PATTERN MATCHING IMAGE COMPRESSION WITH PREDICATION LOOP: Preliminary Experimental Results

Denis Arnaud
ENST
46 Rue Barrault
75013 Paris
France
`darnaud@email.enst.fr`

Wojciech Szpankowski[1]
Department of Computer Science
Purdue University
W. Lafayette, IN 47907
U.S.A.
`spa@cs.purdue.edu`

Abstract

Recently, a novel image compression technique based on pattern matching was proposed, namely *Pattern Matching Image Compression* (PMIC). Basically, it is a lossy extension of the well known Lempel-Ziv scheme. It was proved that such an extension leads to a suboptimal compression, and that the compression ratio can as low as the so called Rényi entropy. Success of PMIC crucially depends on several enhancements such as searching for reverse approximate matching, recognizing substrings in images that are additively shifted versions of each other, introducing a variable and adaptive maximum distortion level, and so forth. In this paper, we introduce another enhancement, namely, predictive coding. More precisely, we implement *Differential Predictive Code Modulation* (DPCM) within PMIC scheme. We report here some preliminary experimental results which show that PMIC enhanced by DPCM can improve compression ratio for good quality images as well as speed of compression for PMIC.

[1]This work was supported by NSF Grants NCR-9415491 and CCR-9201078.

1 Introduction

Since nowdays most media are represented in a digital form, it is natural to apply a technique that was proved successful for inherently digital media such as text, namely Lempel-Ziv type schemes (cf. [17, 18]). In [8, 9] we propose to extend Lempel-Ziv approach to lossy (approximate) compression. While it is know that the lossless Lempel-Ziv is asymptotically optimal (i.e., its compression ratio is close to the entropy), in [9] Luczak and Szpankowski (cf. also [16]) proved recently that a lossy extension of Lempel-Ziv scheme (of low complexity) is suboptimal, that is, it attains the compression ratio equal to the so called generalized Rényi entropy (instead of the optimal rate-distortion).

Using this theoretical underpinning, Atallah, genin and Szpankowski [2] have recently implemented the above idea for image compression. This novel scheme is called *Pattern Matching Image Compression* (PMIC), and it is briefly review in the next section (see also [4] for another implementation of a lossy Lempel-Ziv'78 scheme, however, there is no theoretical justifications for the reported results). In [2] it was concluded that for images of good quality PMIC scheme is competitive to JPEG and wavelet image compression. Superiority of PMIC at decompression (which does not require any computations 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., off-line video).

The central theme of above approach is the notion of approximate repetitiveness. That is, if a portion of data almost-occurs five times, perhaps in close proximity but not necessarily contiguously, then we need only store the first such occurrence: The other four can be stored as (direct or indirect) references to the first occurrence. Somewhat surprisingly, this theme of exploiting approximate repetitiveness is uniformly useful to multimedia data, hence applies to text as well as to image, video, and audio data. However, the approximate repetitiveness can be hidden in various forms for different types of media (so one must consider different distortion measures). This is in contrast with current technology, where a different approach is used for each of these types of data (e.g., text and images). We plan to explore it in our future research.

In [2] several *enhancements* were introduced to the basic idea of PMIC that were instrumental for achieving good results. For example: searching for reverse approximate matching, recognizing substrings in images that are additively shifted versions of each other, introducing a variable and adaptive maximum distortion level, and so forth. These enhancements are crucial to the overall quality of our scheme, and their efficient implementation leads to algorithmic results of interest in their own right. In this paper, we introduce one more enhancement, namely prediction that – as we shall see – will lead to shorter compression time and better compression ratio without significantly deterioration of image quality. Actually, we propose a more ambitious plan, namely implementing DPCM (*i.e.*, Differential Predictive Code Modulation) within PMIC.

To see possible advantage of this new enhancement, we should observe that differential image (composed of the difference between the original image and its prediction) is much less correlated than the original one. Thus, any compression algorithm should gain in performance when applied on the differential image, instead of on the original one. However, for lossy compression there is a need to perform a *prediction loop* as explained in details in Section 3. In DPCM, this prediction loop is performed on each pixel, one by one. Thus, the implementation of a predictive feature for such a scheme as PMIC needs a few adaptations, because PMIC treats several pixels at a time, and does not work on a pixel-by-pixel basis.

