PATTERN MATCHING IMAGE COMPRESSION: Algorithmic and Empirical Results

July 26, 1996

Mikhail Atallah*	Yann Génin	Wojciech Szpankowski†
Dept. of Computer Science	Ecole de la Météorologie	Dept. of Computer Science
Purdue University	42, av. Coriolis	Purdue University
W. Lafayette, IN 47907	31057 Toulouse	W. Lafayette, IN 47907
U.S.A.	France	U.S.A.

Abstract

We propose a non-transform image compression scheme based on approximate pattern matching, that we 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 image that *approximately* occurs in the already processed imager (e.g., in the sense of the *Hamming distance* or, alternatively, of the *square error distortion*). This main algorithm is enhanced with several new features 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 D, 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. Both algorithmic and experimental results are presented. Our scheme turns out to be competitive with JPEG and wavelet compression for graphical and photographical images. A unique feature of the proposed algorithm is that an asymptotic performance of the scheme can be theoretically established. More precisely, under stationary mixing probabilistic model of an image and fixed maximum distortion level D, it is shown that the compression ratio is asymptotically equal to the so called *generalized Rényi entropy* $r_0(D)$. This entropy is in general smaller than the optimal rate distortion function $R(D)$, but there is numerical evidence that these two quantities do not differ too much for small and medium values of D.

Index Terms: Lossy Lempel-Ziv scheme, approximate pattern matching, image compression, generalized Rényi entropy, Hamming and square-root distortion, mixing probabilistic model, string editing and algorithms on words, Fast Fourier Transform, image derivative, JPEG, wavelets and fractal image compression.

*Research supported by the National Science Foundation under Grant CCR-9202807.

†This work was supported by NSF Grants NCR-9415491 and CCR-9201078, and in part by NATO Collaborative Grant CGR.950060.

1. INTRODUCTION

Data compression based on *exact* pattern matching can be traced back to seminal papers of Lempel and Ziv [31, 32, 33], but recently there has been a resurgence of interest in this type of data compression. This might be a consequence of rapid growth in digital representation of multimedia (e.g., text, audio, image, video, etc.) which are particularly amenable to pattern matching manipulations. It was known for a along time that pattern matching based data compression – such as Lempel-Ziv schemes LZ77 [32] and LZ78 [33] – is very attractive for text compression. For example, such schemes were used in the UNIX `compress` and `gunzip` commands, and in a CCITT standard for data compression for modems. An attractive feature of such solutions is that one can prove asymptotical optimality of lossless data compression schemes based on Lempel-Ziv algorithms. A natural question that arises is whether lossy extensions of the Lempel-Ziv scheme are asymptotically optimal, and whether they might be of practical interest. In particular, one may wonder whether image compression based on approximate pattern matching is an attractive solution and can be competitive with standards such as JPEG, or with newer image compression techniques such as fractal or wavelet. In this paper, we believe we can give an affirmative answer to these questions. After recalling (and somewhat extending) some recent theoretical results of Luczak and Szpankowski [18, 19] which constitute a theoretical basis for the lossy compression based on approximate pattern matching, we present our experimental results with pattern matching image compression that support the above claim.

It must be said that early attempts on lossy compression based on pattern matching were rather unsuccessful. Already in 1980 Ziv [31] (cf. also [28]) proposed an optimal lossy compression scheme at fixed rate level, while Ornstein and Shields [21], and independently Kieffer [14] gave a universal lossy compression for coding at fixed distortion level. Unfortunately, all of these schemes were prohibitively expensive from the computational complexity point of view. Recently, a quest for asymptotically optimal and computationally attractive lossy data compression based on approximate pattern matching has begun [9, 23, 24, 30]. But one may wonder whether a practical and optimal lossy compression exists at all. Yang and Kieffer in their recent paper [28] expressed the following opinion: "... it is our belief that a universal lossy source coding scheme with attractive computational complexity aspects will never be found." We share this view, and we believe that investigations of suboptimal and practical heuristics for lossy compression are needed.

In view of this, Luczak and Szpankowski [18, 19] (cf. also [24]) constructed a simple, computationally attractive lossy data compression based on approximate pattern matching.

The main idea was to search for the longest prefix of the uncompressed file that approximately occurs in the already compressed file (the so called "training sequence" or "database sequence"). It was proved in [19] that under a stationary mixing probabilistic model of an image, the compression ratio can asymptotically achieve the so called *generalized Rényi entropy* $r_0(D)$ (cf. next section for a precise definition). It was observed that $r_0(D) \geq R(D)$ where $R(D)$ is the optimal (rate-distortion) compression ratio. The next step to undertake is to see whether a lossy (e.g., for images) compression scheme based on such an approximate pattern matching can lead to a practical and efficient algorithm (i.e., computationally and in terms of compression ratio). In this paper, we discuss algorithmic issues encountered in image compression based on pattern matching, and report our experimental studies.

It must be stressed that the scheme we shall propose in this paper, henceforth called *Pattern Matching Image Compression* (PMIC for short), is a major modification of the basic idea described above and analyzed in [19]. A straightforward implementation of the basic scheme on real images (structured data) seems not to be attractive from a practical point of view, so that the enhancements we describe in what follows play an important role in the quality of the experimental results we obtained. These include searching for reverse approximate matching, recognizing substrings in images that are additively shifted versions of each other, making the maximum distortion level D variable and adaptive, and so forth.

The implementation of our scheme leads to algorithmic issues that are of independent interest, e.g., the computation of the longest prefix of a string that almost-occurs in another string (perhaps in an additively shifted form). The algorithms we give for the Hamming distance case are quite different from those we give for the square error distortion.

Our general practical conclusion can be summarized as follows: The proposed non-transform scheme achieves compression ratios comparable to JPEG (UNIX implementation), wavelet compression (implementation based on [8]), and better than fractal image compression (implemented according to [10]). PMIC works particularly well for images with high frequencies (e.g., containing sharp egdes, etc). The compression time is slower than transform based methods such as JPEG and wavelet, but decompression time seems to be the fastest possible due to the fact that our decompression scheme mostly only reads and writes data without any processing (occasionally it performs one addition operation between the read and the write). We believe that our compression time will become competitive with transform based methods once we implement the faster compression schemes of Section 3.

There is a huge volume of knowledge on image processing (cf. [12, 13, 22]) but the majority of image compression techniques are based on transform methods. On the other hand, lossy Lempel-Ziv schemes based on approximate pattern matching were discussed in [9, 16, 18, 19, 23, 24, 29], however, to the best of our knowledge (with a possible exception of [9]) no real and successful image compression implementation was reported so far. In fact, a literature on the probabilistic analysis of approximate pattern matching is rather scarce, too. We should mention here the paper of Steinberg and Gutman [24], and Łuczak and Szpankowski [18, 19], as well as recent results of Yang and Kieffer [29]. Arratia and Waterman [3] also analyzed an approximate pattern matching problem in the context of molecular biology. The reader is referred to the survey [11] and/or a recent book by Crochemore and Rytter [6] on string matching algorithms. As will become apparent soon (cf. Section 3), most of the algorithms for approximate pattern matching proposed in the literature so far will not be applicable to our situation of lossy image compression.

The paper is organized as follows. In the next section we recall some concepts and results from Łuczak and Szpankowski [19]. Next we discuss algorithmic issues, namely fast algorithms to identify a longest prefix that occurs approximately in the database, and we present several variations of image compression algorithms. Finally, we discuss our implementation of the PMIC scheme, and present several results on graphic and photographic images.

2. PROBLEM FORMULATION AND THEORETICAL RESULTS

In this section, we describe in general terms a lossy data compression based on approximate pattern matching, and review and slightly generalize theoretical results of Łuczak and Szpankowski [19] (cf. also [24, 29]) that constitute a foundation for the performance of the lossy compression scheme. We formulate our results in terms of distortion rate theory of source coding to show their generality and further potential applicability to multimedia (e.g., audio compression).

