2D-Pattern Matching Image and Video Compression

Preliminary Results

November 7, 1998

Marc Alzina
ENST
46 Rue Barrault
75013 Paris
FRANCE
alzina@email.enst.fr

Wojciech Szpankowski*
Dept. of Computer Science
Purdue University
W. Lafayette, IN 47907
U.S.A.
spa@cs.purdue.edu

Ananth Grama†
Dept. of Computer Science
Purdue University
W. Lafayette, IN 47907
U.S.A.
ayg@cs.purdue.edu

Abstract

In this paper, we present a lossy data compression scheme based on an approximate two dimensional pattern matching (2D-PMIC) extension of the Lempel-Ziv lossless scheme. We apply the scheme to image and video compression and report on our theoretical and experimental results. Theoretically, we show that the so called *fixed database model* leads to suboptimal compression. Furthermore, the compression ratio of this model is as low as the generalized entropy that we define in the paper. We use this model for our video compression scheme and present experimental results. For image compression we use a *growing database* model for which we provide an approximate analysis (which will be presented in the final version of the paper). The implementation of 2D-PMIC is a challenging problem from the algorithmic point of view. We use a range of novel techniques and data structures such as k-d trees, generalized run length coding, adaptive arithmetic coding, and variable and adaptive maximum distortion level to achieve good compression ratios at high compression speeds. We demonstrate bit rates in the range of $0.25 - 0.5$ bpp for high quality images and data rates in the range of $0.15 - 0.4$ Mbps for video compression.

1 Introduction

Lossless data compression based on *exact* pattern matching can be traced back to seminal papers of Lempel and Ziv [18, 19]. More recently, there has been a resurgence of interest in extending these schemes to lossy data compression (cf. [1, 2, 3, 8, 11, 12, 13, 15, 16, 17]). This can be largely attributed to the rapid growth in *digital* representation of multimedia (e.g., text, audio, image, video, etc.) which is particularly amenable to exact and *approximate* pattern matching manipulations.

We believe that the first practical implementation of lossy image compression based on approximate pattern matching (i.e., a combination of vector quantization and lossy LZ78) was reported by Constantinescu and Storer [3]. Unfortunately, no theoretical justification was provided for their scheme and experimental results were largely preliminary. Thus a natural question arose whether such a scheme is asymptotically optimal, like its lossless Lempel-Ziv counterpart. This problem was first addressed by Steinberg and Gutman [15], and eventually resolved by Łuczak and Szpankowski [12, 13] and also by Yang and Kieffer [17]. They proved that a straightforward lossy extension of Lempel-Ziv scheme is not asymptotically optimal. It was demonstrated that the compression bit rate asymptotically achieves the so called *generalized Shannon entropy* which is higher than the optimal rate distortion function. Recently, Kontoyiannis [11] proposed a scheme based on approximate pattern matching that is asymptotically optimal for memoryless sources, however, it is not yet clear if this would lead to an implementable solution. Finally, we should mention the asymptotically optimal *golden-washing scheme* proposed by Zhang and Wei [20] which does not seem to work very well in practice.[1]

†This work was supported by NSF Grants EIA-9805743 and EIA-9806741.
*This work was supported by NSF Grants NCR-9415491 and C-CR-9804760.
[1]This comes from a conversation one of the authors had with Dr. V. Wei.

The central theme of a lossy extension of Lempel-Ziv algorithm is the notion of approximate repetitiveness. If a portion of data recurs in an approximate sense, then subsequent occurrences can be stored as direct or indirect references to the first occurrence. This approximate recurrence of data may not be contiguous. Somewhat surprisingly, this theme of exploiting approximate repetitiveness is uniformly applicable to multimedia data from various sources such as text, images, video, and even audio. However, approximate repetitiveness can be hidden in various forms for different types of media. Therefore, one must consider different distortion measures. This is in contrast with current state-of-the-art, where a different approach is used for compressing each of these types of data.

Using the theoretical underpinning of Łuczak and Szpankowski [13], Atallah, Génin and Szpankowski [2] implemented a lossy scheme for image compression. This scheme, called *Pattern Matching Image Compression* (PMIC), was based on *one*-dimensional pattern matching enhanced with some salient features. It was shown that for high quality images PMIC scheme is competitive with JPEG and wavelet image compression up to 1 bpp. However, the superiority of PMIC for decompression (PMIC does not require any computation since it basically only reads data) may turn out to be a crucial advantage in practice where asymmetric compression/decompression is a desirable feature (e.g., video). *However*, a bit rate of 1 bpp is not that interesting from a practical point of view since most image compression applications require bit rates of 0.5 bpp to 0.25 bpp. In this paper, we describe a new implementation based on two-dimensional approximate pattern matching that achieves compression of up to 0.25 bpp for images while preserving other desirable properties of PMIC. We call this scheme 2D-PMIC. We also use this idea to develop a new compression scheme for video data that uses the previous (coded) frame in the group of frames (GOP) as a database. Our preliminary experimental results exceed even our expectations: we obtain high quality video with bit rates of $0.15 - 0.4$ Mbps (without the first frame). We report on our theoretical and experimental findings in this paper. The reader is referred to `http://www.cs.purdue.edu/people/spa` for detailed results and demonstrations.

