

# Compressed Pattern Matching for Predictive Lossless Image Encoding

Tao Tao, Amar Mukherjee

School of Electrical Engineering and Computer Science  
University of Central Florida  
Orlando, FL 32816 USA  
Email: (ttao+amar)@cs.ucf.edu

## ABSTRACT

Pattern matching in compressed image domain is a new topic in computer science. Many works have been reported for pattern matching for compressed text and for lossy compressed image. However, searching of images in lossless compressed domain is almost a blank area and needs to be explored. Lossless image compression is widely used in areas such as medical images, satellite images, geometric images and many other areas that need to losslessly maintain the data of the images. Being able to searching in the compressed domain will save disk space and searching time and bring up considerable economic savings in these areas. In our work, we have studied the possibility of compressed pattern matching for the most three popular lossless image compression schemes: lossless JPEG, CALIC and JPEG-LS. Our study indicates that these algorithms can be search-aware by minor modification. We also present a modified JPEG-LS algorithm and the corresponding searching algorithm. Experimental results show that our method, comparing with the “decompress-then-searching” method, has nearly 30% improvement in searching time for most natural images. The modified JPEG-LS algorithm also has shorter encoding and decoding time, with an improvement of about 12-15% and 8-12%, respectively, for most natural images. The tradeoff is the decrease of compression of about 2% -8%. To our best knowledge, this is the first report on JPEG-LS compressed matching algorithm and this is the first “competitive” compressed pattern matching algorithm for lossless image compression.

**Keywords:** Pattern Matching, Lossless Image Compression, Predictive Encoding, JPEG-LS, Competitive

## 1 INTRODUCTION

Data compression is a very important area in computer science and engineering [Witten99]. The compression algorithm can be categorized as either text compression or image compression, or it can be categorized as either lossy compression or lossless compression. The large amount of data need to be stored is compressed using proper compression algorithm, depends on the features of the data and the applications. Today, the size of the data needs to be processed is getting larger and larger, thus the importance of data compression is increasing.

Pattern matching is another important area [BaezaYates88] [Knuth77]. It has also been well developed and many pattern matching algorithms have been proposed. The most popular text-matching algorithm has been implemented into almost every commercial product of text editor. Fast searching helps people to quickly locate the information they need. These two areas are recently combined and a new area in computer science is produced: compressed pattern matching (CPM). The goal of this new research topic is: *given the compressed format  $I_c$  of an image or a text  $I_u$  and a pattern image or text  $P$ , report all occurrences of  $P$  in  $I_u$  without decompressing  $I_c$  or with partially decompression.*

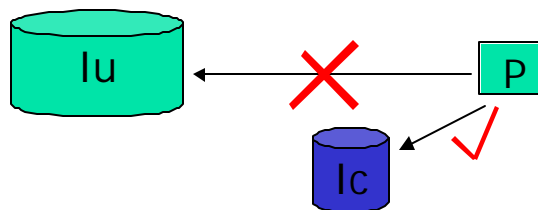


Figure 1 Compressed Domain Pattern Matching Concept

Intuitively, pattern matching in the compressed domain has two main advantages:

1. Since we do not need to decompress the image or text being searched into, the disk and space and memory is saved.
2. Since the compressed data is smaller than its original format, pattern searching in this smaller size of data will result in a shorter searching time. Ideally, if the compressed data is  $m$  times smaller than the original data, we expect the time of searching in the compressed domain to be  $m$  times faster than that in the uncompressed domain.

However, compressed pattern matching is not just a simple combination of works from these two areas. Since the pattern and the compressed text/image are in different domain – original domain and compressed domain, comparisons between these two need extra to make it possible. We may immediately have the following idea:

convert both of them to the same domain. Thus, the choices are:

1. Convert both to uncompressed domain.
2. Convert both to compressed domain.
3. Convert both to a third domain.

The first choice is obviously what people currently are doing and is undesirable. The second choice, which is called *fully compressed pattern matching*, is possible and has actually been used in many algorithms. However, we must be aware that, the compressed format of the pattern does not necessarily have the same representation as its match in  $I_c$  (if it has one). The reason is that most data compression algorithms use the “context” of the symbol being encoded for compression. The context of the pattern and the context of its match in the data to be searched into, are very likely to be different. The third choice is also possible and is also widely used by many algorithms. One possible “third domain” is the “intermediate domain” that between the uncompressed domain and the compressed domain, or “partially compressed domain”. Actually, this is what many CPM algorithms use. We need to be aware that the cost of converting the data to that domain should not exceed the cost of fully decompressing the data. Otherwise it is not worthy of doing that.