2.1 Basic Definitions

Consider a stationary and ergodic sequence $\{X_k\}_{k=1}^{\infty}$ taking values in a finite alphabet $\mathcal{A}$. 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 $X^n = X_1 \ldots X_n$. We also use $P(X^n)$ for the probability of the n-tuple 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. We assume that the reproduction alphabet $\hat{\mathcal{A}} = \mathcal{A}$. More precisely, a 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 representing x_1^n. Then, the *compression ratio* is defined as $r(x_1^n) = \ell(c_n)/n$ (e.g., for image compression $r(x_1^n)$ is expressed in bits per pixel, i.e., bpp), and the *average* compression ratio is $E(r(X_1^n)) = E\ell(C_n)/n$.

Since we are interested in lossy compression, we need a measure of fidelity $d(\cdot,\cdot)$ that defines how far away the reproduction vector $\hat{X}^n$ (e.g., compressed/decompressed image) is from the source vector X^n. We only consider *single-letter fidelity* measures, that is, such that

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

Furthermore, in order to use properly the rate distortion theory we impose the following two conditions on the fidelity measure (cf. [5, 14, 24]):

(F1) SUBADDITIVITY. For any two integers n, m, and given vectors x^{n+m}, y^{n+m} we postulate that

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

(F2) FINITENESS. For each $D > 0$, there exists a countable subset $\mathcal{A}_1$ of $\mathcal{A}$ and a countable measurable partition $\{E_i\}$ of $\mathcal{A}$ such that $d(x, y) \leq D$ for $x \in \{E_i\}$ and $y \in \mathcal{A}_1$ such that

$$-\sum_i P(E_i) \log P(E_i) < \infty \ .$$

Examples of fidelity measures satisfying (F1) and (F2) are: *Hamming distance*, where $d(x_i, \hat{x}_i)$ equals one if $x_i = \hat{x}_i$ and equals zero otherwise, and the *square error distortion* where $d(x_i, \hat{x}_i) = (x_i - \hat{x}_i)^2$.

Our experimental work has concentrated on the square error distortion, which is natural for image compression, and constructs, for a given $D > 0$, a *D-semifaithful* code, i.e., one such that $d(x^n, \hat{x}^n) \leq D$. Our code will also satisfy the additional constraint that $|x_i - \hat{x}_i| \leq \Delta$ where Δ 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 $|x_i - \hat{x}_i| > \Delta$, otherwise $d(x_i, \hat{x}_i)$ is the standard distortion as defined above (i.e., Hamming or square of difference).

We are now in a position to describe an approximate pattern matching that constitutes a basis for a lossy compression scheme. We assume that $\{X_k\}_{k=1}^{M}$ is a file to be compressed (e.g., $M = N^2$ for an $N \times N$ image). Let $X^n = X_1 \ldots X_n$ be a database or training sequence (which is sent to a decoder *without* compression). One can think of X^n as the first few rows in an $N \times N$ image. Our goal is to find a code that represents the rest of the file, X_{n+1}^M, in as few bits as possible, *and* to assure that the construction of the code is *computationally* efficient.

As in Łuczak and Szpankowski [18, 19] we define the depth L_n as the length of the longest prefix of X_{n+1}^∞ that approximately occurs in the database. More precisely:

> Let L_n be the length k of the longest prefix of X_{n+1}^∞ for which there exists i, $1 \le i \le n-k+1$, such that $d(X_i^{i-1+k}, X_{n+1}^{n+k}) \le D$.

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. Once $X_{n+1}^{n+L_n}$ is coded, we append the next L_n symbols to the database, and repeat the procedure with $X_{n+L_n+1}^\infty$. Such a scheme can be called the *enlarged-database* scheme. In another implementation one can keep the database fixed so that the longest prefix is not added to the database. The latter is called *fixed-database* scheme 9cf. [27]).. Finally, in a sliding window implementation one adds L_n symbols to the database and simultaneously the first L_n symbols of the database are deleted keeping the size of the databased constant. In our implementation (cf. Section 3) we keep the database unchanged while compressing the next row of an image, and after processing a *whole* row we add it to the database deleting the first row from the previous database.

In the enlarged-database scheme discussed in this section, the compression ratio r can be approximated by

$$r = \frac{\text{length of the overhead information}}{\text{length of repeated subword}} = \frac{\log n + \log L_n}{L_n} \ . \tag{2}$$

However, Kieffer in a private correspondence [15] pointed out that a precise estimation of the compression ratio is more complicated. Indeed, let $X^{(k)}$ be the database after the kth application of the above procedure. Observe that

$$X^{(k+1)} = X^{(k)} * X_{|X^{(k)}|+1}^{|X^{(k)}|+L_{|X^{(k)}}} \tag{3}$$

where $*$ denotes concatenation. Then, the compression ratio r should be computed as

$$r = \frac{\log|X^{(1)}| + \log|X^{(2)}| + \cdots + \log|X^{(k)}|}{|X^{(k+1)}|} \ . \tag{4}$$

In this paper, we adopt (2) to simplify the presentation.

Finally, we should mention in passing that one can design a block coding (i.e., fixed length code) for lossy data compression based on another parameter, namely the *waiting time* N_ℓ (cf. [24, 26, 29, 27]), defined as the smallest $N \geq 2\ell$ such that $d(X_1^\ell, X_{N-\ell+1}^N) \leq D$. In this paper, we focus on L_n and variable length compression codes.

2.2 Main Theoretical Results

We review and slightly generalize results of Łuczak and Szpankowski [19] to demonstrate the quality of the lossy data compression outlined in the previous subsection. In order to formulate the results we must adopt a suitable probabilistic model. This turns out to be a mixing stationary model, to be defined below.

(M) MIXING MODEL

Let $\mathcal{F}_m^n$ be a σ-field generated by $\{X_k\}_{k=m}^n$ for $m \leq n$. There exists a function $\alpha(\cdot)$ of g such that: (i) $\lim_{g\to\infty} \alpha(g) = 0$, (ii) $\alpha(1) < 1$, and (iii) for any m, and two events $A \in \mathcal{F}_{-\infty}^m$ and $B \in \mathcal{F}_{m+g}^\infty$ the following holds

$$(1-\alpha(g))P(A)P(B) \leq P(AB) \leq (1+\alpha(g))P(A)P(B) \ . \tag{5}$$

The probabilistic behavior of L_n (as well as N_ℓ) depends on a generalized entropy. To define it, we must first introduce a D-ball $B_D(w_k)$ with center $w_k \in \mathcal{A}^k$, which represents all strings of length k that are within distance D from the center w_k; that is, for $w_k \in \mathcal{A}^k$, $B_D(w_k) = \{x_1^k : \ d(w_k, x_1^k) \leq D\}$. We simple write $P(B_D(X_1^n))$ for the probability measure of the set of all sequences of length n within distance D from a random sequence X_1^n.

Definition. (GENERALIZED RENYI ENTROPY) For a fixed $D > 0$ let

$$r_0(D) = \lim_{k\to\infty} \frac{-E \log P(B_D(X_1^k))}{k} \ , \tag{6}$$

provided the above limit exist. □

The following lemma provides a condition under which $r_0(D)$ exists.

Lemma 1. *Under assumption (M) regarding the mixing stationary model, the generalized entropy $r_0(D)$ exists for any subadditive distortion measure satisfying conditions (F1)-(F2), and, furthermore*

$$r_0(D) = \lim_{k\to\infty} \frac{-\log P(B_D(X_1^k))}{k} \quad \text{(a.s.)} \ . \tag{7}$$

Proof. Result (7) follows directly from the subadditivity of the distortion measure (F1) and mixing model assumption (M) by an application of the *Subadditive Ergodic Theorem* along the lines of arguments presented in [19]. ■

We finally can present our main theoretical results that provides a basis for the image compression discussed in the next section.

