used to evaluate the MPEG-7 Core Experiment Shape 1 part A2 database. For future work, we plan to apply this encoding method to other databases that is related to the problem of shape recognition, such as handwritten characters recognition. Also, other feature extraction technique from the reduced set of codes can be used to obtain faster comparison systems. In conclusion, we demonstrated that the neighborhood coding is an interesting method to represent an image and it can be applied in various types of image processing problems.

8. ACKNOWLEDGEMENT

This work was partially supported by FACEPE and CNPQ.

9. REFERENCES

[1] I. J. Tsang, I.R. Tsang, and D. Van Dyck, "Image coding using neighbourhood relations," Pattern Recognition Letters 20, pp. 1279-1286, 1999.

[2] I. R. Tsang and I. J. Tsang, "Neighbourhood Vector as Shape Parameter for Pattern Recognition," WCCI World Congress on Computational Intelligence - IJCNN 2006, pp. 3204 – 3209, 2006.

[3] I. R. Tsang, and I. J. Tsang, "Pattern Recognition Using Neighborhood Coding," Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 600-611, 2006.

[4] J.D. Murray, and W. Van Ryper, "Encyclopedia of Graphics File Formats", 2nd Edition. O'Reilly & Associates, 1996.

[5] L. J. Latecki, R. Lakämper, and U. Eckhardt, "Shape Descriptors for Non-rigid Shapes with a Single Closed Contour," IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp. 424-429, 2000.

[6] S. Theodoridis, and K. Koutroumbas, "Pattern recognition 3rd ed," Elsevier, USA, 2006.

[7] Binary image compression challenge, http://wiki.tcl.tk/12314, 2008.

[8] ImageMagick, http://www.imagemagick.org/, 2009.