Most CPM algorithms are for text compression, especially for LZ family compressions. However, works for Huffman encoding, anti-dictionary and other compressions have also been reported. [Cleary84] [Bell2001] [Mandal99] [Cleary97] [Adjeroh2001] [Bell2001-2] [Bunke95].

Among these algorithms:

Manber [Manber97] proposed an algorithm and the basic idea is to replace common bigrams with special symbols. Although the compression ratio is only about 30%, this compression allows the basic pattern matching algorithms to be used on the compressed file. A similar approach was proposed by Shibata [Shibata99], who used a fast Boyer-Moore algorithm for pattern matching.

LZ family compressions such as LZ77, LZ78 and LZW algorithms can be designed to be search-aware. For example, if word-based search is preferred, LZ77 algorithm can be designed to put words or phrases in the dictionary that start or finish in word boundaries.

Amir [Amir96] proposed a method for searching LZW encoded files, which is claimed as “almost optimal”. The algorithm uses a special data structure called a dictionary tree, which implicitly decodes the compressed text without producing the output symbols. Barcassia extended Amir’s work to an LZ compression method that uses the so-called ID heuristic.

Farach and Thorup proposed a randomized algorithm [Farach98] to determine whether a pattern is present or not in LZ77 compressed text in time  $O(m+n*\log^2(u/n))$ . Navarro and Raffinot [Navarro99] proposed a hybrid compression between LZ77 and LZ78 that can be searched in  $O(\min(u, n*\log m)+r)$  average time, where  $r$  is the total number of matches.

Karpinski [Karpinski95] considered fully compressed pattern search – both pattern and text are compressed and no decompression is done.

Shibata [Shibata99-2] used the anti-dictionary idea. He proposed an algorithm that preprocesses the pattern and the anti-dictionary in  $O(m^2+a)$  time, and then determines all occurrence of the pattern by a linear scan of the compressed text of length  $n$ .

Mukherjee and Acharya [Mukherjee94] [Mukherjee95] studied searching on Huffman coded files. It was concluded that a simple search of the Huffman coded files to find a compressed pattern using a fast search algorithm such as KMP will not produce correct result.

Moura proposed a semi-static word-based modeling scheme for Huffman coded compressed text files which can be searched directly at word boundaries using any fast sequential pattern search algorithm.

It is necessary to point out that the only compressed matching algorithm for *lossless image compression* has been reported is for barcode, which is bi-level image instead of continuous-tone image.

In this paper, we present an original research on pattern matching for lossless compressed image. The problem needs to be solved, is to look for all occurrences of the pattern image, in an image that is compressed with a lossless image compression method, without decompressing the image or with partially decompressing it.

Pattern matching in lossless compressed image is a hard and new topic. Most image matching algorithms in the compressed domain that have been reported are for lossy compression and the matching is usually approximate. Lossless image compression is widely used [Howard93] in areas such as medical images, satellite images, geometric images and many other areas that need to losslessly maintain the data of the images. Being able to searching in the compressed domain will save disk space and searching time and bring up considerable economic savings in these areas.

In section 2, we introduce the three popular lossless image compression algorithm: lossless JPEG, CALIC and JPEG-LS. For each compression algorithm, we discuss

the possibility of compressed pattern matching. In section 3, we introduce our modified JPEG-LS encoding algorithm. The corresponding searching algorithm is also introduced in this section. The experimental results are reported in section 4. Conclusion and a discussion on the future works are given in section 5.

## 2 Predictive Lossless Image Compression Algorithms and CPM

### 2.1 Lossless JPEG

Lossless JPEG is the first lossless still image compression international standard. There are 7 prediction models in Lossless JPEG, as shown in Figure 2 and Table 1. Figure 2 shows the context of X. Table 1 lists all 7 prediction models, where  $X^{\wedge}$  is the predictive value of X.

|    |   |
|----|---|
| NW | N |
| W  | X |

Figure 2 Lossless JPEG Prediction Context

|   |                           |
|---|---------------------------|
| 1 | $X^{\wedge} = N$          |
| 2 | $X^{\wedge} = W$          |
| 3 | $X^{\wedge} = NW$         |
| 4 | $X^{\wedge} = N+W-NW$     |
| 5 | $X^{\wedge} = N+(W-NW)/2$ |
| 6 | $X^{\wedge} = W+(N-NW)/2$ |
| 7 | $X^{\wedge} = (N+W)/2$    |

Table 1 Lossless JPEG Prediction Models

Different images have different structures thus they should use the prediction model that is suitable for them. For example, an image that has many vertical lines may consider model 1 while an image that has many horizontal lines may consider model 2. The prediction model can be pre-determined by a pre-processing of the image.

