2D-Pattern Matching Image and Video Compression:
Theory, Algorithms, and Experiments

May 5, 2000

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 propose a lossy data compression framework based on an approximate two dimensional pattern matching (2D-PMC) extension of the Lempel-Ziv lossless scheme. This framework forms the basis upon which higher level schemes relying on differential coding, frequency domain techniques, prediction, and other methods can be built. We apply our pattern matching framework to image and video compression and report on theoretical and experimental results. Theoretically, we show that the *fixed database model* used for video compression leads to suboptimal but computationally efficient performance. The compression ratio of this model is shown to tend to the generalized entropy. For image compression we use a *growing database* model for which we provide an approximate analysis. The implementation of 2D-PMC is a challenging problem from the algorithmic point of view. We use a range of 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 $0.15 - 0.5$ Mbps for a baseline video compression scheme that does not use any prediction or interpolation. We also demonstrate that this asymmetric compression scheme is capable of extremely fast decompression making it particularly suitable for networked multimedia applications.

Index Terms: Multimedia compression, approximate pattern matching, Lempel-Ziv schemes, rate distortion, generalized Shannon entropy, k-d trees, generalized run-length coding, arithmetic coding.

*This work was supported by NSF Grants NCR-9415491 and C-CR-9804760, and Purdue GIFC Grant.
†This work was supported by NSF Grants ACI-9872101 and EIA-9806741.

1 Introduction

Lossless data compression based on *exact* pattern matching can be traced back to seminal papers of Lempel and Ziv [26, 27]. More recently, there has been a resurgence of interest in extending these schemes to lossy data compression (cf. [2, 3, 4, 5, 11, 13, 15, 17, 20, 21, 23, 24]). 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 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 [4]. However, no theoretical justification was provided for this scheme and experimental results were preliminary. The key issue of whether such a scheme is asymptotically optimal (like its lossless Lempel-Ziv counterpart) was first addressed by Steinberg and Gutman [17], and eventually resolved independently by Łuczak and Szpankowski [15] and Yang and Kieffer [23]. 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 *generalized Shannon entropy*; which is higher than the optimal rate distortion function. More recently, Kontoyiannis [13] proposed a scheme based on approximate pattern matching that is asymptotically optimal for memoryless sources, but its practical applicability is as yet unclear. Other optimal schemes have been proposed by Zamir and Rose [24] and Zhang and Wei [25], however, the *golden-washing scheme* of Zhang and Wei [25] does not seem to work very well in practice.

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 [15, 23], Atallah, Génin and Szpankowski [3] 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, the PMIC scheme is competitive with JPEG and wavelet image compression up to 1 bpp. However, the superiority of PMIC at 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 for image compression, while for general multimedia compression we coin the term 2D– Pattern Matching Compression (2D-PMC). We also use this idea to develop a new compression scheme for video data that uses the previous frame as the database. Our preliminary experimental results exceed even our expectations: we obtain high quality video with bit rates of $0.15 - 0.5$ Mbps. We should underline that this scheme does not use any prediction, interpolation, differential coding, frequency domain transforms, or quantizations. When combined with all these other optimizations, the pattern matching framework is expected to yield even higher compression rates. We report on our theoretical and experimental findings in this paper. The reader is referred to `http://www.cs.purdue.edu/homes/spa/Compression/2D-PMC.html` for more image and video compression samples and results.

In this paper, we first discuss the theoretical underpinnings and subsequently present algorithmic and experimental results. Our video compression scheme is modeled well by the *fixed database model* (cf. [13, 20, 23]). In this model, the first frame serves as a fixed database (reference) for compressing subsequent frames 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 memoryless sources these two rates do not differ by much (cf. [15]). 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. Precise asymptotic analysis of this scheme is beyond the current state-of-the-art. However, we formulate rigorously the problem, propose an approximate analysis, and present some experimental results.

The implementation of 2D-PMC poses a challenging problem from the algorithmic standpoint. Finding an efficient data structure for *approximate* search of multidimensional sets in large databases 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.

The rest of this paper is organized as follows: Section 2 presents theoretical results for the fixed database model and a heuristic analysis for the growing database model. In Section 3 we turn our attention to algorithmic issues and discuss salient features of our implementation. Finally, in Section 4, we present some experimental results for image and video compression and conclude with a brief list of advantages and drawbacks of the proposed compression scheme.

2 Theoretical Underpinnings

Consider a source sequence $\{X_k\}_{k=1}^{\infty}$ taking values from a finite alphabet $\mathcal{A}$ (e.g., $|\mathcal{A}| = 256$). We write $X_m^n = X_m X_{m+1} \dots X_n$, and $P(x_1^n)$ for the probability of $X_1^n = x_1^n$. We encode X_1^n into a compression code and the decoder produces an estimate $\hat{X}_1^n$ of X_1^n. More precisely, a code $\mathscr{C}_n$ is a mapping $\mathscr{C}_n : \mathcal{A}^n \to \{0,1\}^*$, and we write $\mathscr{C}_n(x_1^n)$ for the compression code of 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 $\hat{x}_1^n = \psi(\mathscr{C}_n(x_1^n))$. Let $\ell(\mathscr{C}_n(x_1^n))$ be the length of a code (in bits) representing x_1^n. Then, the *bit rate* is defined as $r(x_1^n) = \ell(\mathscr{C}_n(x_1^n))/n$, the *average* bit rate as $\mathbf{E}[r(X_1^n)] = \mathbf{E}[\ell(\mathscr{C}_n(X_1^n))]/n$, and the *compression ratio* CR as $CR = n \log_2 |\mathcal{A}|/\ell(\mathscr{C}_n(X_1^n))$.

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

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

or more generally, a subadditive distortion measure satisfying, for any two integers n, m, and given vectors x_1^{n+m}, y_1^{n+m},

$$d(x_1^{n+m}, y_1^{n+m}) \le \frac{n}{n+m} d(x_1^n, y_1^n) + \frac{m}{n+m} d(x_{n+1}^{n+m}, y_{n+1}^{n+m}) \ .$$

Examples of fidelity measures satisfying the above criteria include the *Hamming distance* and the *squared error distortion.* The Hamming distance, $d(x_i, \hat{x}_i)$ equals zero if $x_i = \hat{x}_i$ and one otherwise; and in the squared error distortion, $d(x_i, \hat{x}_i) = (x_i - \hat{x}_i)^2$. Our original PMIC was based on the squared error distortion, which is natural for image compression. In the current implementation we develop several novel distortion measures that adjust their maximum value to produce the best possible perceptual results (cf. Section 3.2).

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 $\hat{X}_1^n$ (e.g., the first image in the video stream) generated according to the distribution $\hat{P}$. The source sequence X_1^M (e.g., $M = N^2$ for $N \times N$ images) is partitioned according to Π_n into variable length phrases (e.g., rectangles in 2D-PMC) $Z^1, Z^2, \dots, Z^{|\Pi_n|}$ of lengths $L_n^1, \dots, L_n^{|\Pi_n|}$, respectively. To simplify presentation, we start with the one dimensional case, and subsequently provide bounds for the multi-dimensional case.

For given fixed distortion bound $D > 0$ (to simplify the presentation all our definitions refer to the one-dimensional case)

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

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, if one sets $K_n(m) = \sum_{i=1}^m L_n^i$, the $(m+1)$-st phrase of partition Π_n is defined as:

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

Thus, the source sequence is partitioned according to Π_n as $X_1^M = Z^1 Z^2 \dots Z^{|\Pi_n|}$ while the decoder recovers the string $\hat{Z}^1 \hat{Z}^2 \dots \hat{Z}^{|\Pi_n|}$ 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(s) hence its description costs $\log n + O(\log(L_n^i))$ bits. The total compressed code length is

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

and the bit rate (e.g., in bits per pixel) is given by

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

To formulate our main result, we introduce the generalized Shannon entropy $\hat{r}_0(D)$ defined as (cf. [15])

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

where $B_D(x_1^n) = \{y_1^n : \ d(y_1^n, x_1^n) \le D\}$ is a ball of radius D with center x_1^n, and $\mathbf{E}_P$ denotes the expectation with respect to P.