In this conference paper we first discuss the theoretical underpinnings of our scheme and subsequently present algorithmic and experimental results. Our scheme for video compression is modeled quite well by the *fixed database model*. In this model, the previous frame serves as a fixed database (reference) for compressing the next frame using approximate pattern matching. For this model, we report a theoretical asymptotic bit rate which turns out to be the generalized Shannon entropy. While the generalized Shannon entropy is higher than the optimal rate distortion function, we observe that at least for the memoryless model these two rates do not differ by much (cf. [13]). Our scheme for compressing image data can be modeled using a *growing database* formed by the continually changing reference corresponding to segments of the image that have been compressed. We are currently working on analyzing this model and will report on an approximate analysis in the final version of this paper.

The implementation of 2D-PMIC is a challenging problem from the algorithmic standpoint. Finding an efficient data structure for *approximate* search of multidimensional sets in a huge multidimensional database, is an interesting problem in itself. We use a set of modified k-d trees enhanced by generalized run length coding for approximate search. A key issue for high quality image and video compression is the design of an adaptive distortion measure that automatically adjusts its maximum distortion to produce perceptually high quality results. We propose a few solutions and examine their performance in this paper.

2 Theoretical Underpinnings

Consider a source sequence $\{X_k\}_{k=1}^{\infty}$ taking values from a finite alphabet $\mathcal{A}$ (e.g., $|\mathcal{A}| = 256$). That is, $X_i : \mathcal{S} \to \mathcal{A}$ where $\mathcal{S}$ is a two-dimensional area (e.g., for images $\mathcal{S}$ is an $N \times N$ square of pixels). We write $X_m^n = X_m X_{m+1} \ldots X_n$, and $P(x_1^n)$ for the probability of $X_1^n = x_1^n$. We encode X_1^n into a compression code c_n, and the decoder produces an estimate $\hat{X}_1^n$ of X_1^n. More precisely, code c_n is a function $\phi: \mathcal{A}^n \to \{0,1\}^*$, thus, $c_n = \phi(x_1^n)$, where lower-case letters represent realizations of a stochastic process. On the decoding side, the decoder function $\psi: \{0,1\}^* \to \mathcal{A}^n$ is applied to find $\hat{x}_1^n = \psi(c_n)$. Let $\ell(c_n)$ be the length of a code (in bits) representing x_1^n. Then, the *bit rate* is defined as $r(x_1^n) = \ell(c_n)/n$, the *average* bit rate as $\mathbf{E}[r(X_1^n)] = \mathbf{E}[\ell(C_n)]/n$, and the *compression ratio* CR as $CR = n \log_2 |\mathcal{A}|/\ell(c_n)$.

We consider *single-letter fidelity* distortion measures $d: \mathcal{A} \times \hat{\mathcal{A}} \to \mathbb{R}_+$ such that

$$d(x_1^n, \hat{x}_1^n) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \hat{x}_i) \ ,$$

which is assumed to be subadditive, that is, for any two integers n, m, and given vectors x^{n+m}, y^{n+m}

$$d(x^{n+m}, y^{n+m}) \leq \frac{n}{n+m} d(x^n, y^n) + \frac{m}{n+m} d(x_{n+1}^{n+m}, y_{n+1}^{n+m}) \ .$$

Examples of fidelity measures satisfying the above criteria include the *Hamming distance* and the *square error distortion*. In Hamming distance, $d(x_i, \hat{x}_i)$ equals zero if $x_i = \hat{x}_i$ and one otherwise; and in square error distortion, $d(x_i, \hat{x}_i) = (x_i - \hat{x}_i)^2$. Our original PMIC was based on square error distortion, which is natural for image compression. In the current implementation we develop several novel distortion measures that automatically adjust their maximum value to produce the best possible perceptual results.