The prediction error,  $d = X - X^{\wedge}$  will be encoded using entropy encoder such as Huffman coding or arithmetic coding.

Lossless JPEG algorithm is a simple scheme and achieves good compression.

There are two major steps in lossless JPEG: prediction and entropy encoding. In this paper, we will only discuss CPM on the outputs produced from prediction. Unless otherwise indicated, in the rest of this paper, whenever we talk about “compressed data”, we only mean the data

produced after all other steps but before the entropy encoding step.

We claim that CPM can be done for lossless JPEG compressed image since the context of each pixel is very small. The distance of the pixels used for prediction and the pixel being encode, is only 1 for all cases. In other words, the compressed data does not depend on its distant neighbors but only the closest neighbors. To search a sub-image in the compressed data, we can simply compress the pattern the same way as the image. The compressed representation of the pattern and its match in the compressed image will match except on the boundaries since the information on the boundary is lost.

Let’s take prediction model 2 as example and assume the height of the pattern image is one-pixel to illustrate how we can match the compressed pattern with its match in the image.

Suppose the pattern to be searched is:  $\{100, 101, 102, 100, 102, 104\}$  and a row of the image, where the pattern occurs, is:  $\{..., 99, 105, 100, 101, 102, 100, 102, 104, 110, 105, ... \}$ .

The prediction errors generated from the pattern is:

$$\{100, 1, 1, -2, 2, 2\}$$

The prediction errors generated from that row of image is:

$$\{..., ?, 6, -5, 1, 1, -2, 2, 2, 6, -5, ... \}$$

It is easy to see that, except the “boundary”, the prediction errors of the pattern and its match in the image are the same. Thus, to search a pattern in the compressed domain, we simply compress it and search in the compressed image, without comparing the pixels on the boundary.

We now formally define **boundary** as:

*A boundary of an image pattern contains the pixels that do not have a complete context to predict its value, for a certain prediction model.*

Because of the boundary, the above matching algorithm may cause mismatch since two different patterns may have the same prediction errors if the boundary is not compared. For example, pattern  $\{200, 201, 202, 200, 202, 204\}$  will have the following prediction errors, using model 2:  $\{200, 1, 1, -2, 2, 2\}$ . Without comparing the boundary, this pattern will have a mismatch in the above image row. We propose to use a verification stage to verify the sub-image searched, the verification is very simple – just comparing any pixel value of the pattern and the searched sub-images. Our experiments showed that the mismatch was very rare.

Actually, this could lead to another interesting matching. We can imagine the mismatched pattern as the “intensified” version of the original pattern, i.e., image values in the original pattern are all shifted by a constant number. In a way, these two patterns represent the same feature. For many applications, such as searching the medical images and satellite images, the “intensified” matches may also need to be reported.

## 2.2 CALIC Algorithm

Comparing with Lossless-JPEG, the Context Adaptive Lossless Image Compression (CALIC) algorithm considers a larger context of the predicting pixel. Figure 3 shows the context needs to be considered for predicting X.

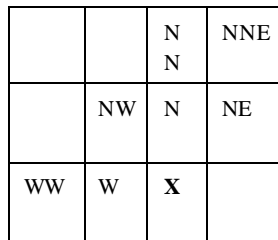


Figure 3 CALIC Prediction Context

CALIC algorithm includes the following steps:

1. Find initial prediction  $X^\wedge$ .
- The initial prediction is calculated in the following way:

```

If (dh - dv > 80)
    X^ = N
Else if (dv - dh > 80)
    X^ = W
Else
{
    X^ = (N+W)/2 + (NE-NW)/4
    If (dh - dv > 32)
        X^ = (X^ + N)/2
    Else if (dv - dh > 32)
        X^ = (X^ + W)/2
    Else if (dh - dv > 8)
        X^ = (3X^ + N)/4
    Else if (dv - dh > 8)
        X^ = (3X^ + W)/4
}

```

where:

$$dh = |W-WW| + |N-NW| + |NE-N|$$

$$dv = |W-NW| + |N-NN| + |NE-NNE|$$

We can tell from above that CALIC is “smarter” than Lossless-JPEG in that it considers the edge directions

(vertical or horizontal) dynamically when predicting the pixel values.