Theorem 1. *Under assumption (M) and condition (F1) regarding the distortion measure, the following holds*

$$\lim_{n\to\infty} \frac{L_n}{\log n} = \frac{1}{r_0(D)} \qquad \text{(pr.)} \tag{8}$$

provided $\alpha(g) \to 0$ *as* $g \to \infty$, *and the rate of convergence of* $\log P(B_D(X_1^n))/n$ *in Lemma 1 is at least as good as* $O(1/n^{1+\delta})$ *for some* $\delta > 0$. *Under the same assumptions, we have*

$$\lim_{\ell\to\infty} \frac{\log N_\ell}{\ell} = r_0(D) \qquad \text{(pr.)} \ , \tag{9}$$

however, $L_n/\log n$ **does not** *converge almost surely to any limit while* $\log N_\ell/\ell \to r_0(D)$ *(a.s.) provided* $\sum_{g\geq 1} \alpha(g) < \infty$.
Furthermore, the compression ratio as defined in (2) becomes

$$r = r_0(D) \qquad \text{(pr.)} \ . \tag{10}$$

Proof. The results (8) and (9) follow directly from Lemma 1 and the *first and second moment methods* along the lines of arguments used in [19]. The almost sure convergence of $\log N_\ell/\ell$ is proved in [29], while the lack of almost sure convergence of $L_n/\log n$ is established in [19]. Finally, (10) is a simple consequence of (2) and (8). We conjecture after Kieffer [15] that the compression ratio as defined in (4) also converges almost surely to $r_0(D)$. ■

In [18, 19] the Rényi entropy $r_0(D)$ was computed for memoryless sources and Hamming distance. In Figure 1 we compared it to the optimal rate distortion $R(D) = h + D\log D + (1-D)\log(1-D)$, where h is the source entropy rate. One should observe that $r_0(D)$ is very close to the optimal $R(D)$ for small and medium values of D. We expect this to be true for Markov sources as well as mixing sources satisfying the conditions of (M).

Finally, in a practical implementation of lossy data compression for images one must vary the maximum distortion measure D in order to avoid visual errors for low frequency components of an image. Let us assume only two distortion levels D_1 and D_2, $D_1 < D_2$, and let the level D_1 be used on a large area of, say A_1 pixels, while D_2 is applied to an area of $A_2 = N^2 - A_1$ pixels, where an image of $N \times N$ pixels is analyzed. Then, one easily sees that

$$r \approx \frac{A_1}{N^2} r_0(D_1) + \left(1 - \frac{A_1}{N^2}\right) r_0(D_2) \ , \tag{11}$$

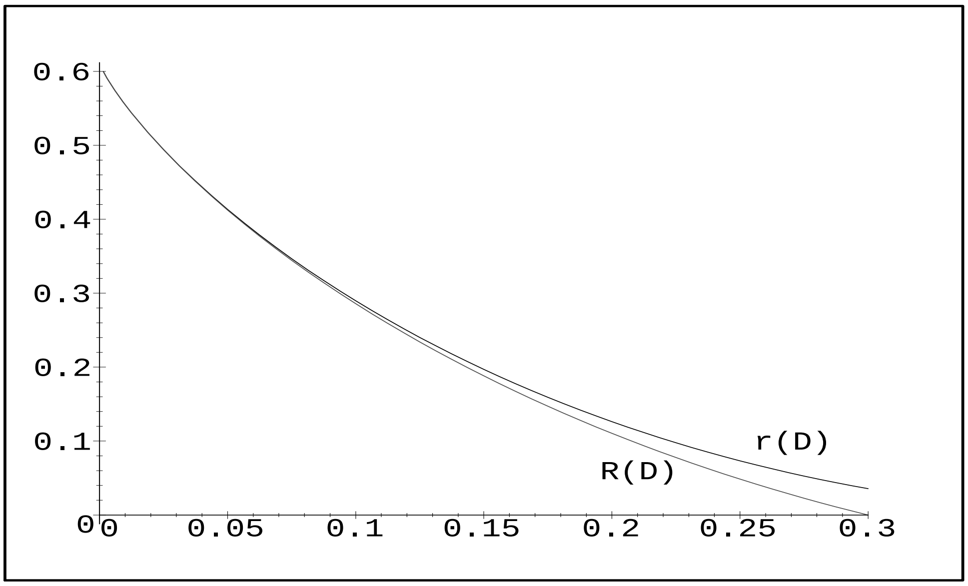


Figure 1: Comparing optimal rate distortion $R(D)$ and Rènyi entropy $r_0(D)$ for $p = 0.3$

where $r_0(D_1)$ and $r_0(D_2)$ are computed according to (6) provided one can adopt assumption (M) on the areas A_1 and A_2. The above formula is valid asymptotically when n becomes large. We should also observe that in fact $r = r_0(D)(1 + O(\log\log n/\log n))$, thus one expects the redundancy to be of order $O(\log\log n/\log n)$.

3. ALGORITHMIC RESULTS

In this section we address the computational challenge, that is, how to compress efficiently an image. The pivotal problem is to find an efficient algorithm that searches for the longest prefix approximately appearing in the database. We discuss several algorithms to accomplish it, and we use them to present general compression algorithms. Detailed implementation, with crucial enhancements, is discussed in the next section.

Throughout the rest of the paper we consider an $N \times N$ pixels image. We often assume that $n = f \times N$ where $1 \le f \le 8$. As before, we write x_1^n as the database, and $y_1^m = x_{n+1}^{n+m}$ as the yet uncompressed file. In other words, we number all $M = N^2$ pixels consecutively from $i = 1$ to $i = M$, and consider a linear string x_1^M. In some cases, it is more convenient for us to use double-index, so we write $\{x_{i,j}\}_{i,j=1}^N$. Finally, we observe that to compress an image of $N \times N$ pixels one must look at least once at every pixel, thus $\Omega(N^2)$ is an obvious lower bound for the compression time complexity.

3.1 Brute-Force Algorithm

As mentioned above, the main algorithmic problem is that of finding an efficient way of computing the longest prefix of y_1^m that approximately occurs in a database x_1^n.

We start with a judicious implementation of the brute force idea that computes the longest prefix in $O(mn)$ steps in the worst case. 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 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 later (additive shift, etc) and to use $O(1)$ variables rather than the S_{ij} array. We avoided doing so here in order not to unnecessarily clutter the exposition.

Of course the above algorithm is used within a compression routine whose goal is to compress all of y_1^m rather than just the prefix y_1^k. More specifically, in an image of size $N \times N$, the database x_1^n consists of the last f rows (we typically use $1 \leq f \leq 8$) encountered prior to the current row (i.e., $n = fN$), and the string y_1^m to be compressed is the current row (i.e., $m = N$). Such an algorithm for compressing a row would use `PREFIX` repeatedly, as follows:

Algorithm `COMPRESS_ROW`

Input: x_1^n where $n = fN$ and y_1^m where $m = N$.

Output: A compressed version of y_1^m, in the form of $(pointer, length)$ pairs.

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** $i = m$:

Call `PREFIX` on x_1^n and y_{i+1}^m.

Let k and t be returned by this call to `PREFIX`:

If k is small (say, ≤ 4)

then y_{i+1}^{i+k} is stored explicitly,

else y_{i+1}^{i+k} is stored as a $(pointer, length)$ pair.

Set $i := i + k$

end

Since `COMPRESS_ROW` uses `PREFIX` $O(N)$ times its time complexity is $O(N^3)$. (Another version of `COMPRESS_ROW` would append the already compressed portion of y_1^m to the database within the main loop, i.e., would subsequently call `PREFIX` on $x_1^n * y_1^i$ and y_{i+1}^m; we expect only a minor performance improvement due to such a change).

The algorithm for compressing an $N \times N$ image by N applications of the above `COMPRESS_ROW` is called `COMPRESS_LONG_DATABASE` (in short: `CLD`) in contrast to another algorithm called `COMPRESS_SHORT_DATABASE` (in short: `CSD`), in which $n = O(\log N)$ and is which discussed later.

Algorithm `COMPRESS_LONG_DATABASE (CLD)`
Method: We use a "sliding window"' of the f previous rows as the database for compressing the current row.