2.1 Fixed Database Model

Our implementation of pattern matching video compression is modeled well by the fixed database scheme. In this scheme, the decoder and the encoder both have access to the common database sequence $\hat{X}_1^n$ generated according to the distribution $\hat{P}$. The source sequence X_1^M (where $M = N^2$ for $N \times N$ images) is partitioned according to Π_n into variable length phrases $Z^1, Z^2, \ldots, Z^{|\Pi_n|}$ of lengths $L_n^1, \ldots, L_n^{|\Pi_n|}$, respectively. Here, for given fixed $D > 0$

$$\begin{aligned} L_n^1 &= \max\{k: \ d(\hat{X}_i^{i+k-1}, X_1^k) \leq D, \ \ 1 \leq i \leq n-k+1\}, \\ Z^1 &= X_1^{L_n^1}. \end{aligned}$$

This implies that the first partition comprises of the longest prefix of the input X_1^M that matches a substring in the database $\hat{X}_1^n$ to the specified tolerance. The string $\hat{Z}^1$ recovered by the decoder is therefore given by:

$$\hat{Z}^1 = \hat{X}_i^{i+L_n^1-1},$$

Subsequent partitions are computed in a similar manner, that is, the $(m+1)$-st phrase of partition Π_n is define as

$$\begin{aligned} L_n^{m+1} &= \max\{k: \ d(\hat{X}_i^{i+k-1}, X_{K_n(m)+1}^{K_n(m)+k}) \leq D, \ \ 1 \leq i \leq n-k+1\}, \\ Z^{m+1} &= X_{K_n(m)}^{K_n(m)+L_n^m-1}, \\ \hat{Z}^{m+1} &= \hat{X}_i^{i+L_n^m-1}. \end{aligned}$$

where $K_n(m) = \sum_{i=1}^m L_n^i$.

Thus, the source sequence is partitioned according to Π_n as $X_1^M = Z^1 Z^2 \ldots Z^{|\Pi_n|}$ while the decoder recovers the string $\hat{Z}^1 \hat{Z}^2 \ldots \hat{Z}^{|\Pi_n|}$ which is within distortion D from X_1^M. This follows from the subadditive property of the distortion measure. We represent each $\hat{Z}^i$ by a pointer `ptr` to the database and its length (recall that $\hat{Z}^i$ is defined over a two-dimensional domain), hence its description costs $\log n + \Theta(L_n^i)$ bits. The total compression code length is

$$\ell_n(X_1^n) = \sum_{i=1}^{|\Pi_n|} \log n + \Theta(\log L_n^i).$$

Thus, the bit rate (in bits per pixel) is given by

$$r_n(X_1^n) = \frac{1}{M} \sum_{i=1}^{|\Pi_n|} \log n + \Theta(\log L_n^i). \tag{1}$$

To formulate our main result, we introduce the generalized Shannon entropy $\hat{r}_0(D)$ defined as

$$\hat{r}_0(D) = \lim_{n\to\infty} \frac{\mathbf{E}_P[-\log \hat{P}(B_D(X_1^n))]}{n}, \tag{2}$$

where $B_D(\hat{x}_1^n) = \{y_1^n : \ d(\hat{x}_1^n, y_1^n) \le D\}$ is the ball of radius D and center x_1^n, and $\mathbf{E}_P$ denotes the expectation with respect to P. In the final version of this paper, we prove the following result.

Theorem 1 *Let us consider the fixed database model with the database $\hat{X}_1^n$ generated by a Markovian source $\hat{P}$ and the source sequence X_1^n emitted by a Markovian source P.*
(i) *The length L_n^1 of the first phrase satisfies*

$$\lim_{n\to\infty} \frac{L_n^1}{\log n} = \frac{1}{\hat{r}_0(D)}, \qquad \text{(pr.)}.$$

More precisely, for any $\varepsilon > 0$

$$\Pr\left\{(1-\varepsilon)\frac{1}{\hat{r}_0(D)}\log n \le L_n^1 \le (1+\varepsilon)\frac{1}{\hat{r}_0(D)}\log n\right\} = 1 - O\left(\frac{\log n}{n^\varepsilon}\right). \tag{3}$$

(ii) *The average bit rate attains asymptotically*

$$\lim_{n\to\infty} \lim_{M\to\infty} \mathbf{E}_P[r_n(X_1^M)] = \hat{r}_0(D). \tag{4}$$

Sketch of Proof. Part (3) follows from [13]. The second part (4) can be derived using the approach of [16], and [11]. ∎

Remark. We can extend the analysis to mixing sequences $\{\hat{X}_k\}$ and $\{X_k\}$ such that the process $(\hat{X}, X)$ is stationary.

2.2 Growing Database Model

In contrast to video compression where a certain reference frame was used as the database, image compression uses a partially compressed image as a database. This can be modeled effectively using a *growing database model*. Let us assume that the source sequence $\{X_k\}_{k=1}^{\infty}$ is generated according to distribution P. We set the initial reproduction sequence of length m to be the same as the first m symbols of the database, that is, $X_1^m = \hat{X}_1^m$. We reveal X_1^m to the decoder (e.g., in a lossless transmission).