2. Compute the current prediction context.
3. Refine prediction by considering the bias of that prediction context.
4. Compute the prediction error (residual) and update the bias of the prediction context

For each pixel, CALIC first create a vector:

$$[N, W, NW, NE, NN, WW, 2N-NN, 2W-WW]$$

CALIC then compares each component in this vector with initial prediction  $X^\wedge$ . If the component is less than  $X^\wedge$ , we replace it with a 1, otherwise we replace it with 0. Thus a binary vector can be computed and it can be proved that there are 144 possible different vectors for the whole image.

CALIC also computes a quantity d for each pixel:

$$d = dh + dv + 2|N - N^\wedge|$$

These quantities are divided into 4 intervals. Thus, we have  $144 \times 4 = 576$  contexts for the whole image.

CALIC maintains a bias for each of the contexts and it refines the prediction by removing the bias of corresponding context from the initial prediction. The bias of that context is updated then.

5. Re-map the residuals so the remapped residuals lie in range  $[0, M-1]$ , provided that the original pixel values lie in that range.

If the original pixel value lies in between 0 and M-1, the prediction error  $X - X^\wedge$  (if we still use  $X^\wedge$  to represent the re-fined prediction) should be in between  $-(M-1)$  and M-1, which means a larger number of bits are needed to represent these values. However, it’s easy to see that the residual must be in range  $-X^\wedge$  to  $M-1-X^\wedge$ , thus we can easily re-map the residuals into the range  $[0, M-1]$  using the following mapping:

|                |    |                 |
|----------------|----|-----------------|
| 0              | -> | 0               |
| 1              | -> | 1               |
| -1             | -> | 2               |
| 2              | -> |                 |
| ...            |    |                 |
| $-X^\wedge$    | -> | $2X^\wedge$     |
| $X^\wedge + 1$ | -> | $2X^\wedge + 1$ |
| $X^\wedge + 2$ | -> | $2X^\wedge + 2$ |
| ...            |    |                 |
| $M-1-X^\wedge$ | -> | $M-1$           |

6. Find the current residual context.
7. Code the residual using that residual context

CALIC doesn't entropy encode the residuals directly. To achieve better compression, it considers the context – what we call “residual context”, for each residual. The residual context is determined as:

```

0 £d£ q1      P Context 1
q1 £d£ q2      P Context 2
q2 £d£ q3      P Context 3
q3 £d£ q1      P Context 4
...
q7 £d£ q8      P Context 8

```

As the Information Theory suggests, a symbol that has higher probability has low entropy and thus need lower bits to represent. The residuals have higher probabilities in their own context than in the entire image. Thus, by encoding the residuals in their contexts, we achieve better compression.

CALIC algorithm is the best compression algorithm so far in terms of compression ratio. (Bit-parallel using a hardware implementation and achieves better in practice).

From the above discussion, we can tell that the CALIC is also possible for CPM. The prediction error step, although use a larger context, have a maximum context size of only 2. Thus, the boundary width is 2 and we just need to ignore the prediction errors on the boundary. The context computation and the computation of bias of each context, however, needs to consider the whole image scope and will cause problem for the searching in the compressed domain. The same problem exists for JPEG-LS, which is to be introduced in section 2.3. In Section 3, we will discuss how to solve this problem.

### 2.3 JPEG-LS

JPEG-LS [Weinberger2000] is the new lossless still image compression international standard that replaces Lossless-JPEG. The compression method JPEG-LS uses is called LOCO-I (Low Complexity Compression for Image), which was developed based on CALIC and is a much simpler predictive encoding method. The compression of LOCO-I is slightly worse than CALIC. However, LOCO-I has much lower complexity than CALIC and this is why it won over CALIC and became as the standard.

Referring the same table used in CALIC, LOCO-I algorithm includes the following steps:

1. Find the initial prediction  $X^\wedge$

```

IfNW  $\geq \max(W, N)$ 
 $X^\wedge = \min(W, N)$ 
Else
{
    IfNW £  $\min(W, N)$ 

```

$X^\wedge = \max(W, N)$

Else

$X^\wedge = W + N - NW$

}

2. Compute the current prediction context  
The context is represented by a 3-component vector Q (Q1, Q2, Q3) where,

```

Di £ -T3      P Qi = -4
-T3 < Di £ -T2  P Qi = -3
-T2 < Di £ -T1  P Qi = -2
-T1 < Di £ 0     P Qi = -1
Di = 0         P Qi = 0
0 < Di £ T1     P Qi = 1
T1 < Di £ T2     P Qi = 2
T2 < Di £ T3     P Qi = 3
T3 < Di         P Qi = 4

```

Where

$D1 = NE - N$

$D2 = N - NW$

$D3 = NW - W$

T1, T2 and T3 are user predefined positive numbers.

