Pakistan Journal of Information and Technology 2(1): 61-64, 2003

Compressing Video Using Matrix Folding

[1]Abdelfatah A. Yahya, [1]Ayman M. Abdalla and [2]Khaled Eisa Al-Qawasmi
[1]Faculty of Science, Al-Zaytoonah University of Jordan, Amman, Jordan 11733
[2]Faculty of Information Technology, Al-Neelain University, Khartoom, Sudan

Abstract: The main purpose of commonly used MPEG format, among many other video formats, is to reduce the size of video. We present a new algorithm designed to further reduce the size of MPEG files and it can be applied to several other types of video or non-video files. This algorithm applies matrix folding to the files to reduce their size by half and it is easily reversible without any degradation of picture quality. The algorithm is fast enough to be used in real time with some video systems such as NTSC. We present experimental results that demonstrate the efficiency of our algorithm.

Key Words: Video Compression, Matrix Folding. CR Categories: H.3 - Information Storage and Retrievaleducation, I.4 - Image Processing and Computer Vision

Introduction

Video files require large storage space and big transmission bandwidth. The Moving Picture Experts Group (MPEG) format (Wiseman, 2002) is commonly used in video compression due to its efficiency in saving storage space and transmission time. Other common formats include ASF, MOV, and AVI. The MPEG movie-format is based on the JPEG (Joint Photographic Expert Group) standard (Wallace, 1991), which is used for single images. JPEG can achieve a powerful compression ratio such as 100:1. However, the ratio should remain between 10:1 and 20:1 to avoid noticeable degradation. Furthermore, JPEG is not suited for line art, cartoons, or one colored blocks.

At the beginning of JPEG encoding, the image is divided into blocks of 8×8 pixels and converted from RGB (Red, Green, Blue) format into YUV (luminance and chrominance) format. The YUV format represents the image using 24 bits per pixel: 8 bits support a monochrome picture, and 8 bits contain color information (Data compression reference center, 1997-2000) . These YUV values are coded by DCT using the equation:

$$(i, j) = c(i, j)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} f(m, n)\cos\frac{\pi(2m+1)i}{2N}\cos\frac{\pi(2n+1)j}{2N}.$$

The DCT coefficients are uniformly quantized, to achieve further compression, by reducing precision down to the desired image quality. Finally, each block is converted into a one-dimensional array that maximizes the probability of occurrence for successive identical values, and Huffman encoding is applied to the array to reduce redundancy.

The main idea of MPEG compression lies in removing the spatial redundancy from video frames and the temporal redundancy between them (Lombardo *et al.*, 2001). Different MPEG extensions, such as MPEG-1, MPEG-2, MPEG-3 and MPEG-4, were developed to target specific applications (Data compression reference center, 1997-2000). We concentrated our experiments on MPEG-2 since its basic concepts are the same as other MPEG formats. In addition, MPEG-2 was designed to be a generic video coding system supporting a diverse range of application, including transmission over ISDN networks (Furth *et al.,* 1995 and Tudor, 1995).

To simplify error handling and editing, MPEG implements video streams in a hierarchy of layers, where the video-stream layer is a sequence of frames, and every frame is coded in a manner similar to JPEG coding explained above.

MPEG consists of a sequence of three frames: I-frames, P-frames, and B-frames (Hoffman and Fernando, 1996). I-frames are coded using only information present in the picture itself, in order to provide potential random access points in the compressed video sequence. The coding is based on the discrete cosine transform according to the JPEG coding technique (Lombardo *et al.,* 2001), and it is intra coded to allow reconstruction without reference to other frames (MPEG, 2002). P-frames are coded using a similar coding algorithm to I-frames, but with the addition of motion compensation with respect to the previous I- or P-frame. Because of this forward prediction, I-frames cannot be reconstructed without reference to other frames (Lombardo *et al.,* 2001 and MPEG, 2002). B-frames are coded with motion compensation with respect to the previous I- or P-frame, the next I- or P- frame, or an interpolation between them (Wiseman, 2002 and Lombardo *et al.,* 2001) .