Set $\hat{X}^0 = X_1^m$ and let $n_0 = m$. The algorithm works as follows:

- Find the first match of length $L^1_{n_0}$ as follows

$$L^1_{n_0} = \max\{t: \ d(\hat{X}_i^{i+t-1}, X_{n_0}^{n_0+t-1}) \le D, \quad 1 \le i \le n_0 - t\}.$$

  Encode pointer $\mathtt{ptr}(1) = i$ and the length $L^1_{n_0}$. Then we enlarge the database

$$\hat{X}^1 = \hat{X}^0 \star \hat{X}_{\mathtt{ptr}(1)}^{\mathtt{ptr}(1)+L^1_{n_0}}.$$

  where $\star$ means concatenation.

- Provided kth reproduction sequence $\hat{X}^k$ of length $n_k = \hat{X}^k$ was constructed, define the $(k+1)$-st match length as

$$L^{k+1}_{n_k} = \max\{t: \ d(\hat{X}_i^{i+t-1}, X_{n_k}^{n_k+t-1}) \le D, \quad 1 \le i \le n_k - t\},$$

  and set $\mathtt{ptr}(k+1) = i$. The corresponding code is $(\mathtt{ptr}(k+1), L^{k+1}_{n_k})$. The enlarged database now becomes

$$\hat{X}^{k+1} = \hat{X}^k \star \hat{X}_{\mathtt{ptr}(k)}^{\mathtt{ptr}(k)+L^{k+1}_{n_k}}.$$

The *operational bit rate* at step k of the algorithm is given by

$$r^k(X_1^{n_k}) = \frac{\log n_0 + \log n_1 + \cdots + \log n_k + \Theta(\log L^1_{n_0}) + \cdots + \Theta(\log L^{k+1}_{n_k})}{n_{k+1}}.$$

The analysis of $r^k(X_1^{n_k})$ is currently beyond our reach since the distorted database $\hat{X}^k$ is non-stationary of unknown distribution. Nevertheless, in the final version of the paper we shall provide an approximate analysis of $r^k(X_1^{n_k})$.

3 Algorithms, Data Structures and Implementation

In this section, we describe algorithmic and implementation issues related to the compression scheme. The decompression scheme works in a similar manner but in the reverse order. We should emphasize, however, that the decompression scheme based on pattern matching is very simple, fast, and does not require any computation (unless we use an arithmetic decoder). This is crucial for possible WWW-based applications if one aims, as we do, to display image and video "on-the-fly" while downloading. This is particularly important for video applications since current schemes (e.g., AVI, QuickTime, MPEG) require significant software support on the decompression side.

The rest of this section addresses distortion level design, growing heuristic, pixel encoding, lossless encoding and video encoding.

3.1 Variable Distortion

In the original PMIC (cf. [2]) we used square error distortion uniformly over an image, that is, D was fixed. We call this *uniform distortion*, and set D to be equal to RandomNoise to indicate that lossy compression can be viewed as adding "noise" to the image. In the original PMIC, we computed the distortion–per–symbol for a block (averaged over the block). We observed experimentally that uniform distortion leads to significant perceptual errors. To avoid this, in 2D-PMIC we adopted another approach in which we assign to every *individual* pixel a *range* that depends on the values of surrounding pixels. The main idea is to be less tolerant (i.e., have a small range) in slowly varying regions (low frequencies-only) and more tolerant in quickly varying regions (high frequencies). We propose here three new solutions, namely: local distortion range, global distortion range and distortion range for graphical images.

In this conference version we only describe the above two methods. In the *local distortion range*, we introduce differences between an individual pixel value and its surrounding neighbors. The range assigned to the pixel at (x, y) is given by:

$$\text{Range}{=}\text{RandomNoise} + \text{LocalNoise} \times \text{LocalVariation}$$

where $D = \text{RandomNoise}$ is the maximum distortion assigned in the uniform distortion method, LocalNoise is additional distortion, while LocalVariation is the maximum of the absolute value of the difference between the pixel at (x, y) and its 9 neighbors. The drawback of this method is that it deals only with the 9 closest pixels and consequently it is very sensitive to noise in the original image. To counter ill effects of the locality of the previous method, in the global distortion range we introduce a function that assigns a large range only when there are few pixels of similar value around it. We refer to our journal version for details.

3.2 Growing Heuristic

Let us assume that a part of an image has been previously encoded. The next step is to encode the pixels located on the right and below using a point that we call the *anchor point*. The selection of the next anchor point is called *growing heuristic* (cf. [3]). The growing heuristic we adopt is the "wave–front": from the top to the bottom of the image. Other examples of growing heuristics are a circle that expands outward from the center, thickening the main diagonal, etc. (cf. [3]). These heuristics give very similar results.