The main idea to use a predictive loop in PMIC is that what is the basic cell for DPCM, namely one pixel, becomes the longest prefix found in the database for PMIC. As for DPCM, the differential image is quantized, thus allowing a slight increase in the compression ratio. Besides, although a differential image is significantly less correlated than the original image, it is still slightly correlated. This is because the low-order, linear predictor typically used here or in DPCM is suboptimal. But, while a Huffman coding scheme would not take advantage of this remaining correlation, since it encodes each differential value independently of its neighboring values, the PMIC scheme does, because it is a context based scheme. Thus, PMIC can achieve a bit rate potentially smaller than the zeroth-oreder entropy of the differential image. We report in this paper our implementation and preliminary experimental results.

2 Review of PMIC

This section is based on Atallah *et al.* [2]. The reader might also want to consult [8, 9, 13, 16].

The detection of the above-mentioned approximate repetitiveness is done by approximate pattern matching, whence the name *Pattern Matching Image Compression* (PMIC). The main idea behind it is a lossy extension of the Lempel-Ziv data compression scheme in which one searches for the longest prefix of an uncompressed file that *approximately* occurs in the already processed file. The distance measure subtending the notion of approximate occurrence can be the *Hamming distance*, or of the *square error distortion*, or the *string edit* distance, or any of the other distances considered in the distortion theory (cf. [3]).

In short, let us consider a stationary and ergodic sequence $\{X_k\}_{k=1}^{\infty}$ taking values in a finite alphabet $\mathcal{A}$ (think of $\{X_k\}$ as an image scanned row by row). For image compression the alphabet $\mathcal{A}$ has size $|\mathcal{A}| = 256$. We write X_m^n to denote $X_m X_{m+1} \ldots X_n$, and for simplicity $X^n = X_1 \ldots X_n$. The substring X_1^n will be called *database sequence* or *training sequence*. 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. As a fidelity measure, we consider any distortion function $d(\cdot,\cdot)$ that is subadditive, and that is a *single-letter fidelity* measures, that is, such that $d(x^n, \hat{x}^n) = \frac{1}{n}\sum_{i=1}^{n} d(x_i, \hat{x}_i)$. Examples of fidelity measures satisfying the above conditions are: *Hamming distance*, and the *square error distortion* where $d(x_i, \hat{x}_i) = (x_i - \hat{x}_i)^2$. As in Łuczak and Szpankowski [8, 9] we define the depth L_n as the length of the longest prefix of X_{n+1}^{∞} (i.e., uncompressed image) that approximately occurs in the database (i.e., already compressed image). More precisely, for a given $0 \leq D \leq 1$ we define: *Let L_n be the length k of the longest prefix of X_{n+1}^{∞} for which there exists i, $1 \leq i \leq n-k+1$, such that $d(X_i^{i-1+k}, X_{n+1}^{n+k}) \leq D$* where D

is the maximum distortion level.

A variable-length compression code can be designed based on L_n. The code is a pair (`pointer to a position` i, `length of` L_n), if a sufficiently long L_n is found. Otherwise we leave the next L_n symbols uncompressed. We clearly need $\log n + \log L_n$ bits for the above code. The compression ratio r can be approximated by

$$r = \frac{overhead\ information}{length\ of\ repeated\ subword} = \frac{\log n + \log L_n}{L_n}\ , \tag{1}$$

and to estimate it one needs to find out probabilistic behavior of L_n. In [8, 9] it is proved that $L_n \sim \frac{1}{r_0(D)} \log n$ where $r_0(D)$ is the so called generalized Rényi entropy (which could be views as follows: the probability of a typical ball in $\mathcal{A}^n$ of radius D is approximately $2^{-nr_0(D)}$). From (1) we conclude that the compression ratio r is asymptotically equal to $r_0(D)$.

The above main idea is enhanced by several observations that constitute PMIC scheme. They are:

Additive Shift. It is reasonable to expect that in an image there are several substrings, say x^m, y^m, that differ only by *almost* constant pattern, that is, $y_i = x_i + c$ for $1 \le i \le m$ where c is not necessary a constant, but whose variation is rather small.