In this paper, we present a new no-degradation algorithm designed to reduce the size of MPEG files, among several other video and non-video formats. We also present experimental results that demonstrate the efficiency of our algorithm.

The Matrix Folding Algorithm: Various media types such as audio, pictures, animation and video need a huge amount of storage space and require long transmission time. In this section, we present a folding algorithm that reduces the size of video files, thus saving storage space and transmission time.

Consider MPEG format as an example. Each video in MPEG format is a one-dimensional array (vector) of Huffman code, which is a compressed version of a given video. We apply our folding algorithm to this compressed-format vector. The vector values that implement the pixel brightness for MPEG-2 range from 0 to 255, where 0 represents "no color" and 255 represents "full color."

The matrix folding operation merges each two elements in the MPEG vector into one element using the equation:

$m_i = 256\ e_i + e_{(n-i+1)}$,

Where:

m_i is the merged element and

$e_1, e_2, ..., e_n$ are the brightness values in the Huffman vector.

For each element in the first half of the movie vector, the algorithm multiplies the element e_i by 256 and then adds the result to element $e_{(n-i+1)}$. This multiplication operation may be replaced with an 8-bit shift to the left and the addition operation may be replaced with a bit-wise-OR operation. The results arê stored in the new vector, *m*, whose length is half the length of the original vector. To obtain the original vector, *e*, from vector *m*, use the unfolding equations:

$e_i = m_i / 256,$
$e_{(n-i+1)} = m_i \text{ MOD } 256,$

where the above two equations produce the integer quotient and remainder, respectively, of dividing an element from vector *m* by 256. These two operations may be replaced with equivalent bitwise operations. It can be mathematically shown that these two equations produce the original values of vector *e*, thus showing that this method produces no loss of information.

Since folding and unfolding perform a constant number of arithmetic operations for each entry in the input vector, the time required for folding and unfolding should remain constant for each entry. Therefore, each folding or unfolding operation will be linear in time, *i.e.*, *O*(*n*), with respect to the number of entries, *n*, in the input vector.

The savings in size result from using the extra space available in each entry in vector *v*. Since MPEG and many other video-formats use 16 bits to represent each entry, we use eight bits to store the color information and the extra eight bits are used for folding. This method was successful in several video formats, as demonstrated by our experiments in the next section. Furthermore, this compression method produces no degradation in image quality since there is no loss of information from the original (MPEG or other format) file.

Implementation: In this section, we present experimental results obtained by implementing the folding algorithm to video files of different types and various sizes. We executed our program using MATLAB 6.1 under Windows XP on a Pentium-III PC with a 600-MHz CPU and a 256-Mbyte RAM. We used an MPEG converter to read MPEG files, frame-by-frame, into a matrix that can be read by MATLAB, where the folding algorithm was applied and the result was stored in a binary file. Other video formats were handled similarly. When it is needed to view an original video, the binary file is unfolded and the resulting matrix is automatically produced in MPEG (or other video format). As Table 1 illustrates, folding reduced the size of different video files by half the original size. The growth in folding and unfolding times, illustrated in Figure 1, were linear with respect to the size of the original video. The unfolding time per frame was approximately constant, as seen in Fig. 2 and depended on the size of the frame. Generally, it required approximately 2 microseconds per byte to unfold each video file. The time required for folding and unfolding our test videos is shown in Table 1. Since the unfolding time is usually faster than 30 frames per second, it is possible to perform unfolding in real time for some video systems such as NTSC. The sizes of the video files after folding were exactly the same as their original sizes, with no loss of data or change in quality.