We prove below that the one-dimensional pattern matching compression scheme achieves, asymptotically, the bit rate $\hat{r}_0(D)$. This provides a rigorous justification for the 1D-PMIC compression scheme implemented in [3]. Following the theorem, we comment on the 2D-PMC scheme. In particular, we show that the bit rate for 2D-PMC lies between $\hat{r}_0(D)$ and $2\hat{r}_0(D)$ (however, we conjecture that the lower bound is the correct bit rate).

Theorem 1 *Let us consider the one-dimensional fixed database model with the database $\hat{X}_1^n$ generated by a Markovian source $\hat{P}$ (with a positive transition matrix) and the source sequence X_1^M emitted by an independent Markovian source P (also with a positive transition matrix).*

(i) *The length L_n^1 of the first phrase satisfies*

$$\lim_{n\to\infty} \frac{L_n^1}{\log n} = \frac{1}{\hat{r}_0(D)}, \qquad \text{(pr.)}. \tag{3}$$

More precisely, for any $\varepsilon > 0$

$$\Pr\left\{(1-\varepsilon)\frac{1}{\hat{r}_0(D)}\log n \le L_n^1 \le (1+\varepsilon)\frac{1}{\hat{r}_0(D)}\log n\right\} = 1 - O\left(\frac{\log n}{n^\varepsilon}\right). \tag{4}$$

(ii) *The average bit rate attains asymptotically*

$$\lim_{n\to\infty}\lim_{M\to\infty} \mathbf{E}_{P\times\hat{P}}[r_n(X_1^M)] = \hat{r}_0(D). \tag{5}$$

Proof. We observe that (4) follows directly from [15] for some mixing sequences, and thus it applies to our situation. We use it in establishing lower and upper bounds for (5).

We start with the upper bound. We adopt the approach of Wyner and Ziv [21] and its simplification proposed by Kontoyiannis [13]. Let us partition all phrases into " long" phrases and "short" phrases. A phrase i is long if $L_n^i \ge (1-\varepsilon)\log n$ for any $\varepsilon > 0$; otherwise it is a short phrase. Let $\mathcal{L}_M$ and $\mathcal{S}_M$ be the sets of long and short phrases, respectively. Observe that

$$N = |\mathcal{L}_M| \leq \frac{M(\hat{r}_0(D))}{(1-\varepsilon)\log n}. \tag{6}$$

Now we proceed as follows (below c is a constant that can change from line to line):

$$\begin{aligned}
r_n(X_1^M) &\leq \frac{1}{M}\sum_{i\in\mathcal{L}_M}(\log n + c\log L_n^i) + \frac{1}{M}\sum_{i\in\mathcal{S}_M}(\log n + c\log L_n^i) \\
&\stackrel{(6)}{\leq} (1+\varepsilon)\hat{r}_0(D) + c\frac{N}{M}\sum_{i\in\mathcal{L}_M}\frac{1}{N}\log L_n^i + \frac{1}{M}\sum_{i\in\mathcal{S}_M}(\log n + c\log L_n^i) \\
&\stackrel{\text{Jensen}}{\leq} (1+\varepsilon)\hat{r}_0(D) + c\frac{N}{M}\log\left(\frac{1}{N}\sum_{i\in\mathcal{L}_M}L_n^i\right) + \frac{1}{M}\sum_{i\in\mathcal{S}_M}(\log n + c\log L_n^i) \\
&\leq (1+\varepsilon)\hat{r}_0(D) + c\frac{N}{M}\log\left(\frac{M}{N}\right) + \frac{1}{M}\sum_{i\in\mathcal{S}_M}(\log n + c\log L_n^i) \\
&\leq (1+\varepsilon)\hat{r}_0(D) + \frac{c\log\log n}{\log n} + \frac{1}{M}\sum_{i\in\mathcal{S}_M}(\log n + c\log L_n^i).
\end{aligned}$$

We now evaluate the expected value of the third term above We shall write below $\widetilde{L}_n^1$ for a match (phrase) that can occur in any position. (Observe that $\widetilde{L_n^1} \stackrel{d}{=} L_n^1$, that is, $\widetilde{L}_n^1$ has the same distribution as L_n^1.) Below, by $I(\cdot)$ we denote the indicator function.

$$\begin{aligned}
\mathbf{E}\left[\frac{1}{M}\sum_{i\in\mathcal{S}_M}(\log n + c\log L_n^i)\right] &\stackrel{(6)}{\leq} \frac{c}{M}\log n\ \mathbf{E}\left[\sum_{i=1}^{|\Pi_n|} I(L_n^i \text{ is short})\right] \\
&\leq \frac{c}{M}\log n\ \mathbf{E}\left[\sum_{i=1}^{M} I(\widetilde{L}_n^1 \text{ is short})\right] \\
&\leq c\log n\ \Pr\left\{L_n^1 \leq (1-\varepsilon)\frac{1}{\hat{r}_0(D)}\log n\right\} \\
&\stackrel{(4)}{\leq} c\frac{\log^2 n}{n^{\varepsilon}}
\end{aligned}$$

where the second inequality above follows by considering not just i but all possible positions on X_1^M where a short match can occur.

We now deal with the lower bound. Let $\mathcal{H}_M$ be the set of those phrases whose length is not bigger than $(1+\varepsilon)\frac{1}{\hat{r}_0(D)}\log n$. Clearly

$$\mathbf{E}[r_n(X_1^M)] \geq \frac{1}{M}\mathbf{E}\left[\sum_{\mathcal{H}_M}\log n\right] = \frac{\log n}{M}\mathbf{E}[|\mathcal{H}_M|]$$

where $|\mathcal{H}_M|$ is the cardinality of $\mathcal{H}_M$. Thus it suffices to prove that

$$\mathbf{E}[|\mathcal{H}_M|] \geq \hat{r}_0(D)\frac{M(1-o(1))}{(1+\varepsilon)\log n} \tag{7}$$

to establish the desired lower bound. We start with the following known fact (cf. [1, 15, 18, 19, 22, 23]: For any $\delta > 0$, we have for all n

$$\Pr\{L_n^1 > n^\delta\} \le n \exp(-An^\delta) \tag{8}$$

where $A > 0$ is a positive constant. Indeed, the above is a consequence of $\Pr\{L_n^1 > k\} \le n\mathbf{E}[\hat{P}(B_D(X_1^k))] \le n\rho^k$ for some $\rho < 1$ where ρ is related to the generalized Rényi entropy of the first order (cf. [1, 15, 18].) Now, let $\bar{\mathcal{H}}_M$ be the complementary set to $\mathcal{H}_M$. We call phrases in $\bar{\mathcal{H}}_M$ as "very long". Observe that

$$M \le |\mathcal{H}_M| \frac{1}{\hat{r}_0(D)}(1+\varepsilon)\log n + \sum_{i\in\bar{\mathcal{H}}_M} L_n^i \tag{9}$$

Taking the expectation of both side of the above and setting $\delta = \varepsilon/2$ in (8), we obtain

$$\begin{aligned}
M \quad & \le \quad \mathbf{E}[|\mathcal{H}_M|]\frac{1}{\hat{r}_0(D)}(1+\varepsilon)\log n + \mathbf{E}\left[\sum_{\{i:\ i\in\bar{\mathcal{H}}_M\ \&\ L_n^i\le n^{\varepsilon/2}\}} L_n^i + \sum_{\{i:\ L_n^i > n^{\varepsilon/2}\}} L_n^i\right] \\
& \le \quad \mathbf{E}[|\mathcal{H}_M|]\frac{1}{\hat{r}_0(D)}(1+\varepsilon)\log n + n^{\varepsilon/2}\mathbf{E}\left[\sum_{i=1}^M I(\widetilde{L}_n^1 \text{ is very long})\right] \\
& + \quad n\mathbf{E}\left[\sum_{i=1}^M I(\widetilde{L}_n^1 > n^{\varepsilon/2})\right] \\
& \le \quad \mathbf{E}[|\mathcal{H}_M|]\frac{1}{\hat{r}_0(D)}(1+\varepsilon)\log n + n^{\varepsilon/2} M \Pr\left\{L_n^1 > (1+\varepsilon)\frac{1}{\hat{r}_0(D)}\log n\right\} \\
& + \quad nM\Pr\{L_n^1 > n^{\varepsilon/2}\} \\
& \overset{(4)\ \&\ (8)}{\le} \quad \mathbf{E}[|\mathcal{H}_M|]\frac{1}{\hat{r}_0(D)}(1+\varepsilon)\log n + M\frac{\log n}{n^{\varepsilon/2}} + n^2 M e^{-An^{\varepsilon/2}}.
\end{aligned}$$