In such a case we store the average difference c as well as a (*pointer*, *length*) pair, which enables us to recover y_i from x_i (instead of storing y_i explicitly). In general, we determine the additive shift c by checking whether the following condition is fulfilled or not

$$\frac{1}{m}\left(\sum_{i=1}^{m}(x_i - y_i)^2 - \frac{1}{m}\left(\sum_{i=1}^{m}(x_i - y_i)\right)^2\right) \le D'$$

where D' is a constant. If the above holds, we compute the shift c as $c = \frac{1}{m}\sum_{i=1}^{m}(x_i - y_i)$. It can be easily verified that for $x_i = y_i + c$, and c a constant, the above procedure returns c. In passing, we should observe that this enhancement improves even *lossless* ($D = 0$) image compression based on the Lempel-Ziv scheme.

Reverse Prefix. We check for similarities between a substring in the uncompressed file and a *reversed* (i.e., read backward) substring in the database.

Variable Maximum Distortion. It is well known that human vision can easily distinguish an "undesired" pattern in a low frequency (constant) background while the same pattern might be almost invisible to humans in high frequency (quickly varying) background.

Therefore, we used a low value of maximum distortion, say D_1, for slowly varying background, and a high value, say D_2, for quickly varying background. We automatically recognize these two different situations by using the derivative $\mathcal{D}_{ij}$ of the image that is defined as

$$\mathcal{D}_{ij} = \frac{(x_{i+1,j} - x_{ij}) + (x_{i,j+1} - x_{ij})}{2}$$

where x_{ij} is the value of (i,j) pixel. In our implementation, we used the lower value of D whenever $\mathcal{D}_{ij} \leq 3$ and the higher value of D otherwise.

Max-Difference Constraint. Our code will also satisfy the additional constraint that $|x_i - \hat{x}_i| \leq DD$ where DD is a suitably chosen value. This additional constraint, which we call *max-difference* constraint, ensures that visually noticeable "spikes" are not averaged out of existence by the smoothing effect of the square error distortion constraint. We incorporate this max-difference constraint in the function $d(\cdot,\cdot)$ by adopting the convention that $d(x_i, \hat{x}_i) = +\infty$ if $|x_i - \hat{x}_i| > DD$, otherwise $d(x_i, \hat{x}_i)$ is the standard distortion as defined above (i.e., Hamming or square of difference).

Small Prefix. It does not make too much sense to store a $(pointer, position)$ when the longest prefix is small. In this case, we store the original pixels of the image.

Run-Length Coding. For those parts of the picture that do not vary at all, or vary little, we used run-length coding: if the next pixels vary little, we store a pixel value and the number of repetitions.

As mentioned above, in this paper we investigate an impact of prediction algorithms on PMIC. A simple prediction enhancement can work as follows (prediction loop – our main contribution of this paper – is described in depth in next sections):

Prediction. A significant speed up in compression without major deterioration in quality can be obtained by using prediction. It works as follows: we compressed only every second row of an image, and when decompressed we restore the missing row by using a prediction (cf. [11], and next sections). For high quality image compression, one can design a two-rounds approach where in the second round one sends the missing rows. Figure 1 shows two versions of the compressed Lena picture: one with PMIC, and the other with PMIC and predictive "reduction/expansion".

Finally, we also explore in this paper one more feature, namely:

Column compression. PMIC is performed on rows. In order to have it performed on columns, one simple way to proceed is to "transpose" the original image before compression. Indeed, the rows of the compressed "transposed" image are the columns of the original one. This algorithm has been performed on several different images, and this feature can sometimes increase either the compression ratio or the quality level, as shown in Figure 1.

Let us discuss algorithmic challenges of PMIC. It boils down to an efficient implementation of finding the longest prefix of length L_n in yet uncompressed file. We present below one possible implementation (for others the reader is refereed to [2]. Let x_1^n denote the database, $y_1^m = x_{n+1}^{n+m}$ denote the yet uncompressed file. As before, we write $d(x_i, y_j)$ for a distortion measure between two symbols, where $d(\cdot,\cdot)$ is understood to incorporate the