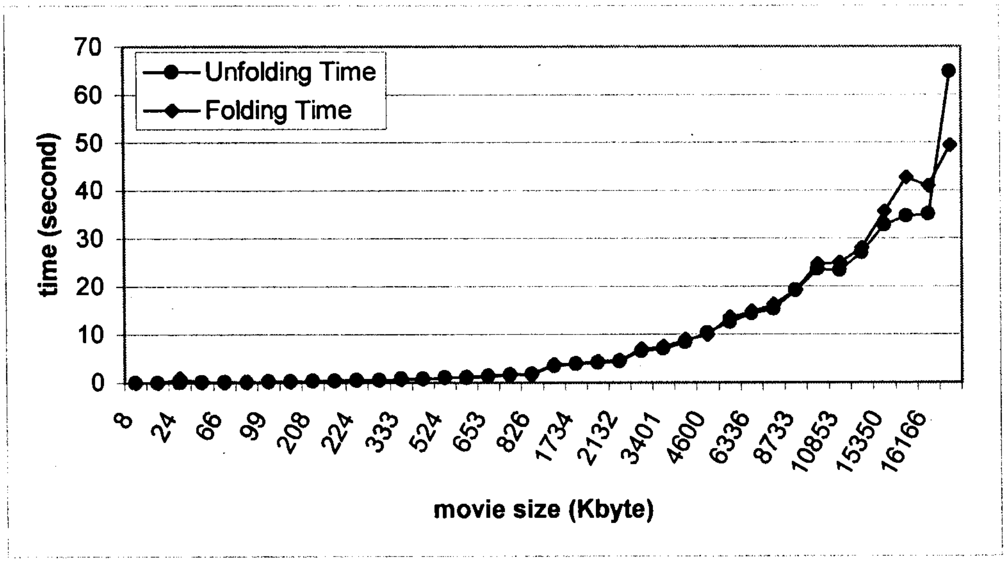

Fig. 1: Time (in seconds) Needed to Fold / Unfold Files of Different Sizes

Table 1: Sample Results Obtained by Applying our Folding Algorithm

Sample	File Type	File Size (byte)	Folding Size (byte)	Folding Time (second)	Unfolding Time (second)
1	mmm	8,000	4,000	0.01	0.02
2	AVI	13,312	6,656	0.03	0.03
3	mpg	24,188	12,094	0.931	0.06
4	mpg	52,764	26,382	0.11	0.111
5	mpg	67,508	33,754	0.15	0.13
6	mpg	96,363	48,182	0.2	0.191
7	AVI	101,630	50,815	0.221	0.21
8	mpg	129,951	64,976	0.29	0.25
9	mpg	213,381	106,691	0.441	0.431
10	AVI	224,768	112,384	0.471	0.45
11	mpeg	228,933	114,467	0.48	0.47
12	mpeg	244,400	122,200	0.581	0.491
13	AVI	340,776	170,388	0.721	0.801
14	wmv	391,000	181,000	0.742	0.771
15	mpg	536,841	268,421	1.131	1.062
16	AVI	549,154	274,577	1.192	1.122
17	mpeg	668,536	334,268	1.412	1.332
18	ASF	813,823	406,912	1.743	1.612
19	AVI	845,986	422,993	1.803	1.723
20	ASF	1,752,382	876,191	3.705	3.485
21	AVI	1,775,570	887,785	3.935	3.866
22	AVI	2,010,380	1,005,190	4.407	4.106
23	ASF	2,182,720	1,091,360	4.606	4.336
24	ASF	3,263,544	1,631,772	6.89	6.48
25	AVI	3,482,568	1,741,284	7.521	7.12
26	AVI	4,108,480	2,054,240	8.872	8.372
27	ASF	4,710,278	2,355,139	9.924	10.385
28	ASF	6,189,826	3,094,913	13.569	12.568
29	AVI	6,487,998	3,243,499	14.812	14.37
30	ASF	7,634,354	3,817,177	16.273	15.392
31	ASF	8,942,752	4,471,376	19.228	19.258
32	ASF	10,982,812	5,491,406	24.655	23.654
33	ASF	11,113,891	5,556,946	24.996	23.284
34	AVI	12,694,186	6,347,093	27.96	27.059
35	ASF	15,718,521	7,859,261	35.632	32.857
36	ASF	16,395,698	8,197,849	42.642	34.65
37	AVI	16,554,220	8,277,110	40.879	35.051
38	ASF	19,052,858	9,526,429	49.371	64.863