It is easy to see that there are  $9*9*9 = 729$  possible contexts. To reduce the number of the context, JPEG-LS replaces any context vector Q whose first component is negative by  $-Q$ , this results in a total of 365 contexts in total.

3. Refine prediction by considering the bias of that predict context
4. Compute the prediction error (residual) and update the bias of the prediction context
5. Re-map the residuals so the remapped residuals lie in range  $[0, M-1]$ , provided that the original pixel values lie in that range
6. Encode the prediction error using adaptively selected codes based on Golomb codes.

CPM for JPEG-LS, as we stated for CALIC, is possible. We will give a detailed discussion in section 3.

## 3 A Modified JPEG-LS Algorithm and the Searching Algorithm

In this section, after a careful study of the original JPEG-LS algorithm, we point it out that the reason that CPM cannot be performed is that the original JPEG-LS algorithm uses a “global” context. By removing the global context, we propose a modified JPEG-LS algorithm. The new algorithm allows the searching of the pattern to be performed directly in the partially compressed data. The experimental results (section 4) show that the modified algorithm has an improvement of about 12-15%

for encoding and an improvement of about 8-12% for decoding over the original JPEG-LS algorithm. Our results also show that the searching time in the compressed domain is about 30% faster than “decompress-then-searching” method. To our best knowledge, this is the first report on JPEG-LS based compressed pattern matching algorithm and this is the first “competitive” CPM algorithm for lossless image compression. The tradeoff of all these benefits, however, is only 2-8% decrease of the compression.

### 3.1 Problems with CPM in JPEG-LS

We first define some terminologies as below:

*P*: the pixel is being coded.

*Local context*: context that is used to predict the value of *p*.

*Global contexts*: contexts that are used to store the biases and update the predictions.

We then generalize the steps of JPEG-LS from a different point of view.

1. Find the initial prediction  $X^{\wedge}$  from *p*'s local context. The local context involves only the immediate left neighbor, the immediate upper neighbor and the immediate upper-left neighbor of *p*.
2. Compute the current prediction context. This prediction context is determined by *p*'s local context with one more: the immediate upper-right neighbor.
3. Refine prediction by considering the bias of that predict context. The bias currently is affected by all pixels that have been encoded.
4. Compute the prediction error (residual) and update the bias of the prediction context. The bias of one particular context is updated every time the encoder is encoding a pixel that has the same prediction context. Thus, if *p*'s context is *q*, how accurate *p* is predicted will affect the bias of *q* and thus affect the refinement of a pixel that appears anywhere after *p* and has the prediction context *q*. In this sense, the bias is a “global context” concept.
5. Re-map the residuals so the remapped residuals lie in range  $[0, M-1]$ , provided that the original pixel values lie in that range. This is just a constant operation and it does not depend on any context.
6. Encode the prediction residuals using adaptively selected codes based on Golomb codes.

Suppose we need to search a pattern *P* in image *I*, since the “global context” of *P* is very unlikely to be the same as its match in *I*, the biases of each prediction context will be different thus the refined residuals will also be different for the same pixel in *P* and in its match in *I*, although the first prediction, which is only determined by the local context, will be the same. Thus, our first step is to remove this bias refinement and see what happens.

The results showed that removing of the refinement process only have minor impact on the compression ration. The decrease is about 2% for most images we have tested. This encouraged us to continue our modification of JPEG-LS.

As a first step, we decided to convert both the pattern and the compressed image to the intermediate format – the residual format. On the pattern side, this is not a problem; we just need to encode the pattern the same way as discussed above, without refinement and without entropy encoding it. For the image part, a problem exists. Since we do not wish to fully decode the image, the actual values of the previously scanned pixels are unknown to us. When we try to Golomb-Rice decode the data and convert the data to residuals, we need to calculate the coding parameter *k*, which is determined by the prediction context and accumulated residuals. However, the prediction context is computed from the actual values of *p*'s local context. Without the actual values, the Golomb-Rice decoder wouldn't be able to calculate the decoding parameter *k* and thus is not able to decode.