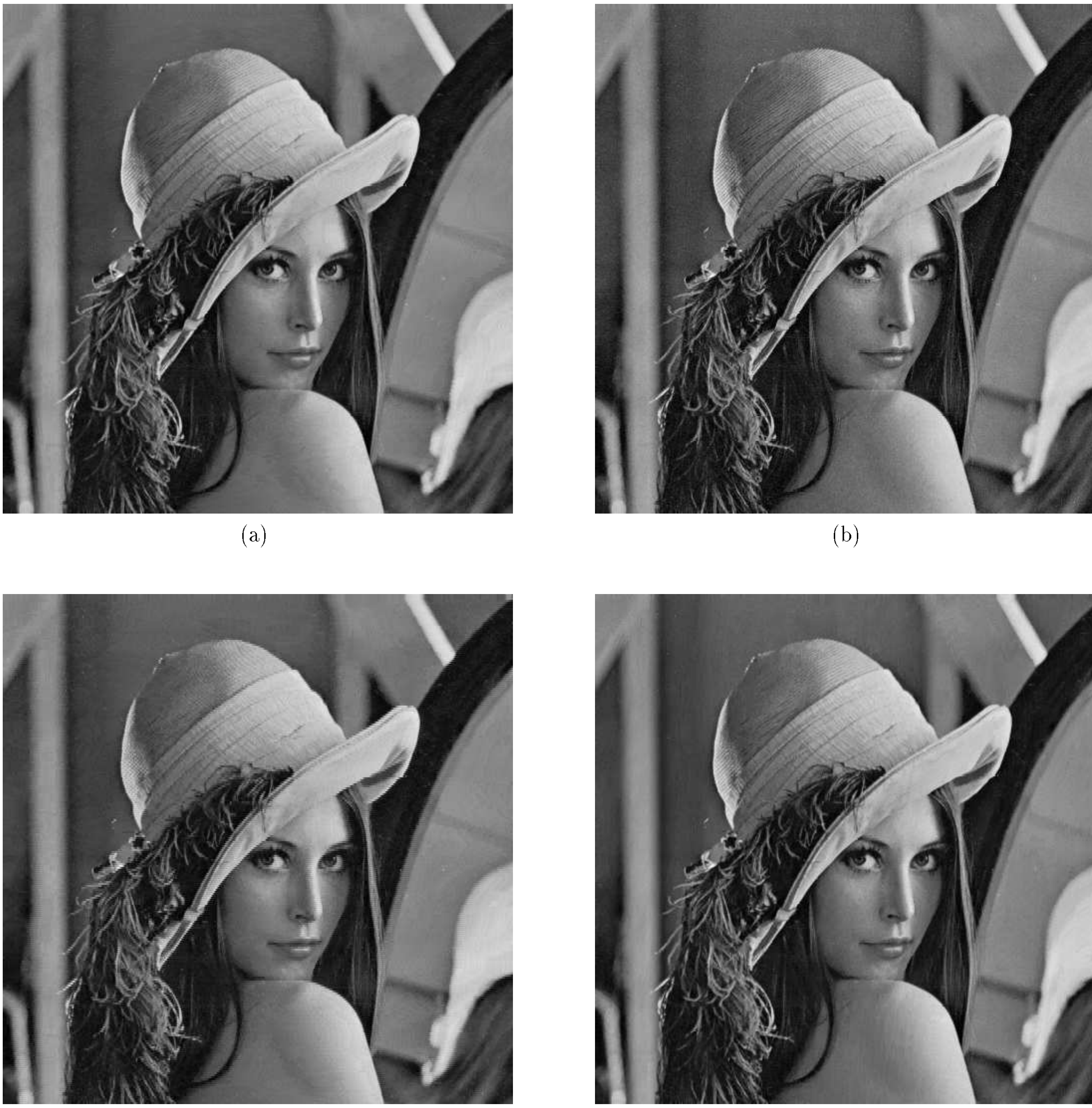

Figure 1: Comparisons of different flavors of PMIC on the “Lena” image: (a) Classical PMIC (compression ratio equal to 4.3); (b) JPEG (compression ratio equal to 7.9); (c) PMIC with simple predictive coding (compression ratio equal to 7.9); (d) PMIC with column compression (compression equal to 5.3).

max-difference criterion discussed earlier.