3.3 Pixel Encoding

The growing heuristic supplies *anchor points* that refer to the area to be encoded. Our scheme has there encoding modes: Enhanced Run-Length-Encoding (ERLE), two-dimensional pattern matching, and if neither of these work, we leave the area uncompressed, i.e., we encode the pixels in their original form. We discuss the first two methods in depth below.

3.3.1 Enhenced RLE

Experimental observations reveal that natural images do not contain too many areas with constant pixel values. Nevertheless, areas containing slowing varying pixel values are more likely to occur. We exploit this in the enhanced run-length encoding (RLE). Consider the encoding of an $m \times n$ rectangle. Let g_i be the value of a a pixel located at (x_i, y_i). The goal is to find a plane $z = ax+by+c$ that minimizes

$$f(a,b,c) = \sum_{i=0}^{mn-1} (ax_i + by_i + c - g_i)^2,$$

that is, we apply the least square-error optimization. To find the coefficients a, b, c we need to solve

$$\begin{pmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum y_i x_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & mn \end{pmatrix} \times \begin{pmatrix} a_0 \\ b_0 \\ c_0 \end{pmatrix} = \begin{pmatrix} \sum x_i g_i \\ \sum y_i g_i \\ \sum g_i \end{pmatrix}$$

subject to

$$\begin{array}{llll} \sum x_i^2 & = n \sum_0^{m-1} i^2 & \sum y_i^2 & = m \sum_0^{n-1} i^2, \\ \sum x_i & = n \sum_0^{m-1} i & \sum y_i & = m \sum_0^{n-1} i, \\ & & \multicolumn{2}{l}{\sum x_i y_i = \sum_0^{m-1} i \sum_0^{n-1} j.} \end{array}$$

This gives a_0, b_0, c_0. In practice, we store the variables c_0, c_x, c_y defined as $c_x = ma_0$ and $c_y = nb_0$. Furthermore, we subtract from c_0 the pixel value that is to the left of the anchor point. This increases the probability of c_0 being close to 0, which gives a better lossless compression for c_0.

We must also determine the best dimensions $m \times n$ of the rectangle on which we apply ERLE. The algorithm used is the following:

1. Initialize x to an arbitrary constant, say $\simeq 2$.
2. Test if the rectangle anchored at the *anchor point* of size $x \times 1$ can be encoded using ERLE with an acceptable quality loss.
3. While step 2 returns a positive answer, double x. When the answer turns out to be negative, x is a maximum for X.
4. Let X be the largest x for which we can encode the rectangle $x \times 1$. Find X by dichotomy.
5. Do the same thing for Y.
6. If $X < Y$, swap the role played by the x-coordinate and the y-coordinate in the next step.
7. For $y = 2$ to $Y - 1$ find by dichotomy the greatest x so that the rectangle $x \times y$ can be ERL encoded.

When running this algorithm, we compare the cost (number of bits used to store the ERLE parameters per pixel encoded) of the current solution and the previous one, and select the better one. Notice that the number of bits used is calculated according to the distribution model used by the lossless compressor. This important fact allows us to save some memory during the lossless encoding phase.

3.3.2 Two-Dimensional Pattern Matching

Two dimensional pattern matching is the most efficient way to compress images among the methods we use in 2D-PMIC. The main idea is to find a two-dimensional region in the uncompressed part of the image that occurs approximately in the compressed part (i.e., database), and to store pointers to it. Observe that we search for the *largest* such an area, hence a brute force search algorithm is too time consuming. Indeed, if the database has $n \times n$ pixels and the as-yet uncompressed region contains $m \times m$ pixels, then the complexity of the algorithm is $O(n^2 m^2)$. This is too expensive since we have to apply it approximately $O(N^2/\log^2 N)$ times on an $N \times N$ image. Furthermore, we should observe that we do *not* match patterns exactly, rather in an approximate sense. Consequently, we need a *range search* not an exact search. To speed up the search process, we use k-d trees (cf. [5]) in a manner similar to the one suggested in [3]. We describe some salient features of our implementation here.

Every pixel of the database is associated with other pixels of its neighborhood taken in a particular order defined by a *template* which forms a pixel vector (e.g., in Figure 1 the vector associated with *P0* according to the template 3×2 of dimension $D = 6$ is $(P0, P1, P2, P3, P4, P5)$). The vectors are then stored in a k-d tree (cf. Figure 2). If we search for a D-dimensional vector (template), then the search is done in a k-D tree, and the test whether to go left or right (as in binary search trees) is performed on the $(d \bmod D)$-th coordinate, where d is the depth of the node. The terminal nodes are called *buckets* since they contain one or several leaves. Leaf nodes store the data such as the vector and the coordinates of the associated pixels.