We solve this problem by merging the 365 prediction contexts as one context and apply this context to every pixel. Thus, without the actual value, the Golomb-Rice decoder still can decode the data. However, since *k* is used for encoder also, and *k* is decided in a way to achieve the best compression, if we use one context instead of 365 contexts, the *k* value computed will not be “customized” for *p*'s context and thus it will give a worse compression. Our results confirmed this. Surprisingly, the sacrifice is not as much as we expected, only about 4 to 5%, which is relative small comparing with the benefits we gained from the modification.

### 3.2 Modified algorithm and proposed searching method

We now describe our modified encoding algorithm as below:

1. Initialize the accumulated residual at the very beginning of the encoding.
- For each of the pixel *p*:
2. Find the prediction  $X^{\wedge}$  from *p*'s local context.
  3. Compute the prediction error (residual).
  4. Re-map the residuals so the remapped residuals lie in range  $[0, M-1]$ , provided that the original pixel values lie in that range.
  5. Compute encoding parameter *k* from accumulated residuals and the number of pixels encoded.
  6. Update the accumulated residual.
  7. Encode the remapped residual using Golomb-Rice coder, with encoding parameter *k*.

The decoding algorithm is described as below:

1. Initialize the accumulated residual at the very beginning of the decoding.
- For each of the pixel  $p$ :
2. Find the prediction  $X^{\wedge}$  from  $p$ 's local context.
  3. Compute decoding parameter  $k$  from accumulated residuals and the number of pixels decoded.
  4. Decode the remapped residual using Golomb-Rice decoder, with decoding parameter  $k$ .
  5. Update the accumulated residual.
  6. Inverse map the residual and produce the prediction error.
  7. Compute the original pixel value by adding prediction to the prediction error.

The time complexities of the above algorithms are the same as JPEG-LS but with smaller constant. The memory usage is in the same order too.

Finally, we propose our searching algorithm based on our modified JPEG-LS algorithms. There are three main steps.

Step1: partially encode the pattern:

For each pixel  $p$  in the pattern:

1. Find the prediction  $X^{\wedge}$  from  $p$ 's local context.
2. Compute the prediction error (residual).
3. Re-map the residuals so the remapped residuals lie in range  $[0, M-1]$ .

Step 2: partially decode the image:

1. Initialize the accumulated residual at the very beginning of the decoding.
- For each of the pixel  $p$ :
2. Compute decoding parameter  $k$  from accumulated residuals and the number of pixels decoded.
  3. Decode the remapped residual using Golomb-Rice decoder, with decoding parameter  $k$ .
  4. Update the accumulated residual.

Step 3: Comparing the residuals in the pattern and from the image and report occurrence of the pattern in the image. Any searching algorithm can be used. In our implementation, we use brute-force searching algorithm because of its simplicity.

Thus, the above searching algorithm converts both the pattern and the image to a third domain, which can be called "intermediate domain" or "residual domain". The searching is performed in the intermediate domain.

The following diagram shows the above searching idea:

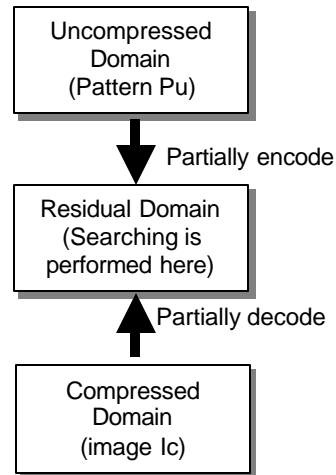


Figure 4 Compressed Pattern Matching for Modified JPEG-LS Encoding

### 3.3 Problems

As we have discussed in section 2, when comparison of the residuals is performed for lossless JPEG, the pixels on the boundary should not be compared. It is also true for JPEG-LS. Thus, in the above searching algorithm, in step 3, we do not compare the residuals on the first row and the first column of the pattern when we perform matching.

As we also have discussed in section 2, the above searching algorithm cannot guarantee that the sub-image found is the exact match of the pattern. It is possible that two different sub-images have the same residual set.

Again, we propose to use a verification stage to verify the sub-image searched – just comparing any pixel value of the pattern and the searched sub-images.

Practically, this is very rare case. In all experiments we have conducted so far, this mismatch has never happened.

## 4 Experimental Results

The benchmarks we used are from standard image library. All benchmarks are 512 by 512 grayscale image. Except one image "fruit", which is a man-made image, all other images are natural image. The pattern we used for search is a 7 by 7 image, the reasons to use a small pattern are: It is easier for debugging and, we can examine the multiple-occurrences of the pattern. The computer used for benchmarking is a laptop with Pentium II processor and 256M RAM.