Algorithm `PREFIX`
Input: x_1^n and y_1^m
Output: Largest integer k such that, for some index t $(1 \leq t \leq n-m)$, $d(x_t^{t+k-1}, y_1^k) \leq D$. The algorithm outputs both k and t.
Method: We compute, for all i, j S_{ij} = total distortion measure between x_i^{i+j-1} and y_1^j.

```
begin
    Initialize all $S_{ij} := 0$.
    for $i = 1$ to $n - m$ do
        for $j = 1$ to $m$ do
            Compute $S_{ij} := S_{i,j-1} + d(x_{i+j-1}, y_j)$
        doend
    doend
    Let $k$ be the largest $j$ such that $S_{ij} \leq jD$, and let $t$ be the corresponding $i$
    Output $k$ and $t$
end
```

The above can easily be modified to incorporate the enhancements discussed above (additive shift, etc.) and to use $O(1)$ variables rather than the S_{ij} array.

In order to compress an $N \times N$ image we apply `PREFIX` several times to compress a row, which we call `COMPRESS_ROW` algorithm, and this is further use to compress a whole image in either `COMPRESS_LONG_DATABASE` (where the database sequence is of multiplicity of N, i.e., fN where f is a small integer) or `COMPRESS_SHORT_DATABASE` (where the database sequence is of length $O(1)$ but located just above the "point" of compression). In the former case the algorithmic complexity of compression is $O(N^4)$ while in the latter case it is $O(n^2 \log N)$ which is only $\log N$ away from the optimal complexity $O(N^2)$. In this paper we only use `COMPRESS_SHORT_DATABASE` algorithm.

3 Review of DPCM

In the following, $x_{i,j} = x_m$ stands for the color value of the pixel currently in position $(i,j) = m$ in the $N \times N$ pixels original image, and $\hat{x}_{i,j} = \hat{x}_m$ is the color value of the "predicted" (reconstructed) pixel of the reconstructed image. As seen in [11] and accordingly to Fig. 2, the configuration of the 2-D predictor is as following:

$$\hat{x}_m = 0.75A - 0.50B + 0.75C. \tag{2}$$

where $x_m = x_{i,j}$, $A = \hat{x}_{i,j-1}$, $B = \hat{x}_{i-1,j-1}$, and $C = \hat{x}_{i-1,j}$.

In the above, e_m stands for the error value between the original and the reconstructed pixels:

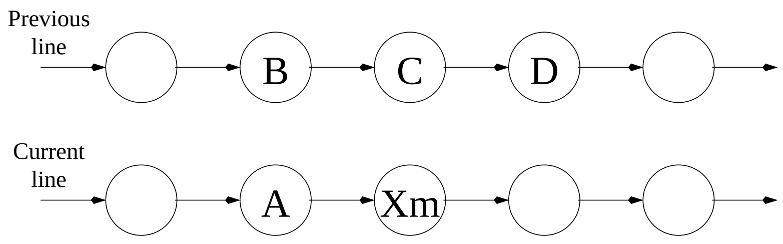

Figure 2: DPCM predictor configuration.

$$e_m = x_m - \hat{x}_m. \tag{3}$$

Finally, $e^\star_m$ is the quantized value of e_m, as described in [11] from which Table 1 is drawn. Then:

Algorithm `PREDICTION`
Input: $x_{i,j} = x_m$, $\hat{x}_{i,j-1} = A$, $\hat{x}_{i-1,j-1} = B$, and $\hat{x}_{i-1,j} = C$ (see Fig. 2).
Output: $e^\star_m$.
Method: We compute a first $\hat{x}_m$, then e_m and $e^\star_m$. Eventually, we update the value of $\hat{x}_m$.

```
begin
    Compute $\hat{x}_m$ accordingly to Eq. 2
    Clip $\hat{x}_m$ to the range [0,255]
    Compute $e_m := x_m - \hat{x}_m$
    Compute $e^\star_m := Quantize(e_m)$
    Upgrade $\hat{x}_m$ by computing $\hat{x}_m := \hat{x}_m + e^\star_m$
    Clip $\hat{x}_m$ to the range [0,255]
end
```

i	$(d_i, d_{i+1}) \to r_i$	Probability	Huffman Code
0	(-255,-16) $\to$ -20	0.025	111111
1	(-16,-8) $\to$ -11	0.047	11110
2	(-8,-4) $\to$ -6	0.145	110
3	(-4,0) $\to$ -2	0.278	00
4	(0,4) $\to$ 2	0.283	10
5	(4,8) $\to$ 6	0.151	01
6	(8,16) $\to$ 11	0.049	1110
7	(16,255) $\to$ 20	0.022	111110

Table 1: 8-level Lloyd-Max quantizer for the "Lena" image.