This establishes (7), and completes the proof of the theorem for a one-dimensional fixed database model. ■

Now, we are in a position to comment on the two-dimensional fixed database model. The main difference lies in the fact that phrases in the two-dimensional model may overlap. We shall see that this only effects the upper bound. More precisely, let us carefully investigate the three pillars of the one-dimensional analysis, namely, (4), (6) and (7). It is not difficult to see that (4) is valid in higher dimensions as long as L_n^i is interpreted as volume (e.g., number of pixels) of the repeated pattern. The lower bound on $\mathbf{E}[|\mathcal{H}_M|]$ (cf. (7)) was based on (9) which is even more true when overlapping occurs. Unfortunately, the upper bound (6) is not valid. Nevertheless, in all our 2D implementations phrases cannot overlap more than twice. Therefore, we can replace (6) with

$$2M \ge \sum_{i\in\mathcal{L}_M} L_n^i + \sum_{i\in\mathcal{S}_M} L_n^i \ge N\frac{1}{\hat{r}_0(D)}(1-\varepsilon)\log n.$$

This implies the upper bound of $2\hat{r}_0(D)$ for the bit rate. In general, for the k-dimensional fixed database model the bit rate $r_n(X_1^M)$ can be bounded as

$$\hat{r}_0(D) \leq r_n(X_1^M) \leq k\hat{r}_0(D)$$

for $n, M \to \infty$. We conjecture that the lower bound is the correct bit rate.

We should also point out that $\hat{r}_0(D) \geq R(D)$ where $R(D)$ is the optimal rate distortion function. In [15] we observe, at least for memoryless sources, that $\hat{r}_0(D)$ and $R(D)$ do not differ by much for moderate values of D.

2.2 Growing Database Model

In contrast to video compression where a reference frame acts as the database, image compression uses the most recent partial compressed image as a (variable) database. This can be modeled by the *growing database model.* Let us assume that the source sequence $\{X_k\}_{k=1}^{\infty}$ is generated according to a distribution P. We set the initial reproduction sequence of length m to be the same as the first m symbols of the source, 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 $L_{n_0}^1$ as follows

  $$L_{n_0}^1 = \max\{t: \; d(X_{n_0+1}^{n_0+t}, \hat{X}_i^{i+t-1}) \leq D, \;\; 1 \leq i \leq n_0 - t + 1\}.$$

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

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

  where $\star$ implies concatenation. We sent $\hat{X}^1$ in a lossless fashion to the decoder. Observe that we do *not* concatenate $X_{m+1}^{n+L_{n_0}^1}$ but the *distorted* database $\hat{X}_{\mathtt{ptr}(1)}^{\mathtt{ptr}(1)+L_{n_0}^1-1}$. In this case, $d(X_1^{m+L_{n_0}^1}, \hat{X}_1^{m+L_{n_0}^1}) \leq D$, as required.

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

  $$L_{n_k}^{k+1} = \max\{t: \; d(X_{n_k+1}^{n_k+t}, \hat{X}_i^{i+t-1}) \leq D, \;\; 1 \leq i \leq n_k - t + 1\},$$

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

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

that we also reveal in a lossless transmission to the decoder. As before, by finding the largest match in the *distorted* database, we assure that

$$d(X_1^{n_k}, \hat{X}^k) \le D.$$

for given $D > 0$.

Our goal is to estimate the *operational bit rate* at step k defined as

$$r_m(X_m^{n_k}) = \frac{\log n_0 + \log n_1 + \cdots + \log n_{k-1} + \Theta(\log L_{n_0}^1) + \cdots + \Theta(\log L_{n_{k-1}}^k)}{L_{n_0}^1 + L_{n_1}^2 + \cdots + L_{n_{k-1}}^k}. \tag{10}$$

Observe that in the growing database model consecutive databases $\hat{X}^0, \hat{X}^1, \ldots, \hat{X}^k$ are non-stationary with, say, probability measures $\hat{P}^0, \hat{P}^1, \ldots, \hat{P}^k$ provided they exist. We present a heuristic analysis of the growing database model under the below hypothesis [H] that must be verified rigorously if one hopes for a precise analysis of the model:

[H] At step k of the algorithm, let the database size be n_k with underlying probability $\hat{P}^k$. There exists a constant $\hat{r}_0^k(D)$ that depends on $\hat{P}^k$ such that in probability and on average

$$L_{n_k}^{k+1}(\hat{P}^k) \sim \frac{1}{\hat{r}_0^k(D)} \log n_k \tag{11}$$

for $n_k \to \infty$.

From the description of the algorithm, we have

$$\begin{aligned} n_0 &= m, \\ n_k &= n_{k-1} + L_{n_{k-1}}^k(\hat{P}^{k-1}), \qquad k \ge 1. \end{aligned}$$

hence after using [H] we arrive at

$$\begin{aligned} \log n_k &= \log m + O\left(\frac{k \log m}{m}\right), \\ L_{n_k}^{k+1}(\hat{P}^k) &= \frac{1}{\hat{r}_0^k(D)} \log m + O\left(\frac{k \log m}{m}\right) \end{aligned}$$

as long as $m \to \infty$. In view of this and (10), we obtain

$$r_m(X_m^{n_k}) = \frac{k}{\frac{1}{\hat{r}_0^0(D)} + \frac{1}{\hat{r}_0^1(D)} + \cdots + \frac{1}{\hat{r}_0^{k-1}(D)}} + O\left(\frac{k \log\log m}{\log m}\right).$$

Finally,

$$\mathbf{E}[r_m(X_m^{n_k})] \approx \frac{k}{\frac{1}{\hat{r}_0^0(D)} + \frac{1}{\hat{r}_0^1(D)} + \cdots + \frac{1}{\hat{r}_0^{k-1}(D)}} \tag{12}$$