1. **begin**
2. **for** $i = 1$ **to** N **do**
3. Use `COMPRESS_ROW` on x_1^n = concatenation of rows $i - f, \cdots, i - 1$, and with y_1^m = row i.
4. **endo**
5. **end**

The worst case time complexity of `COMPRESS_LONG_DATABASE` is $O(N^4)$ because it calls `COMPRESS_ROW` N times, each time at a cost of $O(N^3)$. The corresponding decompression algorithm mostly *copies* and *reads*, and thus is very fast. Its concrete implementation (e.g., how the pointers are stored, how it improves on the above, etc) will be discussed in the

(a) (b)

Figure 2: Comparison of compression quality and computational time for similar compression ratios of the "San Francisco" image by: (a) `COMPRESS_SHORT_DATABASE` PMIC scheme (56 seconds compression time); (b) `COMPRESS_LONG_DATABASE` PMIC scheme (1380 seconds compression time)

next section. The worst case complexity of the above `CLD` algorithm is too expensive for some applications (e.g., in a real-time system). To improve this we either must design a faster algorithm searching for the longest prefix (given later) and/or decrease the database length. The latter solution turns out to be very attractive and we discuss it below.

We next sketch the `COMPRESS_SHORT_DATABASE` algorithm, which imitates `CLD` except that it makes use of two observations to reduce the compression time. The first is that Theorem 1 of Section 2 shows that the length L_n of the prefix we seek is, with high probability, $O(\log N)$. This suggests that it is reasonable to restrict the search to a prefix of length $O(\log N)$ (although there is a chance we may be missing the best possible L_n by only looking for an L_n which is $O(\log N)$). The second observation is that, in an image, most similarities occur within close proximity, which suggests that only a fixed number of positions at the f previous rows be checked for the almost-occurrence of the length-L_n prefix: Namely, if we are at column j of the current row i (the row currently being compressed), then the positions we check are positions (i', j') where $i' \in [i-f, i-1]$ and $j' \in [j-c', j+c']$, and c' is a constant. This implies that the prefix computation now takes $O(\log N)$ time because we are now checking only $2fc'$ $(= O(1))$ positions in the f previous rows rather than all fN

positions in these rows, and each position takes time $O(\log N)$ since we are looking only for a prefix of length $O(\log N)$. Thus the total worst-case complexity is $O(N^2 \log N)$, only a factor of $\log N$ away from the lower bound. The average case complexity is $\mathcal{O}(N^2)$ since, on average, we do $\mathcal{O}(N/\log N)$ prefix computations per row, at a cost of $O(\log N)$ time each, and there are N rows. (We will henceforth write $O(\cdot)$ for the worst case complexity, and $\mathcal{O}(\cdot)$ for the average case complexity.) This is an attractive compression speed which makes the PMIC scheme competitive with other transform based schemes such as JPEG. But, shortening the database and using "locality" lead to a deterioration of the compression ratio. How much do we pay for this ? Fortunately, our experiments indicate that for most images the deterioration is slight. Figure 2 is but one of the figures supporting this claim.

However, the next sub-sections show that it is possible to obtain faster compression times even without resorting to the use of `COMPRESS_SHORT_DATABASE`. They do so by giving algorithms for faster (but approximate) implementations of the `PREFIX` procedure. Two of these approximation schemes are based on the *Fast Fourier Transform* (cf. [17], pp. 290-294), and are discussed in the sequel.

3.2 Faster Algorithm for Square Error

This sub-section deals with a fast approximate implementation of `PREFIX` for the square error, without the max-difference enhancement but with the additive-shift enhancement described earlier (we later explain how to handle max-difference).

The first building block we need is an algorithm which tests whether one specific prefix y_1^k of y_1^m almost-occurs in the database x_1^n, i.e., whether there is a position i in x_1^n for which the following holds:

$$d(x_i^{i+k-1}, y_1^k) = \frac{1}{k}\sum_{j=1}^{k}(x_{i+j-1} - y_j - \bar{\delta})^2 \;\leq\; D \;,$$

where

$$\bar{\delta} = \frac{1}{k}\sum_{j=1}^{k}(x_{i+j-1} - y_j) \;.$$

Before giving the algorithm for detecting the above situation, let us recall that, in such a case, data compression is possible by avoiding storing y_1^k explicitly, and instead storing the additive shift $\bar{\delta}$ together with a $(pointer, length)$ pair $= (i, k)$ (so that, at decompression, one would approximate each y_j by $x_{i+j-1} - \bar{\delta}$, $1 \leq j \leq k$).

It clearly suffices to compute a *score vector* $C(1), \cdots, C(n-k)$ where $C(i) = \sum_{j=1}^{k}(x_{i+j-1} - y_j - \bar{\delta})^2$. Once one has such a vector it is a simple matter to check whether any of its entries is $\leq kD$. We next sketch an algorithm for computing the $C(i)$'s in time $O(n \log k)$. In what follows, let $S_i = k^{-1}\sum_{j=i}^{i+k-1} x_j$, and let $\bar{y} = k^{-1}\sum_{j=1}^{k} y_j$. We expand the equation defining $C(i)$ into 10 terms:

$$C(i) = C_{xx}(i)+C_{yy}(i)+C_{\bar{x}\bar{x}}(i)+C_{\bar{y}\bar{y}}(i)+2C_{x\bar{x}}(i)+2C_{y\bar{y}}(i)-2C_{xy}(i)-2C_{\bar{x}\bar{y}}(i)-2C_{\bar{x}y}(i)-2C_{x\bar{y}}(i)\ ,$$

where

$$\begin{aligned}
C_{xx}(i) &= \sum_{j=1}^{k}(x_{i+j-1})^2 \ , \quad C_{yy}(i) = \sum_{j=1}^{k}(y_j)^2 \ , \quad C_{\bar{x}\bar{x}}(i) = \sum_{j=1}^{k}(S_i)^2 \ , \quad C_{\bar{y}\bar{y}}(i) = \sum_{j=1}^{k}(\bar{y})^2 \\
C_{x\bar{x}}(i) &= S_i\sum_{j=1}^{k} x_{i+j-1} \ , \quad C_{y\bar{y}}(i) = \bar{y}\sum_{j=1}^{k} y_j \ , \quad C_{xy}(i) = \sum_{j=1}^{k} x_{i+j-1}y_j \\
C_{\bar{x}\bar{y}}(i) &= kS_i\bar{y} \ , \quad C_{\bar{x}y}(i) = S_i\sum_{j=1}^{k} y_j \ , \quad C_{x\bar{y}}(i) = \bar{y}\sum_{j=1}^{k} x_{i+j-1}\ .
\end{aligned}$$

The algorithm below computes the $C(i)$'s by computing each of the above 10 component vectors.

Algorithm TEST
Input: x_1^n and y_1^k.
Output: The score vector $C(1),\cdots,C(n-k)$, and an answer YES if any of the $C(i)$'s is $\leq kD$.