To insert a leaf L, we start from the root and follow the path until we find a bucket B. There are three possibilities:

- B is empty, then we put L into it.

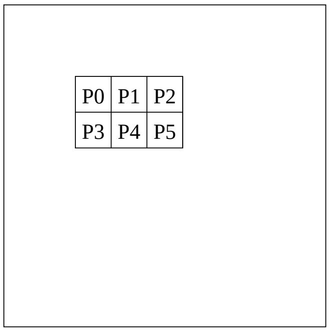

Figure 1: Vector defined by a pixel and a template.

- the depth of this branch is already equal to the maximum depth *MaxDepth*. Then we simply add L to the bucket which is actually a chain of leaves.
- the depth of this branch is less than the limit. In this case, we replace B by a node whose children are B and an empty bucket. We then continue the insertion process.

The search needs two arguments: a *minimum* vector $(m_1, m_2, \cdots, m_D)$ and a *maximum* vector $(M_1, M_2, \cdots, M_D)$. The aim of the search is to find every vector $(v_1, v_2, \cdots, v_D)$ in the k-d tree such that $m_1 \leq v_1 \leq M_1 \ldots m_D \leq v_D \leq M_D$. This is simply done recursively by keeping every subtree that may contain a solution and discarding the others.

In order to find all interesting solutions, we actually use several k-d trees, each one associated with a different template. Furthermore, to decrease the number of potentially non-interesting solutions, pixels and their associated vectors are grouped by location (by rectangles whose typical size is 30×30) and one k-d tree is used per location, so that we can perform a local search.

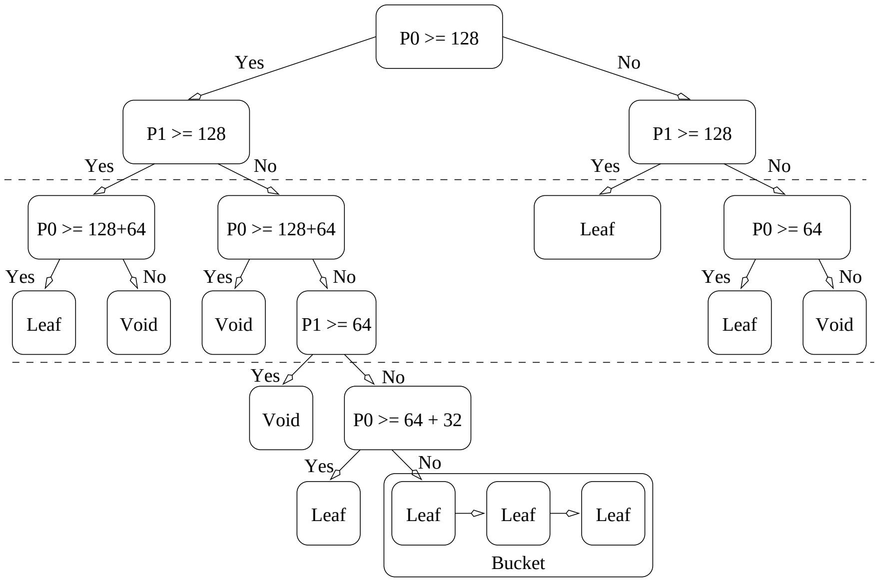

Figure 2: A k-d-tree (Dimension=2, Range $0 - 255$)

Our implementation of the k-d tree methods follows the idea of [3]. The data structure used to represent a k-d tree is a set of two arrays: A_n the array for nodes, and A_l the array for leaves.

Image #	BPP	CR	RMSE	PSNR
2	0.09	84.9	20.6	21.86
3	0.11	73.9	23.6	20.68
2	0.57	14.0	16.8	23.64
3	0.36	22.3	19.3	22.43

Table 1: Video "Train" with and without model pre-filtering

Details will be provided in the final version of the paper.

Finally, the procedure to determine the largest two-dimensional match works as follows: for each template, extract the pixel vector associated to the anchor point and search for similar ones in the k-d trees containing the pixel vectors of the neighboring locations. For each solution, check if it has already been found by searching with another template. If not, apply the same algorithm as for ERLE to find the best match shape. If the number of solutions exceeds a preset limit, we abort the search.

3.4 Lossless Encoding

The pixel encoding phase produces a stream of instructions and parameters (e.g., Pixel_Color (128); Match_Pointer(35,-40);...). Since the probability that a given instruction occurs is not the same for every instruction, and as the distribution of the parameter values is not uniform, we can save memory using lossless compression. To make this compression efficient, a distribution model is built for each parameter according to the values it has previously taken, and the stream is encoded with arithmetic coding. Since we want this compression method to be suitable for WWW-based applications, the instruction/parameters are multiplexed in the same arithmetic coding stream, so that the image/video can be displayed on-the-fly while downloading.