The following table shows the encoding/decoding time improvement and the compression ratio decrease of our modified JPEG-LS algorithm over the original algorithm.

| Image     | Encoding Improvement | Decoding Improvement | Compression    |
|-----------|----------------------|----------------------|----------------|
| lena      | 14.89%               | 10.53%               | -6.65%         |
| airfield  | 12.89%               | 10.47%               | -5.54%         |
| couple    | 16.58%               | 15.00%               | -3.33%         |
| crowd     | 13.48%               | 8.33%                | -16.12%        |
| fruits    | <b>-60.67%</b>       | <b>-55.56%</b>       | <b>-60.49%</b> |
| goldhill  | 15.63%               | 16.50%               | -2.38%         |
| lax       | 16.08%               | 14.22%               | -4.09%         |
| man       | 15.79%               | 12.63%               | -7.89%         |
| peppers   | 15.79%               | 12.18%               | -10.43%        |
| sail_boat | 16.41%               | 13.04%               | -7.75%         |
| splash    | 16.67%               | 13.89%               | -6.36%         |
| Tiffany   | 16.22%               | 13.16%               | -8.56%         |
| woman     | 16.67%               | 16.00%               | -7.23%         |

Table 2 Modified JPEG-LS Performances

Except for one image “fruits”, all other images have significant improvements on both encoding and decoding time. The compression of our algorithm is worse than the original algorithm.

The following table show shows the improvement of our searching algorithm over the “decompress-then-searching” method.

| Image     | Searching Improvement |
|-----------|-----------------------|
| lena      | 25.64%                |
| airfield  | 27.27%                |
| couple    | 31.71%                |
| crowd     | 27.03%                |
| fruits    | <b>-21.05%</b>        |
| goldhill  | 29.27%                |
| lax       | 28.26%                |
| man       | 28.57%                |
| peppers   | 28.22%                |
| sail_boat | 26.89%                |
| splash    | 32.43%                |
| Tiffany   | 30.77%                |
| woman     | 27.80%                |

Table 3 The Improvement of Searching Time

Again, except for the “fruits” image, the searching time for all other images are significantly improved.

Finally we plot all results in one diagram and thus the readers can compare the benefits and the tradeoff.

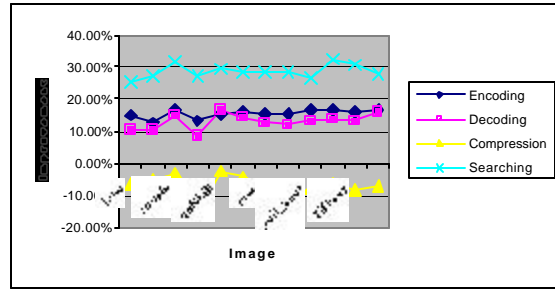


Figure 5 The Overall Comparison

## 5 Conclusion and Future Work

We could further improve the above algorithm and perform the searching purely in the compressed domain. Actually the main problem that kept us from doing a fully compressed matching is the Golomb-Rice coding parameter  $k$ , which uses an accumulate residual and thus prevent the random access to the compressed data. We may use some coding parameter that does not use the accumulated information. This is one work to do in the future.

The above algorithms can also be used for lossless JPEG and CALIC algorithm since all these prediction algorithms work in a similar way. Lossless JPEG may be the easiest one to be modified for CPM. CALIC may give a better compression.

We also need to try larger patterns. In that case, the searching time may become more important and we may need to use faster searching algorithms such as Boyer-Moore algorithm.

Finally, as we have stated in the paper, the searching algorithm we proposed is suitable for “enhanced intensity pattern searching”. We may do some more works such as identifying the applications, on it.

## Reference:

[Adjeroh2001] D.A. Adjeroh, A. Mukherjee, M. Powell, T.C. Bell and N. Zhang, "Pattern matching in BWT-compressed text", (Submitted to DCC2002), available at:

([http://www.csee.wvu.edu/~adjero/progress/cpmPapers/searchBWT\\_qgrams.pdf](http://www.csee.wvu.edu/~adjero/progress/cpmPapers/searchBWT_qgrams.pdf)).

[Amir96] "Let sleeping files lie: Pattern matching in Z-compressed files", A. Amir, G. Benson and M. Farach, *Journal of System Sciences*, **52**: 299-307, 1996.