1. **begin**
2. Compute $C_{xx}(i)$, $1 \leq i \leq n-k$. This is easy to do in $O(n)$ time.
3. Compute $C_{yy}(i)$, which is independent of i, in $O(k)$ time.
4. Compute $C_{\bar{x}\bar{x}}(i) = k(S_i)^2$, $1 \leq i \leq n-k$. This takes $O(n)$ time, by the observation that once we have S_i, obtaining from it S_{i+1} takes constant time.
5. Compute $C_{\bar{y}\bar{y}}(i) = k(\bar{y})^2$, which is independent of i, in $O(k)$ time.
6. Compute $C_{x\bar{x}}(i) = S_i\sum_{j=1}^{k} x_{i+j-1} = (S_i)^2$, $1 \leq i \leq n-k$. We have already observed that computing the S_i's can be done in $O(n)$ total time.
7. Compute $C_{y\bar{y}}(i) = \bar{y}\sum_{j=1}^{k} y_j = \bar{y}^2$. This is independent of i and can be computed in constant time since we have already computed $\bar{y}$.
8. Compute $C_{xy}(i) = \sum_{j=1}^{k} x_{i+j-1}y_j$, $1 \leq i \leq n-k$. This is done as follows: Partition x_1^n into n/k chunks of size k each, call these chunks (in left to right order) $\alpha_1, \alpha_2, \cdots, \alpha_{\lceil n/k\rceil}$. Let β consist of the vector of length $2k$ obtained by first reversing y_1^k and then padding with k zeroes. The convolution product of β with $\alpha_1 * \alpha_2$ can be done in $O(k\log k)$ and contains the values of $C_{xy}(i)$ for all $i \in \{1,2,\cdots,k\}$. Similarly, the convolution product of β with $\alpha_2 * \alpha_3$ contains $C_{xy}(i)$ for all $i \in \{k+1, k+2, \cdots, 2k\}$. And so on: We do a total of n/k such convolutions in order to obtain all the $C_{xy}(i)$'s. Since each such convolution takes $O(k\log k)$ time, the total time for this step is $O((k\log k)(n/k)) = O(n\log k)$.

9. Compute $C_{\bar{x}\bar{y}}(i) = kS_i\bar{y}$, $1 \le i \le n-k$. Time: $O(n)$.

10. Compute $C_{\bar{x}y}(i) = S_i \sum_{j=1}^{k} y_j = S_i\bar{y}$, $1 \le i \le n-k$. Time: $O(n)$.

11. Compute $C_{x\bar{y}}(i) = \bar{y} \sum_{j=1}^{k} x_{i+j-1} = \bar{y}kS_i$. Time: $O(n)$.

12. Compute the $C(i)$'s from the 10 component vectors computed above, and check whether any of them is $\le D$. Time: $O(n)$.

13. **end**

The time complexity of TEST is dominated by the convolution computations, and is $O(n \log k)$. Notice that TEST is an exact algorithm and involves no approximations. It is in the way we use TEST that the approximation comes in the picture: We use it $\log \hat{k}$ times, in a forward binary search for the value $\hat{k}$ defined as the largest k for which TEST returns YES when given x_1^n and y_1^k as inputs. In other words we use TEST for $k = 2, 4, 8\ldots$ until we hit a value of k (say, k') for which TEST fails: ¿From then on TEST is used as in the standard binary search over the interval $[k'/2, k']$ (recursive halving of the search interval). It is strictly speaking inaccurate to use binary search, because TEST might fail for k and yet that does not preclude that TEST would succeed for a bigger value than k. This fortunately does not occur often, and can easily be accounted for in the implementation by checking a few additional matchings in the modified binary search. This approach turned out to be a reasonable approximation, and we adopted it in our implementation discussed in the next section.

What about the max-difference criterion, which is not accounted for in the above ? As a practical matter, we use that criterion as an additional filter (on line 12) to the output of the above convolution-based computation, which actually returns more than just a YES since it also gives all the candidate locations which correspond to the YES. Therefore we can eliminate, by using the max-difference criterion, the candidate locations that fail that criterion (if all of them are eliminated by that criterion then the YES actually becomes a NO). The time taken by this max-difference screening is $O(n)$ so long as the candidate locations being screened correspond to nonoverlapping sections of x_1^n, but even in the worst-case is guaranteed to be no more than $O(n\hat{k})$.

Note that the above is not meant as a substitute for CLD (or CSD) but rather as a faster way of performing the PREFIX computation. It could indeed be used, e.g. within CLD, as a substitute for the old, brute force PREFIX (in that case we would have $n = \Theta(N)$ and $\hat{k} = \mathcal{O}(\log N)$). We have not yet implemented the above way of performing the prefix computation.

3.3 Faster Algorithm for Hamming Distance with Additive Shifts

We show now how to compute the longest prefix when Hamming distance with additive shifts is used. Observe that one can use Fast Fourier Transform (FFT) to find the "compare-add" convolution of two strings (this is similar to the usual convolution of two sequences, except that the "product" of two symbols is 1 if they are equal and zero otherwise) in time $O(|\mathcal{A}|n\log n)$ (cf. [1]) where $|\mathcal{A}|$ is the size of the alphabet. This was used in [1] to obtain an $O(n^{1.5} polylog(n))$ time algorithm for the case when $|\mathcal{A}| = n$. Here we seek to achieve $O(n \cdot polylog(n))$ time irrespective of the size of the alphabet, so we propose below an *approximate* algorithm.

We start with the general idea behind our approach. Let $x^n = x_1 \cdots x_n$ be a *text* string and $y^m = y_1 \cdots y_m$ be a *pattern* string ($m \leq n$), both over the alphabet $\mathcal{A} = \{1, 2, \cdots, V\}$. Recall that our goal is to find: (i) the largest k such that $y_1 \cdots y_k$ *almost occurs* as a substring of x^n (here and in the following analysis the notion of "almost occur" is understood to incorporate the additive shift idea, that is, some additively shifted version of $y_1 \cdots y_k$ is close, in the Hamming distance sense, to x^n); and (ii) the position i in x^n at which it almost occurs, as well as the corresponding amount of additive shift. For given k and i, the almost occurrence of $y_1 \cdots y_k$ at position i in x^n will be determined by computing the modulus of the following function

$$F_k(i) = \sum_{j=1}^{k} e^{2\pi V^{-1}\sqrt{-1}d(x_{i-1+j}, y_j)} \tag{12}$$

where $d(\cdot,\cdot)$ is the distortion measure function. For computational reasons, we use $d(a,b) = a - b$, that is,

$$F_k(i) = \sum_{j=1}^{k} e^{2\pi V^{-1}\sqrt{-1}(x_{i-1+j} - y_j)} \ . \tag{13}$$

We claim that $y_1 \cdots y_k$ almost occurs in x^n if the modulus $|F_k(i)|$ is close to k. This is based on the following observations:

(i) The special case of $y_j = x_{i-1+j}$ for all $j \in \{1, \cdots, k\}$ implies an $F_k(i) = k$.

(ii) The special case of $y_j + c = x_{i-1+j}$ for all $j \in \{1, \cdots, k\}$, where c is a constant (the additive shift), also results in the above sum having a modulus of k and an angle of $2\pi c/V$, that is, $F_k(i) = ke^{2\pi V^{-1}\sqrt{-1}}$.

(iii) Let $y_j + C = x_{i-1+j}$ for all $j \in \{1, \cdots, k\}$, where C is random noise with a symmetric distribution function. For example, for C uniformly distributed over an interval $[c-\delta, c+\delta]$ where c and δ are constant and $\delta \leq c$, extensive experimentation has shown that when x

and y are random and δ is "small" compared to V then the modulus of the above sum is close to k. Indeed, observe that

$$
\begin{aligned}
F_k(i) &= \sum_{j=1}^{k} \left(\cos(2\pi V^{-1}C) + \sqrt{-1}\sin(2\pi V^{-1}C)\right) \\
&= \sum_{j=1}^{k} \left(1 - 2\pi^2 V^{-2} C^2 + \cdots + \sqrt{-1}(2\pi V^{-1}C - \frac{2}{3}\pi^3 V^{-3} C^3 + \cdots)\right) .
\end{aligned}
$$

Thus, $EF_k(i) \approx k$ since $EC^{2j+1} = 0$ ($j \geq 0$) by symmetry of the distribution, and $EC^{2j} \approx 0$ for $j \geq 1$ by the above assumptions.

(iv) When the modulus of the above sum is close to k, we observed through an extensive experimentation that the angle of $F_k(i)$, that is, $\arg F_k(i) =^{def} \theta$, is approximately equal to $2\pi V^{-1} \sum_{j=1}^{k}(x_{i-1+j} - y_j)$, thus $y_j \approx x_{i-1+j} - \theta V(2\pi)^{-1}$. This is not surprising since, when $|F_k(i)| \approx k$, we have $\theta_j =^{def} \arg\left(e^{2\pi V^{-1}\sqrt{-1}(x_{i-1+j}-y_j)}\right) \approx 0$. Since $\sin(\theta_i) \approx 0$ and $\cos(\theta_i) \approx 1$, we have $\arg(F_k(i)) \approx 2\pi V^{-1} \sum_{j=1}^{k}(x_{i-1+j} - y_j)$, as observed.