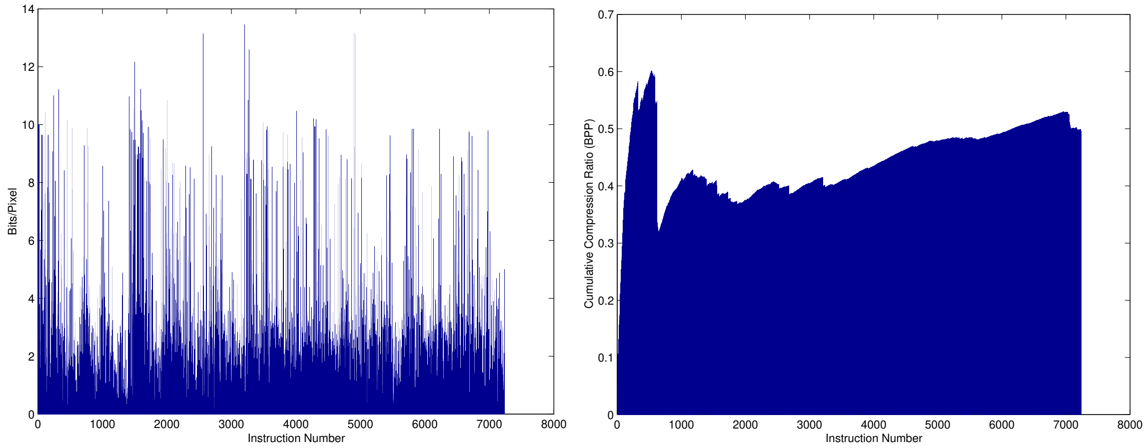


Image Basselope at 0.5 BPP

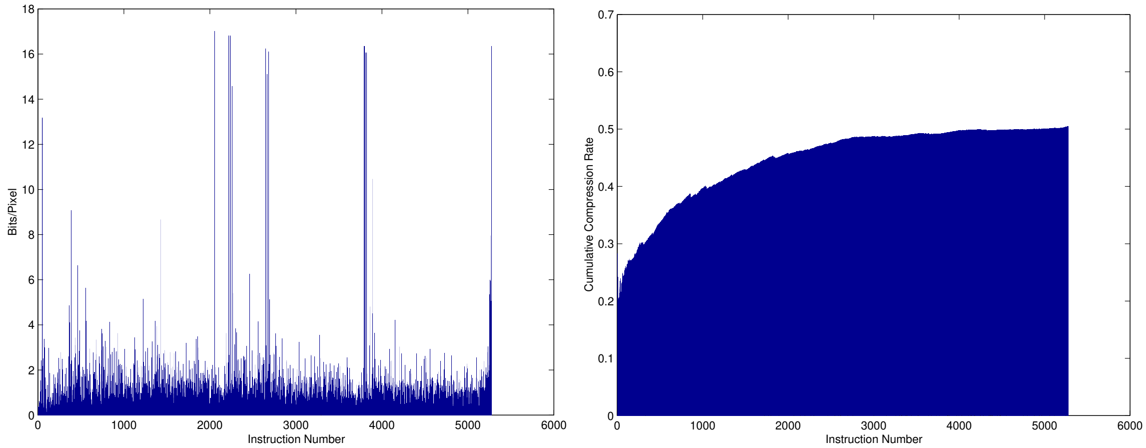


Image Lena at 0.5 BPP

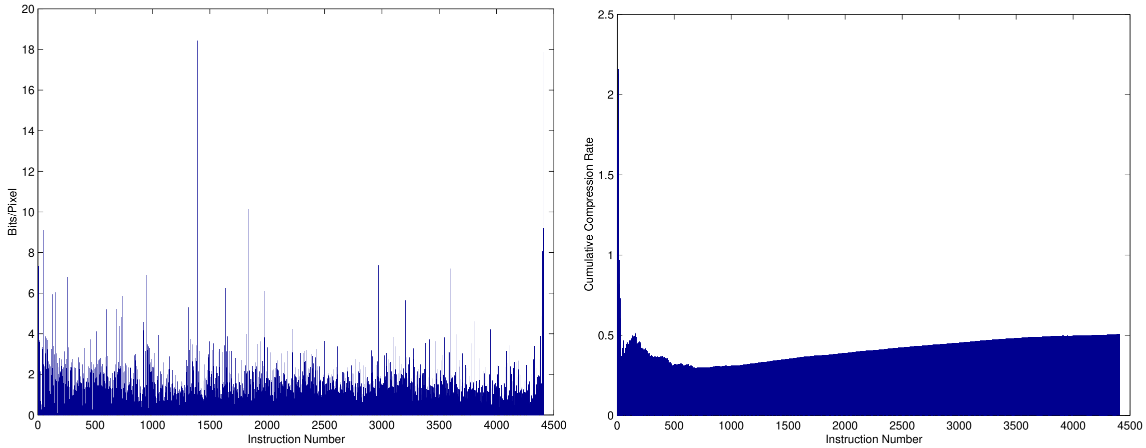


Image San Francisco at 0.5 BPP

Figure 1: Variation of compression rate as the pattern matching process proceeds (Left Column), and Cumulative BPP of the compression process (Right Column).

provided $k = o(\log m / \log\log m)$ and $m \to \infty$.

The above approximate analysis relies crucially on the hypothesis [H]. Its validity depends on how non-stationary the distorted database is. Our preliminary theoretical investigations indicate that one may construct a non-stationary database such that the asymptotic of (11) does not hold (cf. Figure 1), and one must work with $\limsup$ and $\liminf$. But, even if [H] holds, the compression ratio (12) may still not converge as $k \to \infty$. Figure 1 shows the variation of the bit rate during 2D-PMIC (growing database model). The right-hand side plot in Figure 1 indicates that the average bit rate may not converge to a limit.

3 Algorithms, Data Structures and Implementation

In this section, we describe algorithmic and implementation issues related to the compression scheme. The scheme relies on a range of techniques centered around 2-D pattern matching used in conjunction with variable adaptive distortion and run-length encoding. The key computational kernel is a k-d tree based matching heuristic along with a region growing scheme that identifies maximal matches in an image. The corresponding decompression scheme expands matched patterns and run-length encoded segments. Due to its simplicity, the decompressor runs extremely quickly and requires minimal computation. This is crucial for many applications where one aims to display images 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 coding techniques based on 2-D matching, run-length encoding, and lossless encoding, distortion metrics, and algorithmic complexities along with implementation details for image and video streams.

3.1 Encoding techniques

The compression scheme proposed here uses a combination of three techniques to code data: 2-D pattern matching, enhanced run-length encoding (ERLE), and lossless coding. These techniques are applied to progressively code the image. Assuming that a part of an image has been previously encoded, the key task is to encode the pixels located to the right of and below a point we call the *anchor point* (cf. Figure 2(c)). The selection of the next anchor point is based on a *growing heuristic* (cf. [4]). The growing heuristic we adopt is the "wave–front" scheme. This scheme sweeps the image from top to bottom of the image. The anchor point at an intermediate stage in compression of image San Francisco is illustrated in Figure 2(c). Other heuristics grow regions in a circular manner from a center of the image or expand from the main diagonal. These heuristics have been observed to yield similar results [4]. Once the anchor point has been identified, the partial image is coded using either 2-D pattern matching, ERLE, or lossless coding. We observe that for all our growing heuristics phrases do not overlap more than twice.

3.1.1 Two-Dimensional Pattern Matching

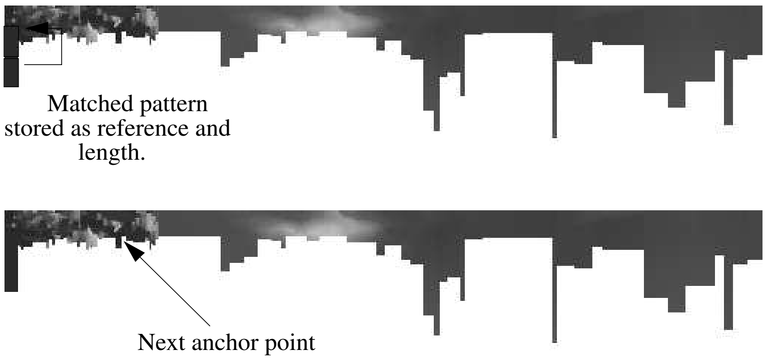
Two dimensional pattern matching is the most efficient way of compressing images among the methods we use in 2D-PMC. The basic idea is to find a two-dimensional region (rectangle) in the uncompressed part of the image that occurs approximately in the compressed part (i.e., database), and to store a pointer to it along with the width and the length of the repeated rectangle. This is illustrated in Figure 2(c). The notion of approximate occurrence is discussed in greater detail in Section 3.2 along with distortion measures. Observe that the objective is to search for the *largest* such area. Consequently, 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 $O(n^2m^2)$ (cf. [3]). This is too expensive since we have to apply it on average $O(N^2/\log N)$ times on an $N \times N$ image.

The problem of identifying maximal matches in a database at an anchor point can be accomplished in two stages: (i) identifying candidate seed points in the database at which potential matches can be found; and (ii) growing regions around seed points and selecting the one that yields best compression ratio with respect to selected distortion metric. Note that the selection of a matching region is a locally greedy process and does not guarantee a globally optimal compression ratio.

To identify candidate seed points, we consider a template of k pixels around the anchor point. For example, assume that this template is a 3×2 region with the anchor point as the top-left pixel in the template. The resulting template corresponds to a 6-dimensional vector. Figure 3 illustrates the vectors for two templates of dimension 2×3 and 3×2. The problem of finding candidate seed points then becomes one of finding all vectors in a 6-D space that match these vectors at the anchor point in an approximate sense. In the example in Figure 3, this corresponds to finding all approximate instances of vectors $[100, 63, 65, 63, 105, 65]$ and $[100, 63, 63, 105, 65, 18]$ and flagging them as candidate seed points for pattern matches. In general, searching for a single template does not yield the best compression. Our pattern matching scheme allows users to define multiple templates and generate candidate seed

(a) Original Image: San Francisco.