In fact, we add an offset of 128 to $e^\star_m$, making all of the error values positive (for an 8-bit original) so that they can be printed on a output device. The error image for a perfectly reconstructed image is thus uniform gray field with a code value of 128.

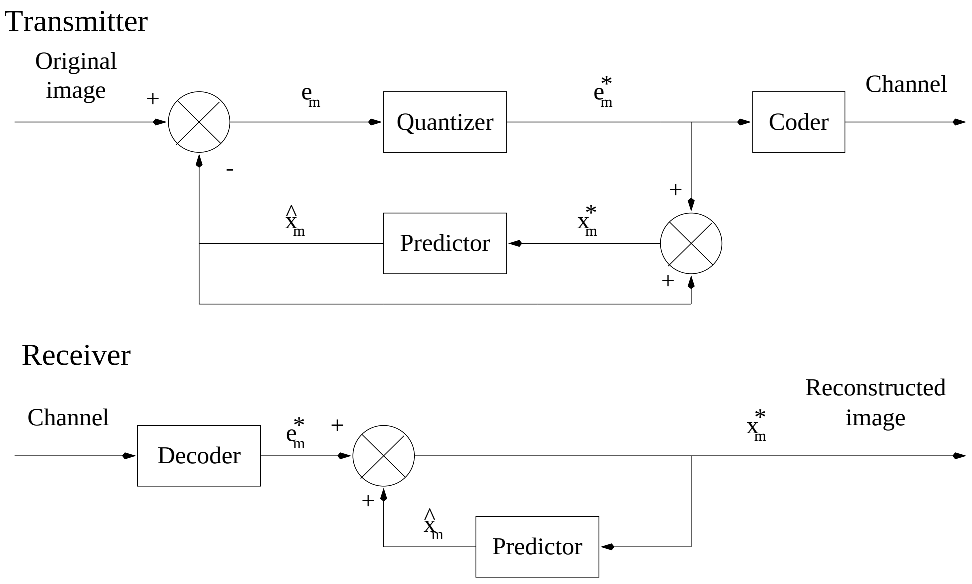

Figure 3: DPCM block diagram.

The DPCM algorithm is summarized in Figure 3. We observe that it applies to every pixel of the image. That leads to the following algorithm:

```
    **Algorithm** PREDICTION_LOOP
**Input:** $x_{i,j} = x_m$, for all $(0 \le i \le N-1)$ and $(0 \le j \le N-1)$.
**Output:** $e^\star_m$, for all $(0 \le i \le N-1)$ and $(0 \le j \le N-1)$.
**Method:** We use PREDICTION to compute $e^\star_m$ for each pixel.

begin
    Initialize $\hat{x}_m = x_m$ and $e^\star_m = x_m$ for all $(0 \le i \le N-1)$ and $j = 0$.
    for $j = 1$ to $N-1$ do
        for $i = 0$ to $N-1$ do
            Call PREDICTION on $x_{i,j} = x_m$, $\hat{x}_{i,j-1} = A$, $\hat{x}_{i-1,j-1} = B$, and
            $\hat{x}_{i-1,j} = C$ (see Fig. 2).
        doend
    doend
end
```

From Figure 3 one concludes that the prediction (reconstructed pixel) $\hat{x}_m$ for the transmitter is exactly the same as the one used by the receiver. They both need, and only need, the quantized difference $e^\star_m$. The transmitter puts this difference in a file, whereas the receiver reads this difference from the file.

Finally, $e^\star_m$, which is the quantized difference for each pixel, and the whole picture (formed with these quantized differences) is compressed with Huffman code.

4 New Algorithm: PMIC with DPCM

Like in Huffman compression (the compression part of DPCM), the PMIC compression algorithm should be performed on the quantized differential image. However, the algorithms seen above are well adapted to Huffman coding, because they deal on a pixel-by-pixel basis. Unfortunately, the PMIC algorithm, proceeds several pixels at a time (i.e., the longest prefix found in the database). Besides, we do not know in advance what the length of this prefix will be. But, there is a way around all of those problems. In fact, if we consider the `PREDICTION` algorithm to be performed, not on a pixel basis, but on a cell (which is, in the case of DPCM, one single pixel) basis, then this algorithm needs just a few adaptations in order to be suited for PMIC: in this latter, the cell becomes naturally the longest prefix found in the database.