The above implies that, for a given k, determining whether $y_1 \cdots y_k$ almost occurs in x_n can be done by computing the $F_k(\cdot)$ vector and checking whether the modulus of $F_k(i)$ is within a factor α of k *for some* i, where the parameter α is close to 1 (say, $\alpha = 0.95$). The problem of computing the $F_k(\cdot)$ vector is easily seen to be a convolution computation: The convolution of the two n-length vector $A = a_1 \cdots a_n$ and $B = b_1, \cdots, b_k, 0, \cdots, 0$ where $a_j = e^{2\pi\sqrt{-1}V^{-1}x_j}$, and $b_j = e^{-2\pi\sqrt{-1}V^{-1}y_j}$ if $j \leq k$ and $= 0$ if $j > k$. Doing this convolution directly takes time $O(n \log n)$, but this time can be reduced to $O(n \log k)$ by doing instead n/k convolutions of $(2k)$-length vectors each (partitioing x^n as in Step 8 of algorithm `TEST` of the previous subsection, etc).

The algorithm called `APPROXIMATE PREFIX` replaces the brute force `PREFIX` algorithm: start with a small value for k (say, $k = 4$) and binary search upwards for the largest value of k (call it $\hat{k}$) for which the largest modulus among the entries of the $F_k(\cdot)$ vector is $\leq \alpha k$.

The worst case complexity of `APPROXIMATE PREFIX` is $O(n(\log \hat{k})^2)$ since the computation of each $F_k(\cdot)$ vector takes $O(n \log \hat{k})$ time, and an additional $O(\log \hat{k})$ factor comes from the binary search (see end of Sec. 3.2 for a comment on the approximate nature of such a binary search).

3.4 Suffix Tree Algorithm

Finally, we briefly mention an $O(N^2)$ approximate algorithm based on a suffix tree (cf. [2, 20]).

The main idea is quite simple. We first build a suffix tree of the whole image $N \times N$ (either using McCreight's algorithm in $O(N^2)$ time [20] or by a brute force approach in $\mathcal{O}(N^2 \log N)$ time [2]). Once the suffix tree is constructed, one can find *exact* longest prefix in a row starting at any position in $O(N)$ steps or with high probability in $O(\log N)$ steps. Searching for an approximate prefix it much more complicated and usually leads to an explosion in the search time. To circumvent this problem, we explore from every node of the suffix tree only a fixed number, say g, of paths (cf. [9]). Our preliminary experimental results indicate that such a restriction does not lead to significant deterioration of the compression ratio and quality of the image.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

This section describes in details the currently implemented version of Pattern Matching Image Compression that is based on the ideas discussed above. As we shall see several enhancements to this main idea were implemented. Results of our experimental studies of PMIC are also discussed. We conclude with a discussion of the advantages and disadvantages of PMIC method.

4.1 Implementation

The main idea of our scheme was already described in the previous sections. In short, after selecting a database – which consists of few last rows in in image – we search for the longest prefix in the uncompress image that approximately occurs in the database. In our implementation we chose quadratic distortion measure with the max-difference criterion, as already discussed in Section 3.2. Recall that this distortion between strings x_1^k and y_1^k is:

$$\begin{aligned} \bar{\delta} &= \frac{1}{k}\sum_{i=1}^{k}(x_i - y_i) \\ d(x^k, y^k) &= \frac{1}{k}\sum_{i=1}^{k}(x_i - y_i - \bar{\delta})^2 \ . \end{aligned}$$

This main idea is modified with several enhancements that make PMIC attractive. We describe them in the sequel:

Additive Shift. As mentioned in the previous section, 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. Storing c as well as a $(pointer, length)$ pair 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

$$c=\frac{1}{m}\sum_{i=1}^{m}(x_i-y_i) \ .$$

The reader can easily verify that for $x_i = y_i + c$, and c a constant, the above procedure returns c.

Reverse Prefix. It turns out that in an image there are substantial similarities between a substring in the uncompressed file and *reverse* substring in the database. That is, we search for the largest prefix of y^n that occurs approximately in x_n^1 (scanned backward).

Variable Maximum Distortion. It is well known that human eyes can easily distinguish an "undesired" pattern in a low frequency (constant) background while the same pattern might be almost invisible to human eyes in high frequency (quickly varying) background (cf. [7, 13]). 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. To recognize these two different situations, we review the notion of the *derivative* $\mathcal{D}$ of an image $\mathcal{I}$. It is defined in a natural way as the average difference between a pixel and its down and right pixels. More precisely, the derivative $\mathcal{D}_{ij}$ of pixel x_{ij} at position (i,j) is a scalar computed as

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

In our implementation, we used the lower value of D whenever $\mathcal{D}_{ij}\le 3$ and the higher value of D otherwise. In Figure 3 we compared the quality of the compression for constant D and variable D (with two values of D). In Figure 3(a) a constant D was used, and one can easily recognize the distortion around the cloud on the slowly varying sky. This problem disappears in Figure 3(b) where a variable D is used.

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.

We are finally able to describe the compression code which is also presented in Figure 4. It consists of:

(a) (b)

Figure 3: Comparison of the "San Francisco" image compressed by PMIC: (a) with constant D and compression ratio 7.5; (b) variable D and compression ratio 8.3.

- 1 bit (C/F) to indicate whether the next 4 bytes are copied from the image or a $(position, length)$ pointer pair,
- 1 bit (F/R) to indicate whether we store forward or reverse prefix,
- variable length pointer to the position in the database or copy from the database depending whether $C/F = 1$ or 0,
- variable length of the longest prefix,
- additive shift c described above.

Optionally, we can also store the following information if the run-length code is used

- 1 bits (Rn) whether run length code is used or not,
- repeated pixel,
- number of repetitions.

From the above description, one can easily see that the decompression algorithm is very simple and amounts most of the time to reading a value and writing it elsewhere (with, occasionally, an additive shift c). No inverse transforms, no multiplications – it is indeed a very simple decompression algorithm.

C/P	F/R	copy or pointer to database	length of longest prefix	shift	Rn	repeated byte	number of repetitions

Figure 4: Structure of the compression code

4.2 Experimental Results

In order to assess the quality of our PMIC scheme, we performed a series of experiments, and compared PMIC with the standard JPEG (UNIX implementation), wavelet compression (cf. [8]), and also fractal compression (implemented as described in [10]).

Before presenting our experimental results, one must decide about metrics of comparison. It is natural to use (real) compression (scale) = (size of the original file)/(size of the compressed file) as one metric. We also use the compression ratio r in bpp (bits per pixel), as defined in Section 2.

There is, however, more controversy on how to measure the quality of compression (cf. [13, 22]). In Section 2 we discussed a measure of distortion $d(\cdot,\cdot)$ which usually for image compression becomes square-root measure or *root-mean-squared-error* (RMSE) defined as

$$RMSE = \sqrt{\frac{1}{N^2}\sum_{i,j=1}^{N}(x_{ij}-\hat{x}_{ij})^2}$$

where $\hat{x}$ represents the compressed/decompressed image. In fact, traditionally this measure is redefined as

$$PSNR = 20\log_{10}\left(\frac{255}{RMSE}\right) .$$

One also observes that another possible measure is simply the *average-gray-level-error* (AGLE), that is

$$AGLE = \frac{1}{N^2}\sum_{i,j=1}^{N}|x_{ij}-\hat{x}_{ij}| .$$

It is well known (cf. [13]), and cofirmed by our experiments, that either all the above measures were doing well or none of them was good. For example, in our experimental study, the latter situation occured when comparing "variable-D" versus "constant-D" methods. In Figure 3 both images scored almost the same RMSE/PSNR/AGLE while visual comparison clearly indicates that "variable-D" method is much better. In such a situation, we either must resort to sophisticated visual tests such as Contrast Sensitivity Function (CSF) (cf.