(b) Sample run-length encoded segments.



(c) Sample pattern matched segment.

Figure 2: Pattern matching and enhanced run-length encoding techniques in PMIC demonstrated for sample image San Francisco.

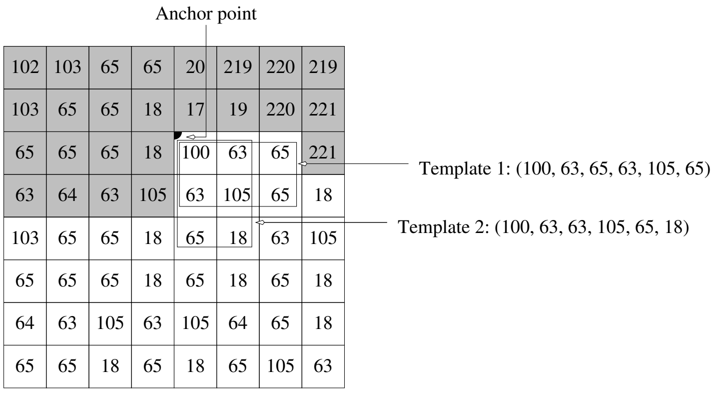


Figure 3: Two templates defined around an anchor point and corresponding six-dimensional vectors that must be searched in the k-d tree formed by the database.

points for all of these templates. Typical template sizes used for our compression results are: 2×3, 3×2, 1×5, and 5×1.

Search for template matches is performed using k-d trees (cf. [8, 14]). A 1-D tree is an ordinary binary search tree; a 2-D tree is similar to a 1-D tree, except, the nodes at even levels compare x-coordinates and the nodes on odd-levels compare y-coordinates when branching. In general, a k-d tree has nodes with k coordinates, and branching at each level is based on only one of the coordinates; for example, we might branch on coordinate number $(l \bmod k) + 1$ on level l. Our implementation of k-d tree search resembles the idea proposed in [4]. To search for a k-dimensional vector using k-d trees, we trace the path from the root down to one of the terminal nodes. The decision whether to go left or right at a node is performed on the $(l \bmod k)$-th coordinate, where l is the depth of the node. This is illustrated in Figure 4 in the context of a 6-D tree for searching a vector $[a1, a2, a3, a4, a5, a6]$. Since our 2-D pattern matching scheme allows the use of multiple templates, one such tree is constructed for each template and various templates at each anchor point are searched through their respective k-d trees.

In our implementation we build a k-d tree of bounded height *MaxDepth* in which the the mean value of the range is used for branching (discriminating) at each node. For example, if the initial range is $[a, b]$, the first discriminating value is $\frac{a+b}{2}$, the second one (for this coordinate) is $\frac{a+b}{4}$ (for the left subtree) and $3\frac{a+b}{4}$ (for the right one), and so on (cf. Figure 4 where branching decisions at level seven on $a1$ are 64 and 192 respectively). The terminal nodes are called *buckets* since they contain one or more leaves. Leaf nodes store data corresponding to the vector and 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

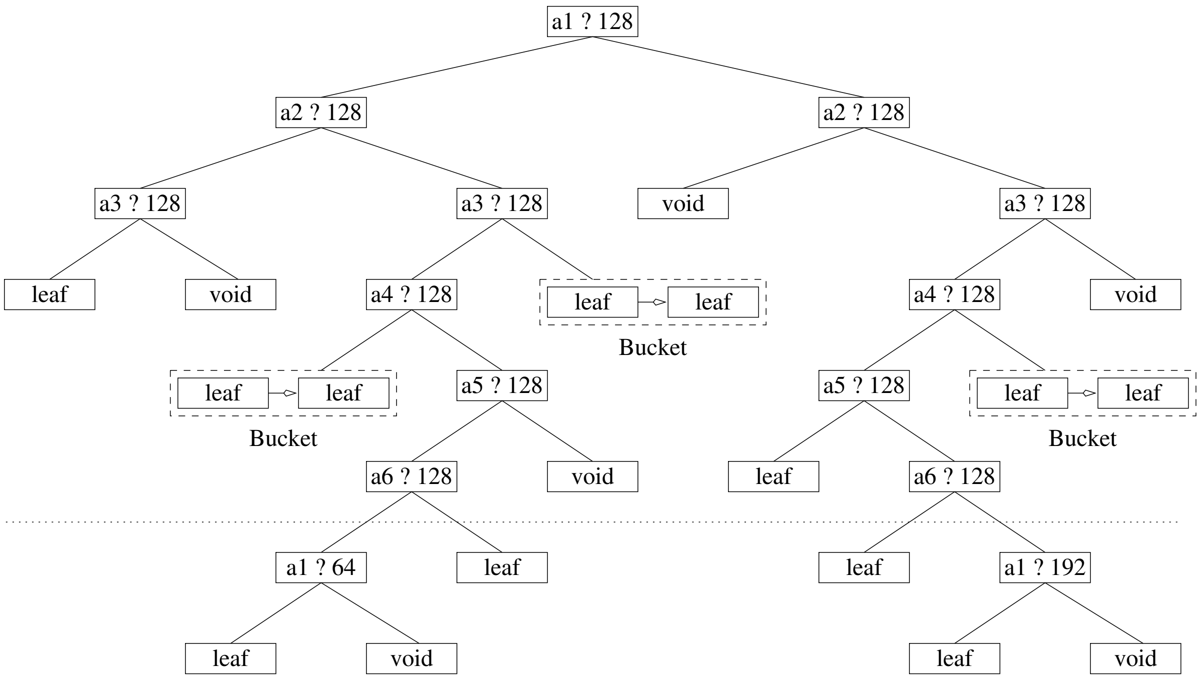


Figure 4: A 6-D-tree (range $0 - 255$) with all left edges signifying a 'less than' decision and right edges signifying a 'greater than or equal to' decision.

possibilities:

- Bucket B is empty; if so, we insert L into it.
- The depth of this branch is already equal to the maximum depth *MaxDepth*. In this case, we simply add L to the bucket which is maintained as a chain of leaves.
- The depth of the node is less than the maximum depth, but the associated bucket B is full. In this case, we expand the node to its children and select the branching decision as the mean of the node range. All of the leaves in bucket B are inserted into the buckets at corresponding children and the original leaf L is re-inserted.

Since we search for an approximate occurrence of a template, we do a *range search* on a k-d tree with two arguments: a *minimum* vector $(m_1, m_2, \cdots, m_k)$ and a *maximum* vector $(M_1, M_2, \cdots, M_k)$. These vectors are determined by the distortion measures discussed in Section 3.2. We search for all k–dimensional vectors $(v_1, v_2, \cdots, v_k)$ such that $m_1 \leq v_1 \leq M_1, \ldots, m_k \leq v_k \leq M_k$. This is done recursively by keeping every subtree that may contain a solution and discarding the others. For example, if the first discriminating value is 128, then for $m_1 = 120$ and $M_1 = 130$ we explore both left and right subtrees of the root, while for $m = 80$ and $M = 90$ we just explore the left subtree.

To get an idea of the complexity of this search we first review some known results. Flajolet and Puech [7] present a detailed analysis of random k-d trees. Among other things, they prove that the average search time for a key in a k-d tree built from N randomly inserted nodes is

$$\frac{2}{k} \ln N + O(1).$$

However, the situation is more intricate when some (say t) coordinates are not specified. Then the search time increases to $\Theta(N^{1-t/k+\theta})$ where $\theta < 0.06$. If the k-d tree is balanced with height $\approx \log_2 N$, then Lee and Wong [14] prove that a range search of rectangle areas is bounded by $O(tN^{1-1/k} + q)$, where t of the coordinates are restricted to subranges and there are q such records. Finally, Friedman et al. [8] showed that if the search area is small and near cubical, then the search time is $O(\log N + q)$.