These considerations lead to the algorithms given below, which should replace the `PREFIX` and `COMPRESS_ROW` algorithms for more detail) respectively:

Algorithm `DIFFERENTIAL_PREFIX`
Input: $\hat{x}_1^n$, ex_1^n (where $n = fN$), and y_1^N
Output: Largest integer k such that, for some index t $(1 \leq t \leq n-N)$, $d(ex_t^{t+k-1}, ey_1^k) \leq D$. The algorithm outputs both k and t, as well as $\hat{y}_1^N$ and ey_1^N, both returned by the call to the `PREDICTION_LOOP` algorithm.
Method: We first use `PREDICTION_LOOP` to form a differential row, then we compute, for all i, j, S_{ij} = total distortion measure between ex_i^{i+j-1} and ey_1^j.

```
begin
    Call PREDICTION_LOOP on $x_1^N$
    Let $\hat{y}_1^N$ and $ey_1^N$ be returned by this call to PREDICTION_LOOP.
    Initialize all $S_{ij} := 0$.
    for $i = 1$ to $n - N$ do
        for $j = 1$ to $N$ do
            Compute $S_{ij} := S_{i,j-1} + d(ex_{i+j-1}, ey_j)$
        doend
    doend
    Let $k$ be the largest $j$ such that $S_{ij} \leq jD$, and let $t$ be the corresponding $i$
    Output $k$ and $t$
end
```

One should note that the `PREDICTION_LOOP` algorithm is here used on generally more pixels than needed, since it is used on N pixels, and that the longest prefix will have a size $\leq N$.

Algorithm `COMPRESS_DIFFERENTIAL_ROW`
Input: $\hat{x}_1^n$, ex_1^n (where $n = fN$), y_1^N.
Output: A compressed version of ey_1^N, in the form of $(pointer, length)$ pairs, and the new reconstructed portion, $\hat{y}_1^N$, of the database.
Method: We use `PREFIX` to "peel off" a prefix of the row being compressed, and repeat on the remaining portion of that row until we use it all up.

```
begin
    Initialize $i := 0$.
    Repeat the following until $i \geq N$
        Call PREFIX on $\hat{x}_1^n$, $ex_1^n$ and $y_{i+1}^N$
        Let $k$ and $t$, $\hat{y}_{i+1}^N$ and $ey_{i+1}^N$, be returned by this call to PREFIX
        If $k$ is small (say, $\leq 4$)
        then $ey_{i+1}^{i+k}$ is stored explicitly,
        else $ey_{i+1}^{i+k}$ is stored as a $(pointer, length)$ pair,
        Compute $\hat{y}_{i+1}^{i+k} := \hat{y}_{i+1}^{i+k} + ey_{i+1}^{i+k}$ (addition of vectors).
        Set $i := i + k$
end
```

We have used here the notation ex and ey for x and y, as they are understood in the `COMPRESS_ROW` algorithm of [2] and discussed briefly above, except that what is now compressed is the quantized differential image, whence the e before x and y. Therefore, $\hat{x}_1^n$ represents the reconstructed ("predicted") database (and is, in fact, the image as it will be again after decompression); ex_1^n means the database of the differential image (image actually being compressed); y_1^N is the currently processed row of the original image; ey_1^N is the "differential" row currently being compressed, that is, the differential pendant of y_1^N; and $\hat{y}_1^N$ is the new portion of the reconstructed ("predicted") database. Thus, $\hat{y}$ leads to $\hat{x}$, and ey gives ex.

Table 2: "Lena" image compressed by JPEG (quality=57), PMIC with constant D (D = 155, DD = 20), PMIC with variable D (D1 = 500, D2 = 55, DD = 22), and PMIC-PL (D1 = 600, D2 = 50, DD = 16).

	JPEG	Const D	Var D	PMIC-PL
Compression ratio	10.03	9.99	10.01	10.04
Bit rate (bits/pixel)	0.8	0.8	0.8	0.8
RMSE (0-255)	4.65	8.90	8.44	15.97
PSNR (dB)	34.78	29.14	29.61	24.06

Table 3: "Basselope" image compressed by JPEG (quality=57), PMIC with constant D (D = 800, DD = 38), PMIC with variable D (D1 = 800, D2 = 100, DD = 39), and PMIC-PL (D1 = 1000, D2 = 45, DD = 15).