3.5 Video Implementation

Pattern matching video compression is done using the first frame in GOP as the database. In order to facilitate the computation of motion vectors from one image to the next, we take the uncompressed version of the image used as a database, so that the error introduced during its compression does not contribute to the pattern matching algorithm. Of course, during the decompression process, the compressed version of the database is used.

However, on applying this idea directly to video compression, we noticed that the matched patterns were often quite small leading to low compression ratios. This is a consequence of applying distortion (i.e., approximate pattern matching) to *every* individual pixel instead of averaging over the matched area. To circumvent this problem, we apply pattern matching to a low-pass filtered version of the image. In principle: to decide whether or not two areas are alike, we compare their filtered versions. If we find a good match in the filtered image, we apply it to the non-filtered image. Preliminary results indicate that for some samples of video for which the compression did not work (compression ratio $\simeq 4$) this method could improve performance enormously (e.g., $CR \simeq 50$ for "train" video) without noticeable loss in quality. Table 1 compares the compression with and without the low-pass filter.

The low-pass filtered image is computed using a convolution matrix defined as follows:

$$C = \begin{pmatrix} c_{-1,-1} & c_{-1,0} & c_{-1,+1} \\ c_{0,-1} & c_{0,0} & c_{0,+1} \\ c_{+1,-1} & c_{+1,0} & c_{+1,+1} \end{pmatrix}.$$

2D-PMIC				JPEG			
BPP	CR	RMSE	PSNR	BPP	CR	RMSE	PSNR
0.25	32.58	17.0	23.5	0.25	31.90	15.5	24.3
0.50	16.09	13.1	25.8	0.50	15.91	10.6	27.6
1.05	7.59	6.7	31.6	1.00	8.02	6.8	31.5
2.03	3.95	2.9	38.8	2.01	3.98	3.5	37.2

Table 2: Compression results for the image “San Fracisco”.

2D-PMIC				JPEG			
BPP	CR	RMSE	PSNR	BPP	CR	RMSE	PSNR
0.27	29.56	21.0	21.7	0.25	32.31	19.3	22.4
0.51	15.58	12.6	26.2	0.50	16.17	12.5	26.2
0.96	8.33	0.0	Inf	1.00	7.95	6.9	31.4
				2.01	4.08	2.6	39.7

Table 3: Compression results for the image “Basselope”.

That is, the new pixel values become

$$P_{i,j} \leftarrow \sum_{-1\le k,l\le 1} P_{i+k,j+l}\, c_{k,l}.$$

The filters we have tried are:

- A low-pass filter with the frequency response $\mid ((1+\cos(2\pi fx)) \times (1+\cos(2\pi fy)) \mid /4$, $0 \le fx \le 1/20 \le y \le 1/2$. The convolution matrix is:

$$C = \frac{1}{4}\begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}.$$

- A low-pass filter with the frequency response $\mid ((1+2\cos(2\pi fx)) \times (1+2\cos(2\pi fy))) \mid /9$, $0 \le fx \le 1/20 \le fy \le 1/2$. The convolution matrix in this case is:

$$C = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

- The same as above, but with matrix C of dimension 5.

We observed that the second filter turns out to be the best.

4 Experimental Results

In this section, we present detailed experimental results for image and video compression. Pattern matching image compression is compared to JPEG compression in Table 2– 3, while preliminary video compression results are presented in Table 4.

For image compression we conclude that 2D-PMIC is very competitive with JPEG and wavelet compression up to 0.25 bpp. In fact, as Table 3 and Figure 3 suggest, for graphical images 2D-PMIC is superior to both JPEG and wavelets compressions. For natural images, transform-based methods are still better for low bit rates.

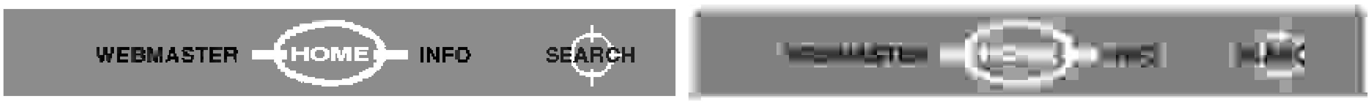

Figure 3: Banner 0.29 bpp, 2DPMIC vs. JPEG

Video	Mbps	PSNR
Claire_hq	0.14	33.7
Claire_lq	0.07	31.0
Ping-Pong_hq	0.40	24.0
Ping-Pong_lq	0.26	22.6

Table 4: Video compression results: low quality (lq) and high quality (hq).