[7]) or use another measure of comparison. We choose the latter, which is basically a weighted L^b norm for some $b > 0$ (cf. [13]).

We first observe that visual distortion is most visible for low frequency (slowly varying) background. This suggests the introduction of *weights* that are inversely proportional to the derivative $\mathcal{D}_{ij}$ at pixel (i, j). Let

$$W_{ij} = \begin{cases} \mathcal{D}_{ij}^{-1} & \mathcal{D}_{ij} \neq 0 \\ Z & \mathcal{D}_{ij} = 0 \end{cases}$$

where Z is a large number such that $Z \gg \max_{\mathcal{D}_{ij} \neq 0} \{\mathcal{D}_{ij}^{-1}\}$. Then, for a parameter b we define

$$q_b(x, \hat{x}) = \left(\sum_{ij=1}^{N} w_{ij} (x_{ij} - \hat{x}_{ij})^b \right)^{1/b}$$

where

$$w_{ij} = \frac{W_{ij}}{\sum_{i,j=1}^{N} W_{ij}} \ .$$

Observe that for $b \to \infty$ we obtain

$$q_\infty(x, \hat{x}) = \max_{ij} \{x_{ij} - \hat{x}_{ij}\} \ .$$

In most of our comparisons, we use the simplest "average grey level error" AGLE.

Now, we are in position to present some of our images and compare them with other methods of compression. Since PMIC compression systematically was superior to a fractal compression (cf. Figure 7) we concentrate here on comparing PMIC with JPEG and wavelet compression. In Figure 5 the "San Francisco" image is shown. It has two features interesting to us: First, the sky has almost constant (low frequency) background. Secondly, buildings have sharp edges (high frequency). In Figure 3 we already investigated the former problem, and concluded that for such images adaptive level of distortion (i.e., variable-D) is needed to obtain high quality picture. The latter problem is better shown in Figure 6 where a magnified fragment of the "San Francisco" image is presented.

From these three pictures (Figures 3, 5 and 6) we conclude the following. While Figure 5 suggests that the overall quality of JPEG, wavelet compression and PMIC is almost the same, Figure 6 indicates that PMIC is quite superior for images containing substantial amounts of high frequencies. On the other hand, we observe that one has to use adaptive level of maximum distortion to obtain a good quality image. Finally, from Figure 2 we conclude by a visual test that `COMPRESS_SHORT_DATABASE` does not deteriorate significantly the quality of the picture, while decreasing by a factor of 25 the compression time.

(a)

(b)

(c)

(d)

Figure 5: Comparison of "San Francisco" image compression: (a) Original picture; (b) JPEG with the average gray-level error (AGLE) equal to 4.6 and compression ratio equal to 8.5 ($\sim$ 1 bpp); (c) Wavelet based compression with AGLE=3.56 and compression equal to 6.82 (1.2 bpp); (d) PMIC with AGLE equal to 6.4 and compression ratio equal to 8.3 ($\sim$ 1 bpp).

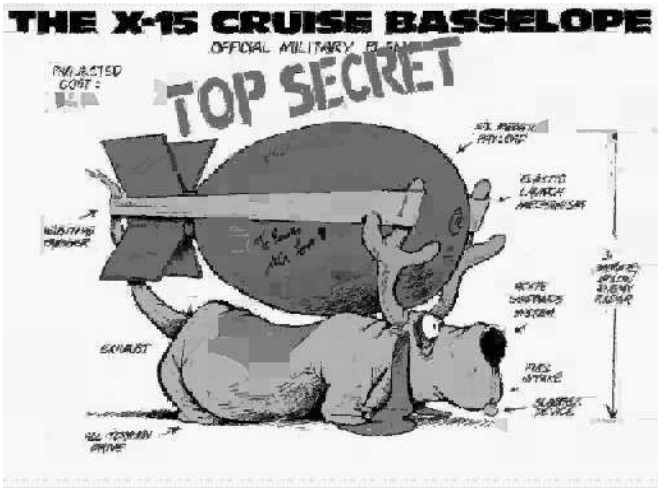
(a) (b)

Figure 6: Comparison of a fragment of the "San Francisco" image: (a) JPEG; (b) Pattern Matching Image Compression (PMIC).

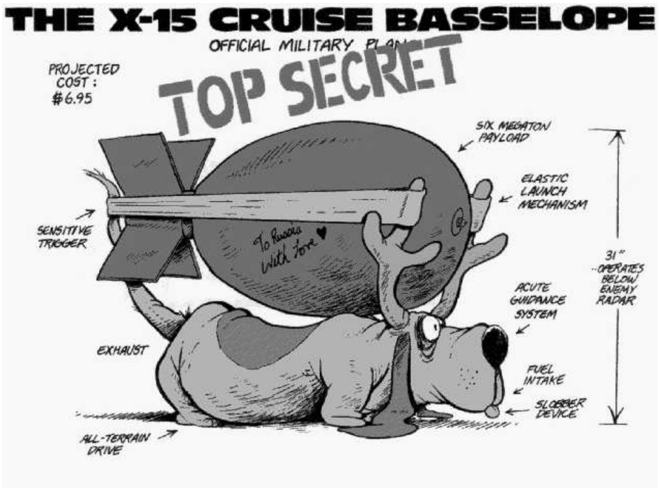
So far, we discussed only "photographic" images. It turns out that PMIC works particularly well for "graphic" images like Figure 7, which are well suited for run-length encoding. In addition, in Figure 8 we plotted a more extensive comparison graph where compression ratio r (in bpp) versus average gray-level error AGLE is presented and compared with JPEG and the wavelet compression approach. Our conclusion is that PMIC is competitive with JPEG and wavelet based compression, and on many instances superior.

5. CONCLUDING REMARKS

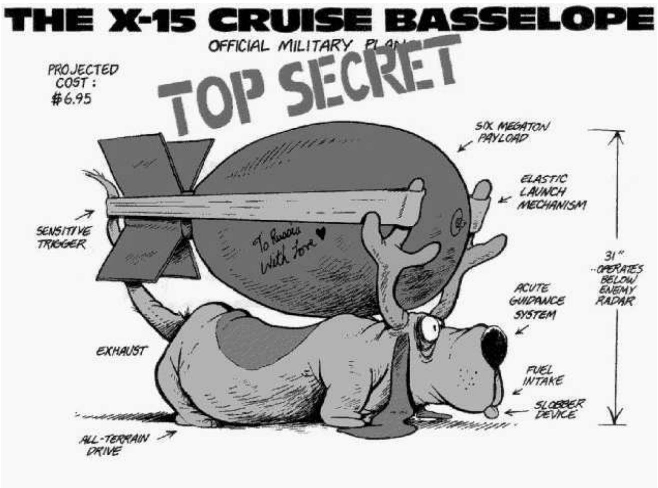
In this paper we described an experimental image compression called Pattern Matching Image Compression (PMIC) that is non-transform technique for processing images. It is based on an approximate pattern matching and lossy extension of the Lempel-Ziv scheme. To the best of our knowledge, it is the first image compression method for which one can prove guaranteed performance metrics, that is, theoretical compression ratio versus maximum measure of distortion. In Section 2 we indicated that for the basic scheme the compression ratio achieves the rate of the generalized Rényi entropy $r_0(D)$ introduced in [19]. While $r_0(D)$ is greater than the optimal rate-distortion function $R(D)$, numerical evaluation (cf. Figure 1) indicates that the difference is not too big for small values of