	JPEG	Const D	Var D	PMIC-PL
Compression ratio	9.96	10.04	10.04	9.96
Bit rate (bits/pixel)	0.8	0.8	0.8	0.8
RMSE (0-255)	9.07	12.22	8.44	78.89
PSNR (dB)	28.97	26.39	29.61	10.19

Finally, we present some experimental results. We summarize them in Tables 2—4 where JPEG, PMIC with constant and variable D, and PMIC-PL (PMIC with Prediction

Table 4: "Lena" image compressed by JPEG (quality=21), PMIC with constant D (D = 400, DD = 38), PMIC with variable D (D1 = 500, D2 = 55, DD = 22), and PMIC-PL (D1 = 600, D2 = 50, DD = 16).

	JPEG	Const D	Var D	PMIC-PL
Compression ratio	10.03	10.00	9.95	10.02
Bit rate (bits/pixel)	0.8	0.8	0.8	0.8
RMSE (0-255)	7.96	13.92	12.87	20.41
PSNR (dB)	30.11	25.26	25.94	21.94

Loop) are compared for three images, namely: "Lena" image, "Basselope" image (a graphic image) and "San Francisco" image (a photographic image). We conclude that prediction, as discussed in Section 2, can lead to substantial speed up of compression time, butter compression time without significant deterioration of image quality. More sophisticated PMIC-PL cna lead to further improvements, as our preliminary results suggest. However, results in Tables 2-4 are, in fact, worse than expected. We believe they originate from the fact that we use variable-D scheme which seems to be unsuited for the predictive loop (the derivative is actually done on the original image, and not on the differential image, as it should be). This might be an interesting point further research (*i.e.*, to implement a good derivative scheme on the differential image). Then, we believe we can achieve much more substantial improvements. This will be reported in our journal version of this paper.

References

[1] A. Habibi, Comparison of nth-order DPCM encoder with linear transformations and block quantization techniques, *IEEE Trans. Commun. Tech.*, COM-19, 948-956 (1971).

[2] M. Atallah, Y. Genin, and W. Szpankowski, A Pattern Matching Approach to Image Compression, *Proc. International Conference on Image Processing*, vol. II, 349-352, Lausanne (1996); see also Purdue University, CSD-TR-95-083 (1995).

[3] T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*, Englewood Cliffs, NJ: Prentice-Hall, 1971.

[4] C. Constantinescu and J. Storer, Improved Techniques for Single-Pass Adaptive Vector Quantization, *Proc. of the IEEE*, 82, 933-939 (1994).

[5] M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, New York (1995).

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

[7] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood (1989)

[8] T. Łuczak and W. Szpankowski, A Lossy Data Compression Based on String Matching: Preliminary Analysis and Suboptimal Algorithms, *Proc. Combinatorial Pattern Matching*, Asilomar, LNCS 807, 102-112, Springer-Verla (1994).

[9] T. Łuczak and W. Szpankowski, A Suboptimal Lossy Data Compression Based in Approximate Pattern Matching, *IEEE International Symposium on Information Theory*, Whistler, 1995; also under revision in *IEEE Trans. Information Theory*, and Purdue University, CSD-TR-94-072 (1994).

[10] A. N. Netravali and J. O. Limb, Picture Coding : A Review, *Proc. IEEE*, 68, 366-406 (1980).

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

[12] D. K. Sharma and A. N. Netravali, Design of Quantizers for DPCM Coding of Picture Signals, *Proc. IEEE Trans. Commun.*, COM-25, 1267-1274 (1978).

[13] 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).

[14] W. Szpankowski, A Generalized Suffix Tree and Its (Un)Expected Asymptotic Behaviors, *SIAM J. Computing*, 22, 1176-1198 (1993).

[15] A. Wyner and J. Ziv, Some Asymptotic Properties of the Entropy of a Stationary Ergodic Data Source with Applications to Data Compression, *IEEE Trans. Information Theory*, 35, 1250-1258 (1989).

[16] E.H. Yang, and J. Kieffer, On the Performance of Data Compression Algorithms Based upon String Matching, preprint (1995).

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

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