The k-d tree search used here falls under the model of Friedman et al. [8]. To obtain a more precise estimate of the search time, let us bound the number of paths explored per coordinate assuming the maximum depth is $d = O(\log_2 N)$, and the search range is $[\alpha, \beta]$ for every coordinate. We assume that the initial range is $[a, b]$ (e.g., $[0, 255]$). The maximum number of subranges that overlap with the search range is

$$s = \lceil \frac{(\beta - \alpha)2^d}{b - a} \rceil + 1. \tag{13}$$

Thus, for a problem of dimension k, we must search at most s^k buckets. A bound for the maximum number of comparisons is $q = O(ks^k d)$. For example, for $b - a = 255$, $\beta - \alpha = 8$, $d = 5$, and $k = 6$ we have $s = 2$ and $q = 1953$. In view of this, the total search time is $O(ks^k \log N)$. Actual search performance of a k-d tree is illustrated by the following example:

- dimension: $k = 6$
- database size: $262144 = 512 \times 512$ random vectors (8 bits per coordinate)
- maximum tree depth: 30 (5 levels per coordinate)
- solutions found: 72
- search time: 2.7ms (8.11s for 3000 searches) on a PII 300MHz.

However, as mentioned before in order to find all potential matches, we actually use several k-d trees, each one associated with a different template. Furthermore, to reduce the number of potentially non-interesting solutions, pixels and their associated vectors are grouped by locations (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. Our implementation of the k-d tree follows the idea of [4]. The data structure used to represent a k-d tree is a set of two arrays: A_n the array of nodes, and A_l the array of leaves. A cell, A_n, is composed of two pointers P_l and P_r that can be used differently:

- If P_l is equal to a special value *IsLeaf*, then the node is actually a leaf, and the contents of the leaf can be found in the cell pointed to by P_r in A_l;
- Else P_l and P_r are pointers to the left and right children. A NULL pointer represents an empty bucket.

The structure of a cell of A_n is as follows:

- The vector used to index the leaf;
- A reference (i.e., the coordinates of the point associated with the color vector);
- A pointer to the next leaf. This is used to chain leaves if the bucket contains more than one leaf.

Finally, the procedure to determine the largest two-dimensional match works as follows: for each template, we extract the pixel vector associated with the anchor point and search for approximate occurrences in the k-d trees built over the pixel vectors of the neighboring locations. For each solution, we check if it has already been found before when searching for other templates. If not, we apply the same algorithm as for the enhanced run-length encoding (discussed in Section 3.1.2) to find the best matching shape. If the number of solutions exceeds a certain limit, we abort the search.

3.1.2 Enhanced Run-Length Encoding

Run-length encoding (RLE) of images identifies regions of the image with constant pixel values. We enhance RLE by giving it the capability of coding regions in which pixel values can be (approximately) modeled by a planar function. We call this technique *enhanced run-length encoding* (ERLE). ERLE approximates in the least-squared error sense a given grid $m \times n$ of pixels by a planar surface. The coefficients of the planar surface are computed by solving a system of normal equations associated with the least-squared error procedure. Once the planar surface is determined, a sub-segment of the $m \times n$ grid that is within the distortion distance is identified and coded using ERLE. We observe that this is particularly useful for synthetic images typically found on the Web.

To identify the largest subsegment of a given $m \times n$ grid that can be coded using ERLE, we use the following procedure:

1. Initialize x to an arbitrary constant, say $x = 2$.

2. Test if the rectangle anchored at the *anchor point* of size $x \times 1$ can be encoded using ERLE within given distortion distance.

3. If step 2 gives a positive answer, double x.

4. Let X be the largest x found through the above procedure.

5. Apply the same strategy to find the largest width Y.

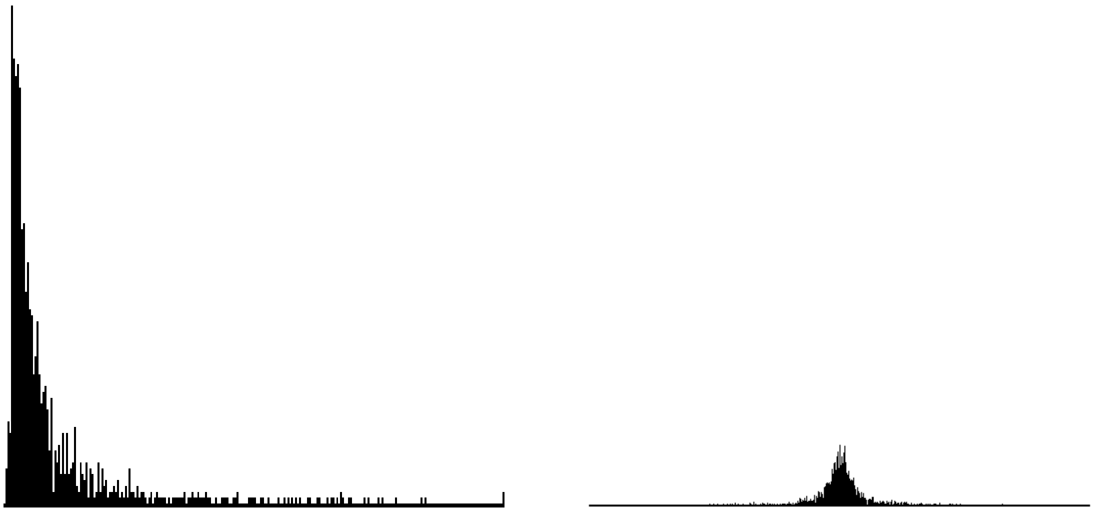
6. If $X \geq Y$, then for $y = 2$ to $Y - 1$ find by dichotomy the greatest x so that the rectangle $x \times y$ can be ERL encoded.

7. If $X < Y$, swap the role of the x-coordinate and the y-coordinate in step 6 above.

It is possible that a single anchor point may have a matching pattern as well as an ERLE representation. In such cases, we compare the cost (in terms of the number of bits used to store the ERLE parameters versus the pattern matching parameters per pixel of compressed sub-image) and select the better one. Note that the number of bits used is calculated according to the distribution model used by the lossless compressor discussed in Section 3.1.3. This important fact allows us to save some more memory during the lossless encoding phase. Another important observation relates to the fact that we might have overlapping coded segments (i.e. pixels with multiple representations). This is not a problem since either representation of the pixel is within specified distortion bound. Furthermore, the pixel value is set to the value determined by the last coded instruction at the decompressor side. This prevents segmentation of the image and artifacts such as specking.

3.1.3 Lossless Encoding

Once the lossy image compression is completed, the image description is a script in the following form: `instruction1, parameters1, instruction2, parameters2`, and so on where `instructionN` $\in$ { `Pixel_Color, Enhanced_RLE, Match_Pointer`} and where the parameter set is different for each instruction. Here, `Pixel_Color` indicates that the next pixels are sent without compression, `Enhanced_RLE` indicates that we use the ERLE, and



(a) (b)

Figure 5: Typical histograms of: (a) the height of matching areas; (b) of the coefficient c_0 in the ERLE coding that fits a plane $c_0 + c_1x + c_2y$ on a sub-image.

`Match_Pointer` refers to the 2D pattern matched encoding. The stream sent to the arithmetic encoder appears as follows:

$$\underbrace{10}_{\text{instruction}}\ \underbrace{10101001}_{\text{parameter}_{10}1}\ \underbrace{001110100100}_{\text{parameter}_{10}2}\cdots$$
$$\underbrace{01}_{\text{instruction}}\ \underbrace{0100}_{\text{parameter}_{01}1}\ \underbrace{001001}_{\text{parameter}_{01}2}\ \ldots$$

with different ranges and distributions for each instruction/parameter.

Since the distribution of the parameter values is not uniform (cf. Figure 5), we can further improve compression by losslessly encoding this data. To make this compression efficient, a distribution model is built for each parameter based on the previously observed values and the stream is encoded using arithmetic coding. Since we want this compression method to be suitable for web-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. We implement our own arithmetic encoder based on the ideas discussed in [12], hence we refrain from describing it in detail.