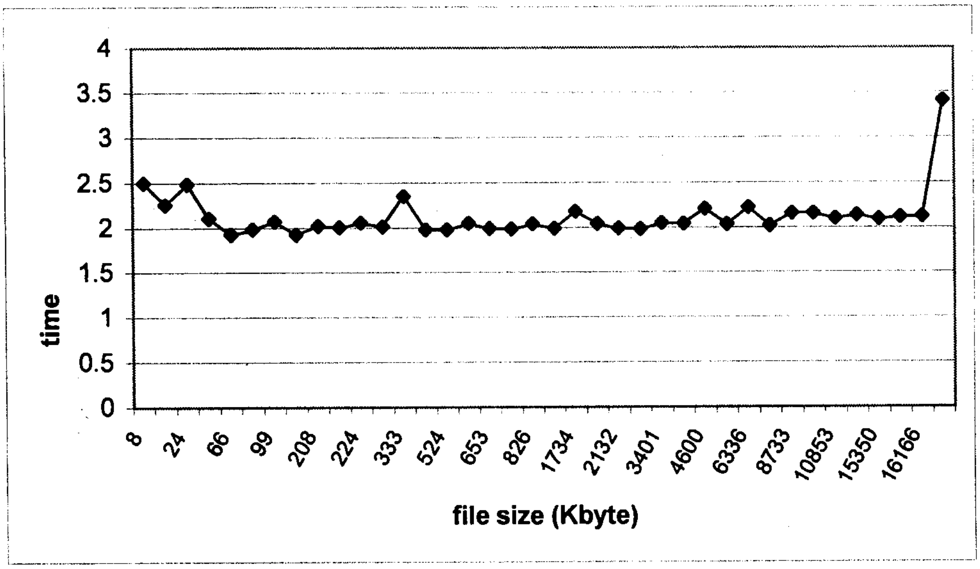

Fig. 2: Time (microseconds) Per Byte Needed to Unfold Files of Different Sizes

Conclusion

We presented a new algorithm that reduces the size of files of different types. Our experiments show a reduction of 50%, where the results were obtained in a time suited for real-time retrieval on some systems.

References

Data Compression Reference Center, 1997-2000. URL: www.rasip.fer.hr\research\compress\index.html. Accessed 14-May-2002.

Furth, B., S. W. Smoliar and H. Zhang, 1995. Video and image processing in multimedia Sys. Kluwer Academic Publishers, London.

Hoffman, D. and G. Fernando, 1996. RTP Payload Format for MPEG1/MPEG2 Video, Sun Microsystems, Inc. V. Goyal, Precept Software, Inc.

Lombardo, A., G. Morabito; S. Palazzo and G. Schembra, 2001. A Markov-Based Algorithm for the Generation of MPEG Sequences Matching Intra- and Inter-GoP Correlation. *European Transactions on Telecommunications*, Vol. 12, No.2.

MPEG Video Compression Technique, 2002. URL: http://tan.informatik.tu-chemnitz.de/~jan/MPEG/HTML/MPEG_tech.html. Accessed 14-May-2002.

Tudor, P. N., 1995. MPEG-2 Video Compression, IEEE J.

Wallace, G. K., 1991. The JPEG Still Picture Compression Standard. In IEEE Transactions on Consumer Electronics.

Wiseman, J., 2002 An introduction to MPEG video compression, URL: http://members.aol.com/symbandgrl/. Accessed 14-May-2002.