[BaezaYates88] R. Baeza-Yates and G. H. Gonnet, “Fast string matching with  $k$  mismatches”, *Data Structuring*

Group Research Report CS-88-36, Institute of Computer Research, University of Chile, September, 1988

[Bell2001] T.C. Bell, D.A. Adjeroh and A. Mukherjee, "Pattern matching in compressed text and images", May 2001, available at: (<http://www.csee.wvu.edu/~adjero/progress/cpmPapers/acmSurvey2001.pdf>).

[Bell2001-2] Tim Bell, Matt Powell, Amar Mukherjee and Don Adjeroh "Searching BWT Compressed Text with the Boyer-Moore Algorithm and Binary Search", (Submitted to DCC2002), available at: <http://www.cosc.canterbury.ac.nz/tim/tmp/dcc02match.pdf>.

[Boyer77] R.S. Boyer and J.S. Moore, "A fast string searching algorithm", *Communications of the ACM*, **20**(10), 762-772, 1977.

[Bunke93] H. Bunke and J. Csirik, "An algorithm for matching run-length coded strings", *Computing*, **50**: 297-314, 1993.

[Bunke95] H. Bunke and J. Csirik, An improved algorithm for computing the edit distance of run-length coded strings, *Information Processing Letters*, **54**: 93-96, 1995.

[Cleary84] J.G. Cleary and I.H. Witten, "Data compression using adaptive coding and partial string matching", *IEEE Transactions on Communication*, **32**(4), 396-402.

[Cleary97] J.G. Cleary and W.J. Teahan, "Unbounded length contexts for PPM", *The Computer Journal*, **40**(2/3), 67-75, 1997

[Farach98] M. Farach and M. Thorup, "String matching in Lempel-Ziv compressed strings", *Algorithmica*, **20**: 388-404.

[Howard93] P.G. Howard and J.S. Vitter, "Fast and efficient lossless image compression", *Proceedings, Data Compression Conference*, pp. 351-360, 1993.

[Karp81], R. M. Karp and M. O. Rabin, "Efficient randomized pattern matching algorithms", *Technical Report*, (TR-31-81), Aiken Computation Lab, Harvard University, 1981.

[Knuth77] D.E. Knuth, J.H. Morris and V.R. Pratt, "Fast pattern matching in strings", *SIAM Journal of Computing*, **6**(2), 323-350, 1977.

[Karpinski95] M. Karpinski, W. Plandowski and W. Rytter, "The fully compressed string matching for

Lempel-Ziv encoding", *Technical Report*, (85132-CS), Department of Computer Science, University of Bonn", April, 1995, available at: (<ftp://theory.cs.uni-bonn.de/pub/reports/cs-reports/1995/85132-cs.ps.gz>)

[Manber97] U. Manber, "A text compression scheme that allows fast searching directly in the compressed file", *ACM Transactions on Information Systems*, **15**(2), 124-136, 1997.

[Mandal99] M. K. Mandal and F. Idris and S. Panchanathan, "A critical evaluation of image and video indexing techniques in the compressed domain", *Journal of Image and Vision Computing*, **17**(7), 513-529, 1999.

[Mukherjee94] A. Mukherjee and T. Acharya, "Compressed pattern-matching", *Proceedings, Data Compression Conference*, pp. 468, 1994.

[Mukherjee95] A. Mukherjee and T. Acharya, "VLSI Algorithms for compressed pattern search using tree based codes", *Proceedings, International Conference on Application Specific Array Processors*, pp.133-136, 1995.

[Navarro99] G. Navarro and M. Raffinot, "A general practical approach to pattern matching over Ziv-Lempel compressed text", *Proceedings, Combinatorial Pattern Matching*, **LNCS 1645**: 14-36, 1999.

[Shibata99], Y. Shibata, T. Kida, S. Fukamachi, T. Takeda, A. Shinohara, S. Shinohara and S. Arikawa, "Byte-pair encoding: An text compression scheme that accelerates pattern matching", *Technical Report*, Department of Informatics, Kyushu University, Japan, 1999.

[Shibata99-2], Y. Shibata, M. Takeda, A. Shinohara and S. Arikawa, "Pattern matching in text compressed by using antictionaries", *Proceedings, Combinatorial Pattern Matching*, **LNCS 1645**: 37-49, 1999.

[Weinberger2000], M.J. Weinberger, G. Seroussi and G. Sapiro, The LOCO-I lossless image compression algorithm: Principles and standandization into JPEG-LS", *IEEE Transactions on Image Processing*, **9**(8), 2000.

[Witten99], I. H. Witten, A. Moffat and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufman, 2<sup>d</sup> edition, 1999.