3.2 Distortion Measures

The process of lossy compression can be viewed as adding "noise" to the image. For this reason, we also refer to the distortion level D as `RandomNoise`. The original pattern matching scheme PMIC [3] computed the distortion–per–symbol for a block of data (averaged over the block). This distortion measure was called *uniform distortion* in [3]. We observed experimentally, however, that uniform distortion leads to significant perceptual errors. To avoid this, in 2D-PMC we adopt an alternate approach in which we assign to each *individual* pixel a *range* (i.e., lower and upper bounds on the allowable distortion value per pixel) that depends on the values of surrounding pixels. The basic idea is to be less tolerant (i.e., have a small range) in slowly varying regions and more tolerant in quickly varying regions. We have experimentally evaluated several approaches and describe some of these.

3.2.1 Local Distortion Range

In order to reproduce more accurately slowly varying regions, we introduce differences between an individual pixel value and its surrounding neighbors. The range assigned to the pixel located at (x, y) is given by:

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

where D = `RandomNoise` is the maximum distortion assigned in the uniform distortion method, `LocalNoise` is additional distortion, and `LocalVariation` is the maximum of the absolute value of the differences between the pixel values at (x, y) and its 8 neighbors. The drawback of this method is that it uses only the 8 closest pixels and consequently is very sensitive to noise.

3.2.2 Global Distortion Range

To counter ill effects of the locality of the previous method, we introduce a function that assigns a large range only when there are few pixels of similar value around it.

We now show how to compute the range R for pixel P_0 located at (x, y). Define the weight W_{ij} as

$$W_{ij} = \max(0, 1 - \frac{\sqrt{i^2 + j^2}}{r}).$$

where $r = pr\_strech$ (cf. Figure 6) is the radius of a circle surrounding pixel P_0. Observe that $0 \leq W_{ij} \leq 1$, and its value decreases with the distance from P_0. Denoting the pixel value at location (i, j) by V_{ij}, we define δ_{ij} as follows

$$\delta_{ij} = |V_{i+x,j+y} - V_{x,y}|,$$

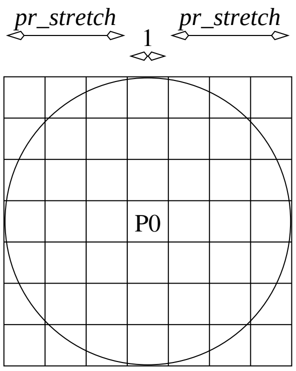


Figure 6: Illustration of the Global Distortion Range ($pr\_strech = r$)

and the total weight W as

$$W = \sum_{\substack{0<|i|\leq r \\ 0<|j|\leq r}} W_{ij}.$$

We compute the sum S as

$$S = \frac{pr\_min}{W} \sum_{\substack{0<|i|\leq r \\ 0<|j|\leq r}} \frac{W_{ij}}{\min(pr\_max, \max(pr\_min, \delta_{i,j}))}$$

where pr_min and pr_max are constants. The selection of pr_min and pr_max is such that we assure that $\delta_{ij} \in [pr\_min, pr\_max]$ for all $0 < i, j \leq r$. The parameter S should be close to 1 for slowly varying area (e.g., if $\delta_{ij} = 0$ for all i, j, then $S = 1$), and S should be smaller than 1 for quickly changing area. Finally, the range R for pixel $P_0 = (x, y)$ is computed as

$$R = \texttt{RandomNoise} + \texttt{Amplitude} \times (\frac{1}{S} - pr\_flat)$$

where pr_flat is chosen close to 1, and `Amplitude` is a constant. Thus, the pixel value $V_{x,y}$ of P_0 can take values in the interval $[-R + V_{xy}, R + V_{xy}]$.

We observe that the variation in R, and hence the distribution of errors in this method is less specky than the one computed with the local distortion. This method gives better visual results than the previous one, because it hides the compression noise more efficiently. The difficulty in applying this method lies in tuning the parameters pr_min, pr_max and `Amplitude` in order to obtain the best visual quality of an image.

3.3 Overall Space and Time Complexity

We argue that the overall time complexity of the compression algorithm is $\mathcal{O}(N^2)$ on average and $O(N^2 \log N)$ in the worst case; the corresponding decompression time is $O(N^2)$ (but on

Image	BPP	Compr. Time	Decompr. Time
Lena	0.25	108	0.075
	0.5	129	0.153
Basselope	0.27	122	0.106
	0.5	120	0.198
San Francisco	0.25	99	0.061
	0.5	115	0.125
Banner	0.25	8	0.007
	0.5	7	0.010

Table 1: Compression and decompression times (in seconds) for four images on a 400 MHz Pentium II.

average only $\mathcal{O}(N^2/\log N)$ pointer reads are required), and the space complexity is $O(N^2)$. (We will henceforth write $O(\cdot)$ for the worst case complexity, and $\mathcal{O}(\cdot)$ for the average case complexity.)

Let us first consider the compression algorithm. The most time consuming step is the two-dimensional pattern matching that is applied with high probability $\mathcal{O}(N^2/\log N)$ times since we proved in Theorem 1 that a typical repeated rectangle has $\mathcal{O}(\log N)$ pixels. In the worst case we need to apply the 2D pattern matching search $O(N^2)$ times. The search time in a k-d tree is $O(\log N)$, hence the total time average-case complexity is $\mathcal{O}(N^2)$ and the worst-case complexity is $O(N^2 \log N)$. Unfortunately, the constant in front of N^2 is large due to many potential searches allowed in the k-d tree, that is, it can be as large as ks^k where s is defined in (13) (cf. Section 3.1.1). This constant can be reduced by limiting the search window for potential seed points for pattern matching. Typical compression times for four images are shown in Table 1.

As mentioned before, the decompression is extremely simple and very fast. It requires on average only $\mathcal{O}(N^2/\log N)$ read operations, and it does not require any additional computation, if the arithmetic coding is not used. This is clearly evidenced by the decompression times in Table 1. The simple and inexpensive decompression scheme allows one to send the decompression program along with the data when downloading images from the web, obviating the need for decoding software on the decompression side. Such a JAVA decompressor is being implemented and preliminary results will be reported in forthcoming paper.

The space complexity is clearly $O(N^2)$. To get an idea of actual memory requirements, we note that k-d trees for "Lena", "Basselope", and "Banner" require 31.6 MB, 21MB, and

2MB respectively. While most current PCs easily support such memory requirements, these can be further improved. Our implementation maintains k-d trees corresponding to entire images. However, using the assumption that likely matches are found in the vicinity of the anchor point, we can reduce the space requirement considerably.

3.4 Video Implementation

Pattern matching video compression uses a decompressed representation of the previous frame that has been compressed in our framework as the database to compress current frame. We refer to this decompressed representation as a lossy image. The reason for using the lossy image as a database is that we want the decompressed image at the client side to be within a distortion bound from the original image. Since the decompressor only knows the compressed frame (in its compressed and uncompressed form), this must be used as the database for pattern matching compression. In contrast, if the original previous frame was used as the database, this database is not available at the decompressor. Consequently, errors propagate quickly through subsequent frames during decompression. Since the previous compressed frame (in its uncompressed form) forms a static database, the compression process is modeled well by our analytical framework of Section 2.1.

Applying this framework directly to video compression, it was 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 *each* 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. Experimental results indicate that for some samples of video for which the compression did not work well (e.g., compression ratio (CR) $\simeq 4$) this method could improve the performance enormously (e.g., $CR \simeq 50$ video) without noticeable loss in quality. Table 2 compares the compression with and without the low-pass filter.

To implement this idea, we keep two copies of the previous frame. One as it has been encoded, and another that is a filtered version. We also keep at hand a plain and a filtered version of the image being encoded. We populate the k-d trees with templates from the filtered version of the previous image. When trying to find a match, we search k-d trees for a template that is at the location pointed to by the anchor point in the filtered image. On finding a match, we store a pointer to it. Notice, however, that during the decoding phase,

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 2: Video "Train" with and without model pre-filtering.

since the decoder knows nothing about the filtering, it will make a copy of the non-filtered pixels. Thus, the image reproduced is still sharp, even if we use a smoothed version to determine "motion vectors".