In Table 4 we present preliminary results of video compression for two video clips, namely "Claire" and "Ping_Pong". We compress these using 2-D PMIC with first frame of GOP as the database. The results presented do not include the first frame and do not make use of any predication. A demo is set up on `http://www.cs.purdue.edu/people/spa` for the reader's evaluation. Table 4 presents results for "low-quality" (lq) and "high-quality" (hq) compression. Needless to say, the results are very promising especially since the decompression technique is so simple that can be transmitted with the data.

In conclusion, we remark that:

- PMIC is slower in the compression time but the fastest and simplest possible in the decompression time;
- Overall quality of compression is very good and competitive with JPEG and wavelet based compression up to 0.25 bpp. PMIC works significantly better on graphical images while transform-based methods work better on natural images. In general, PMIC works better when high frequency components dominate the image;
- PMIC naturally adapts to progressive transmissions,
- 2D-PMIC video is particularly promising, and we are exploring it actively as a part of our ongoing research.

References

[1] D. Arnaud and W. Szpankowski, Pattern Matching Image Compression with Prediction Loop: Preliminary Experimental Results, *Proc. Data Compression Conference*, Snowbird 1997; also Purdue University, CSD-TR-96-069, 1996.

[2] M. Atallah, Y. Génin, and W. Szpankowski, A Pattern Matching Image Compression, *Proc. International Conference on Image Processing*, vol. II. 349-352, Laussane 1996; extended version see Purdue University, CSD-TR-95-083, 1996.

[3] C. Constantinescu and J. A. Storer, Improved Techniques for Single-Pass Adaptive Vector Quantization, *Proc. IEEE*, 82, 933-939, 1994

[4] W. Finamore, M. Carvalho, and J. Kieffer, Lossy Compression with the Lempel-Ziv Algorithm, *11th Brasilian Telecommunication Conference*, 141-146, 1993.

[5] J. H.Friedman, J. Bentley and R. Finkel, An Algorithm for Finding Best Matches in Logarithmic Expected Time, *ACM Trans. Mathematical Software*, 3, 209–226, 1977.

[6] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*, Kluwer, 1992.

[7] J. Gibson, T. Berger, T. Lookabaugh, R. Baker *Multimedia Compression: Applications & Standards*, Morgan Kaufmann Publishers 1998.

[8] P. G. Howard, Lossless and Lossy Compression of Text Images by Soft Pattern Matching, *Proc. Data Compression Conference*, 210-219, Snowbird 1996.

[9] P. G. Howard and J. Vitter, Practical Implementations of Arithmetic Coding, Brown University, Department of Computer Science, TR- 92–18, revised version, April 1992.

[10] P. Howard and J. Vitter, Analysis of Arithmetic Coding for Data Compression, Brown University, Department of Computer Science, *Proc. Data Compression Conference*, 3-12, Snowbird 1991.

[11] I. Kontoyiannis, An Implementable Lossy Version of the lempel-Ziv Algorithm– Part I: Optimality for Memoryless Sources, preprint 1998.

[12] T. Łuczak and W. Szpankowski, A Lossy Data Compression Based on String Matching: Preliminary Analysis and Suboptimal Algorithms, *Proc. Combinatorial Pattern Matching*, 102-112, Asilomar 1994.

[13] T. Łuczak and W. Szpankowski, A Suboptimal Lossy Data Compression Based in Approximate Pattern Matching, *IEEE Trans. Information Theory*, 43, 1439–1451, 1997.

[14] M. Rabbani and P. Jones, *Digital Image Compression Techniques*, SPIE Optical Engineering Press, Bellingham 1991.

[15] Y. Steinberg and M. Gutman, An Algorithm for Source Coding Subject to a Fidelity Criterion, Based on String Matching, *IEEE Trans. Information Theory*, 39, 877-886, 1993.

[16] A. Wyner and J. Ziv, The Sliding Window Lempel-Ziv algorithm Is Asymptotically Optimal, *Proc. IEEE*, 82, 872-877, 1994.

[17] E.H. Yang, and J. Kieffer, On the Performance of Data Compression Algorithms Based upon String Matching, *IEEE Trans. Information Theory*, 44, 47-65, 1998.

[18] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Information Theory*, 23, 3, 337-343, 1977.

[19] J. Ziv and A. Lempel, Compression of Individual Sequences via Variable-rate Coding, *IEEE Trans. Information Theory*, 24, 530-536, 1978.

[20] Z. Zhang and V. Wei, An On-Line Universal Lossy Data Compression Algorithm via Continuous Codebook Refinement – Part I: Basic Results, *IEEE Trans. Information Theory*, 42, 803-821, 1996.