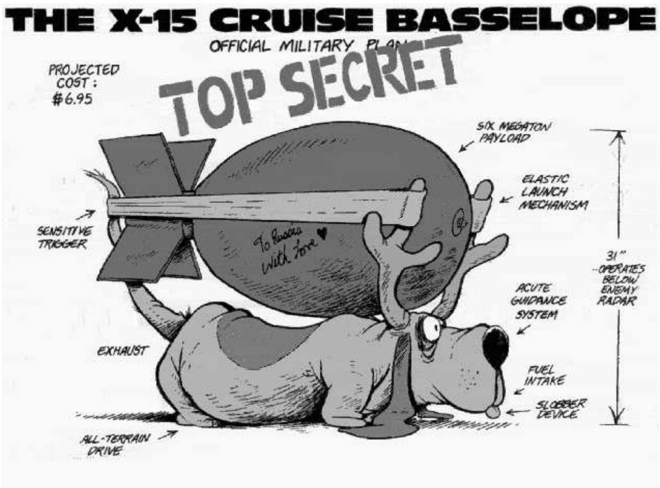
(a)



(b)



(c)



(d)

Figure 7: Comparison of image compression of the "Basselope": (a) Fractal compression with an average gray-level error AGLE=2 and compression 5.5 (1.45 bpp); (b) Wavelet based compression with AGLE=2.66 and compression 6.14 (1.3 bpp); (c) JPEG with AGLE=4.3 and compression 7.8 (1 bpp); (d) Pattern Matching Image Compression (PMIC) with AGLE=2.9 and compression 7.8 (1 bpp).

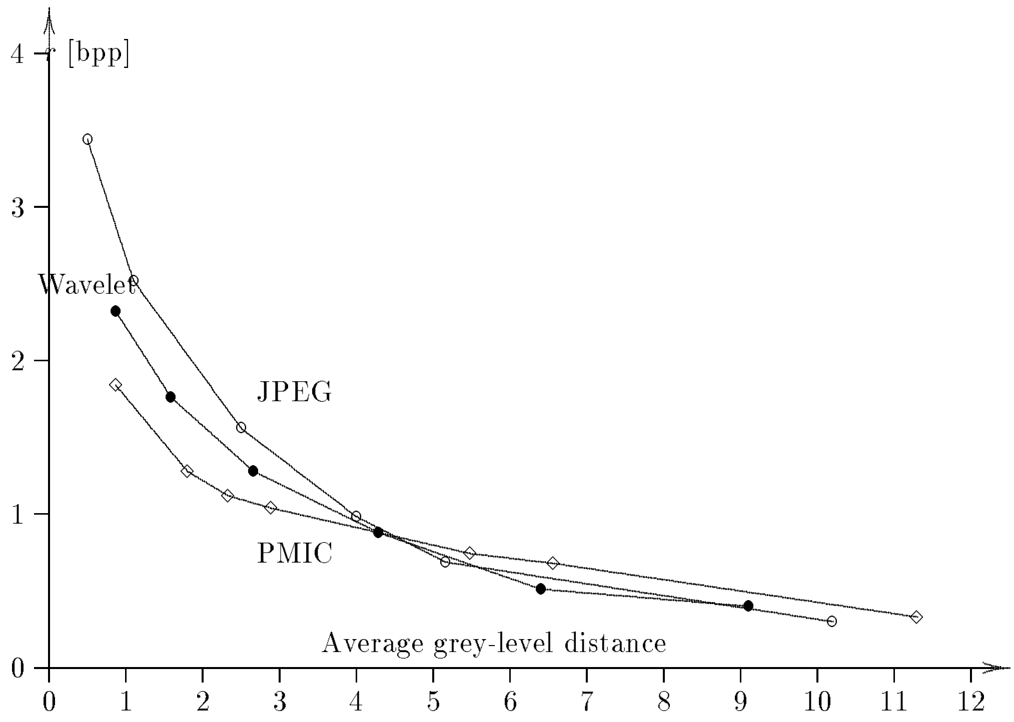


Figure 8: Compression ratio r [bpp] versus the average greay-level error (AGLE) for JPEG ($\circ$), wavelet transform ($\bullet$), and PMIC ($\diamond$) of the "Basselope" image

D. More importantly, such a suboptimal lossy compression is computationally efficient. In Section 3 we presented several algorithms for image compression.

Finally, the actual implementation of PMIC with several enhancements was discussed in Section 4. Experimental image compressions shown in that section suggest that:

- PMIC is slower in the compression time but the fastest possible in the decompression time,
- Overall quality of compression is good and competitive with JPEG and wavelet based compression; PMIC compression works better when high frequency components dominate the image,
- PMIC is systematically better than fractal compression in the compression time and quality,
- PMIC is an automatic image compression, in the sense that its parameters adjust adaptively during the compression.

While so far results are promising, more research in this area is needed. In fact, compression based on exact and approximate pattern matching is not restricted to image compression. It is obviously good for text compression, but we believe we have identified a scheme for audio compression. This will be reported in a forthcoming paper.

ACKNOWLEDGEMENT

The authors are grateful to Professor Bradley Lucier for sharing the wavelet compression data with them. We also thank Professor Zygmunt Pizlo for valuable discussions regarding visual tests.

References

[1] K. Abrahamson, Generalized String Matching, *SIAM J. Comput.*, 16, 1039-1051, 1987.

[2] A. Apostolico and W. Szpankowski, Self-Alignments in Words and Their Applications, *J. Algorithms*, 13, 446-467 (1992).

[3] R. Arratia and M. Waterman, The Erdös-Rényi Strong Law for Pattern Matching with Given Proportion of Mismatches, *Annals of Probability*, 17, 1152-1169 (1989).

[4] M. Atallah, P. Jacquet and W. Szpankowski, Pattern matching with mismatches: A probabilistic analysis and a randomized algorithm, *Proc. Combinatorial Pattern Matching*, Tucson, LNCS 644, 27-40, Springer-Verlag 1992.

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

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

[7] T. Cornsweet, *Visual Perception*, Academic Press, New York (1970).

[8] R. DeVore, B. Jawerth, and B. Lucier, Image Compression Through Wavelet Transform Coding, *IEEE Trans. Information Theory*, 38, 719-746 (1992).

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

[10] Y. Fisher, E. Jacobs, and R. Boss, Fractal Image Compression Using Iterated Transforms, in *Image and Text Compression* (ed. J. Storer), 35-62, Kluwer Academic Publishers, Boston (1992).

[11] Z. Galil and R. Giancarlo, Data Structures and Algorithms for Approximate String Matching, *J. Complexity*, 4, 33-72, (1988).

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

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

[14] J.C. Kieffer, A Survey of the Theory of Source Coding with a Fidelity Criterion, *IEEE Trans. Information Theory*, 39, 1473-1490 (1993).

[15] J.C. Kieffer, Private Correspondence.

[16] H. Koga and S. Arimoto, Asymptotic Properties of algorithms of Data Compression with Fidelity Criterion Based on String Matching, *Proc. 1994 IEEE Information Symposium on Information Theory*, 24-25 (1994).

[17] Knuth, D.E., *The Art of Computer Programming. Seminumerical Algorithms*, Vol. 2, Addison-Wesley (1981).

[18] 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).

[19] 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).

[20] E.M. McCreight, A Space Economical Suffix Tree Construction Algorithm, *JACM*, 23, 262–272 (1976).

[21] D. Ornstein and P. Shields, Universal Almost Sure Data Compression, *Annals of Probability*, 18, 441-452 (1990).

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

[23] I. Sadeh, On Approximate String Matching, *Proc. of Data Compression Conference*, 148-157 (1993).

[24] 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).

[25] W. Szpankowski, A Typical Behavior of Some Data Compression Schemes, *Proc. of Data Compression Conference*, Snowbird, 247-256 (1991).

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

[27] 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).

[28] E.H. Yang, and J. Kieffer, Simple Universal Lossy Data Compression Schemes Derived From Lempel-Ziv algorithm, preprint (1995).

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

[30] Z. Zhang and V. Wei, An On-Line Universal Lossy Data Compression Algorithm via Continuous Codebook Refinement, preprint.

[31] J. Ziv, Distortion-rate Theory for Individual Sequences, *IEEE Trans. Information Theory*, 26, 137-143 (1980).

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

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