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

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

That is, the new pixel values $\widetilde{V}_{ij}$ of the smoothed image become

$$\widetilde{V}_{i,j} = \sum_{-1 \le k,l \le 1} V_{i+k,j+l} c_{k,l}$$

where V_{ij} are pixel values of the original image.

The filters we have tried are:

- A low-pass filter with the frequency response $h(f_x, f_y) = \mid \frac{(1+\cos(2\pi f_x))*(1+\cos(2\pi f_y))}{4} \mid$, $0 \le f_x \le \frac{1}{2}, 0 \le f_y \le \frac{1}{2}$. The convolution matrix for this filter 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 $h(f_x, f_y) = \mid \frac{(1+2\cos(2\pi f_x))(1+2\cos(2\pi f_y))}{9} \mid$, $0 \le f_x \le \frac{1}{2}, 0 \le f_y \le \frac{1}{2}$. 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 yields the best performance.

4 Experimental Results

2D-PMIC				JPEG			
BPP	CR	RMSE	PSNR	BPP	CR	RMSE	PSNR
Image: Banner							
0.29	28.00	9.5	28.6	0.29	28.00	27.4	19.4
0.50	16.00	1.3	45.8	0.50	16.00	15.3	24.5
0.54	14.89	0.0	Inf	1.01	7.94	15.1	24.6
				2.00	4.00	15.1	24.6
Image: Basselope							
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
Image: Lena							
0.25	32.01	10.8	27.5	0.25	32.30	8.9	29.1
0.49	29.30	8.7	29.3	0.50	16.03	5.8	32.9
1.05	7.61	5.6	33.1	1.00	8.04	4.2	35.7
1.94	4.13	3.6	37.1	2.01	3.81	2.7	39.4
Image: San Francisco							
0.25	32.00	17.0	23.5	0.25	32.00	15.5	24.3
0.50	16.00	13.1	25.8	0.50	16.00	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 3: Comparison of compression results from 2D-PMIC and JPEG for different images.

In this section, we present some experimental results for image and video compression. Pattern matching image compression is compared to JPEG compression in Table 3, Figure 7 and Figure 8. Several observations can be made from this table:

- For synthetic images (Banner and Basselope), 2D-PMIC outperforms JPEG both in terms of compression ratios as well as quality. This is also evident from Figure 7. A comparison between 2D-PMIC and wavelet compression shows similarly the superiority of 2D-PMIC [3].

- For compression of general images, 2D-PMIC is competitive with JPEG and wavelet

Banner 0.29 bpp, 2DPMIC vs. JPEG

Banner 0.50 bpp, 2DPMIC vs. JPEG

Figure 7: Comparison of image quality using 2D-PMIC and JPEG compression.

compression up to 0.25 bpp. But in Figure 8 2D-PMC seems to be (perceptually) superior to JPEG for San Francisco for 0.5 bpp even though PSNR is better for JPEG image (24.3 dB versus 23.5 dB for 2D-PMC).

- At very low bit rates, frequency domain methods are still better than 2D-PMC for natural images.

These results indicate that for synthetic images and higher bit rates, 2D-PMC is the method of choice, particularly because of the simplicity of the decompression procedure.

In Table 4 we show some results of video compression for two video clips, namely "Claire" and "Ping_Pong". We compress them using 2D-PMC with the previous uncompressed frame as the database. A demo is set up at `http://www.cs.purdue.edu/homes/spa/Compression/2D-PMC.html` for the reader's evaluation. Table 4 presents results for "low-quality" (lq) and "high-quality" (hq) compression. It is clear from the table that 2D-PMC yields excellent compression ratios and quality while supporting very high decompression rates (speed of decompression). Indeed, by augmenting this technique with inter-frame prediction (in a manner similar to MPEG), the performance can be further enhanced. The objective here however is to demonstrate the power of the pattern matching framework upon which the rest of the conventional compression machinery can build. We are currently implementing a JAVA decompressor that can be transmitted with the compressed video obviating the need for installation of decompression clients. Preliminary results can be viewed on `http://pier.ecn.purdue.edu/ dxm/Work/Research/Project1/` (image decompression) and `http://pier.ecn.purdue.edu/ dxm/Work/Research/Project2/` (video decompression).

(JPEG)

(2D-PMC)

(JPEG)

(2D-PMC)

Figure 8: San_Francisco and Lena at 0.5 bpp: Comparison of image quality using JPEG and 2D-PMC compression.

Video	Mbps	CR	PSNR	Image Size (8 bit pixels)	Number of Frames
Claire_hq	0.14	169	33.7	360×288	168
Claire_lq	0.07	347	31.0	360×288	168
Ping-Pong_hq	0.40	58	24.0	352×288	107
Ping-Pong_lq	0.26	89	22.6	352×288	107

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

5 Concluding Remarks

In this paper, we have presented a 2D pattern matching framework for compression of image and video data. The following conclusions can be made:

- 2D-PMIC yields excellent compression ratios and is competitive with JPEG and wavelet based compression up to 0.25 bpp. 2D-PMIC works better on synthetic images while transform-based methods work better on natural images. In general, 2D-PMIC works better when high frequency components dominate the image;
- 2D-PMC is slower in terms of compression time but extremely fast and simple in terms of decompression. Decompression can be done on-fly with minimal software support at the client.
- 2D-PMC naturally adapts to variable resource availability by changing the distortion bound. This results in higher compression rates as well as faster compression times.
- The pattern matching framework of 2D-PMC can be used in conjunction with many of the current techniques to leverage the benefits of these schemes.

We are currently working on improving the pattern matching framework and combining it with other conventional techniques based on frequency domain methods. More work is needed on automatic selection of parameters for the pattern matching scheme. 2D-PMC for video is particularly promising, and we are exploring it actively as a part of our ongoing research. When applied in conjunction with prediction and frequency transforms, our framework has the potential to yield significantly improved performance. Alternately, the pattern matching framework can itself be used as a homogeneous compression technique for textual, audio, image, and video data leading to a uniform multimedia compression standard.

ACKNOWLEDGMENT

It is our privilege to acknowledge valuable discussions with I. Kontoyiannis (Purdue University) and Y. Reznik (RealNetwork Inc.). We also thank two referees whose comments were valuable in improving the presentation.

References

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

[2] 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.

[3] M. Atallah, Y. Génin, and W. Szpankowski, A Pattern Matching Image Compression: Algorithmic and Empirical Results, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21, 618-627, 1999.

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

[5] A. Dembo and I. Kontoyiannis, The Asymptotics of Waiting Times Between Stationary Processes, Allowing Distortion, *Annals of Applied Probability*, 9, 413-429, 1999.

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

[7] P. Flajolet and P. Puech, Partial Match Retrieval of Multidimensional Data, *J. of the ACM*, 33, 371–407, 1986.

[8] 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.

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

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

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

[12] 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.

[13] I. Kontoyiannis, An Implementable Lossy Version of the Lempel-Ziv Algorithm– Part I: Optimality for Memoryless Sources, *IEEE Trans. Information Theory*, 45, 2285-2292, 1999.

[14] D.T. Lee and C.K. Wong, Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees, *Acta Informatica*, 9, 23–29, 1977.

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

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

[17] 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.

[18] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York, 2000.

[19] A.J. Wyner, The Redundancy and Distribution of the Phrase Lengths of the Fixed-Database Lempel-Ziv Algorithm, *IEEE Trans. Information Theory*, 43, 1439–1465, 1997.

[20] A. Wyner and J. Ziv, Fixed Data Base Version of the Lempel-Ziv Data Compression Algorithm, *IEEE Trans. Information Theory*, 37, 878-880, 1991.

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

[22] E.H. Yang, and J. Kieffer, On the Redundancy of the Fixed–Database Lempel-Ziv Algorithm for Φ-Mixing Sources, *IEEE Trans. Information Theory*, 43, 1101–1111, 1997.

[23] 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.

[24] R. Zamir and K. Rose, Natural Type Selection in Approximate String Matching, preprint 1999.

[25] 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.

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

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