Available online at www.sciencedirect.com
ScienceDirect
J. Vis. Commun. Image R. 18 (2007) 35–44
JOURNAL OF VISUAL Communication & IMAGE Representation
www.elsevier.com/locate/jvci

Grayscale true two-dimensional dictionary-based image compression

Nathanael J. Brittain, Mahmoud R. El-Sakka *

Computer Science Department, The University of Western Ontario, London, Ont., Canada N6A 5B7

Received 13 September 2005; accepted 12 September 2006
Available online 25 October 2006

Abstract

Dictionary-based encoding methods are popular forms of data compression. These methods were initially implemented to reduce the one-dimensional correlation in data, since they are designed to compress text. Therefore, they do not take advantage of the fact that adjacent pixels in images are correlated in two dimensions. Previous attempts have been made to adapt dictionary-based compression schemes to consider the two-dimensional nature of images, but mostly for binary images. In this paper, a two-dimensional dictionary-based lossless image compression scheme for grayscale images is introduced. The proposed scheme reduces correlation in image data by finding two-dimensional blocks of pixels that are approximately matched throughout the data and replacing them with short codewords. Test results show that the compression performance of the proposed scheme outperforms and surpasses any other existing dictionary-based lossless compression scheme. The results also show that it slightly outperforms JPEG-2000s compression performance, when it operates in its lossless mode, and it is comparable to JPEG-LS's and CALIC's compression performance, where JPEG-2000 and JPEG-LS are the current image compression standards, and CALIC is a Context-based Adaptive Lossless Image Coding scheme.

Keywords: Image encoding; Lossless compression; Dictionary-based schemes; Two-dimensional compression; LZ schemes; Prediction

1. Introduction

In the field of image compression there are two major approaches, lossless and lossy. In lossless compression, when an image is compressed and then decompressed, the reconstructed image is an exact copy of the original. In lossy compression, some information about the image is discarded in order to achieve better compression. This means only a close replica of the original image can be retrieved from the compressed data. The compression scheme presented in this paper is a lossless scheme.

Among the most popular methods of lossless compression are dictionary-based schemes. Dictionary compressors encode a string of data by partitioning the string into many sub-strings, and then replacing each sub-string by a codeword. Communication between the compressor and decompressor is done using messages. Each message consists of a codeword and possibly other information. The dictionary in these schemes is the set of every possible codeword. LZ77 [1] and LZ78 [2] are two of the most famous dictionary-based compression schemes.

In LZ77, the dictionary is a portion of the most recently encoded data. This is also called the search buffer. Codewords for sub-strings are pointers to the longest match for the sub-string found in the search buffer. Each message consists of the codeword for the sub-string, the length of the match and the code of the next symbol.

There are many modifications to the original LZ77 scheme. Rodeh et al. introduced LZR [3], a scheme that uses LZ77 but with variable-size pointers. This means the pointer can index a sub-string anywhere in the previously encoded data, rather than just a previous portion. Storer and Syzmanski introduced LZSS [4], in which a flag bit is used to distinguish two types of messages, a pointer or a character. Bell introduced LZB [5], which also uses LZSS but with variable sized pointers as in LZR. LZH [6] is similar to LZSS, but it uses a Huffman encoder [7] to further

* Corresponding author. Fax: +1 519 661 3515.
E-mail address: elsakka@csd.uwo.ca (M.R. El-Sakka).

1047-3203/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jvcir.2006.09.001

compress the generated messages. In software, the PNG file format is based on LZ77.

The LZ77 approach assumes that similar patterns occur close to each other. In situations where a pattern repeats over a period larger than the search buffer size, the repetition cannot be taken advantage of. In LZ78 [2], no search buffer, is used. Instead, the dictionary in this scheme is an indexed list of some previously encountered sub-strings. In LZ78, each codeword consists of two parts, a pointer to the dictionary and the code of the next symbol.

As in LZ77, there are many modifications to the original LZ78 scheme. Welch introduced LZW [8], which is similar to LZ78, but its dictionary initially contains an entry for every possible symbol. Thus, LZW eliminated the need to include the code of the next symbol in messages. Miller and Wegman [9] introduced LZMW. LZMW is similar to LZW but is slightly modified when adding dictionary entries. Where LZW composes the last codeword sent with the next symbol to be encoded, LZMW composes the last codeword sent with the entire next codeword. Jakobsson introduced LZJ [10], which is similar to LZW but when the dictionary becomes full, codewords that have been used the least are replaced. Tischer introduced LZT [11]. In this scheme, the dictionary entries are arranged according to their recent use. When the dictionary becomes full, each new entry replaces the least recently used entry. In software, Unix Compress and the GIF file format are based on LZW.

Fiala and Greene introduced LZFG [12], which is similar to LZ77 because it uses a sliding window but also similar to LZ78 because only particular codewords are stored in the dictionary.

LZ77, LZ78, and their variants, take advantage of the fact that adjacent data values are highly correlated. These dictionary-based schemes are designed to compress text and so only reduce one-dimensional correlations in data. Therefore, they do not take advantage of the fact that adjacent data values (pixels) in images are highly correlated in two dimensions.

There have been few attempts to adapt LZ compressors to suit the two-dimensional nature of images. Perhaps the most straightforward attempt was to find a way to linearize the data and then use a one-dimensional compressor on the data [13]. However, tests showed that no one linearization is best for all images.

Storer and Helfgott [14] and Rizzo et al. [15] present a generalization of LZ77 to lossless compression of *binary* images. The algorithm, known as *two-dimensional sliding window block matching*, uses a wave heuristic to scan the image and a multi-shape *two-dimensional suffix trie* data structure to represent the dictionary, which is a window in previously encoded pixels. However, it is likely that this scheme will not perform well in the case of grayscale or color images, since the chances of finding large *exact* matches would be very small.

Dai and Zakhor [16] present a two-pass two-dimensional LZ77-based scheme for *binary* images. In this scheme, pixels are encoded by searching for an *exact* matching between these pixels and the already encoded pixels. Once such a match is found, these matched pixels are represented by the match location information. As in Storer and Helfgott and Rizzo et al. schemes, it is likely that, if this scheme is applied on grayscale or color images, it would not achieve very good results, due to the small chances of finding large *exact* matches.

Alzina et al. [17] introduced a *lossy* two-dimensional pattern matching compression scheme (2D-PMC) that is based on LZ77. The central part of this scheme is the search, in all previously encoded pixels, for approximate matches of rectangular blocks of certain predetermined sizes (typically 2×3, 3×2, 1×5, or 5×1). A block is encoded by a reference pointer to the previously encoded occurrence that produces the least error.

The dictionary-based scheme presented in this paper is designed to take advantage of the two-dimensional correlation between pixels in grayscale images. It is similar to Storer and Helfgott [14], Rizzo et al. [15], and Dai and Zakhor [16] two-dimensional dictionary encoding schemes, but it allows f*or approximate* matches since it is designed to compress grayscale images. It is also similar to Alzina et al. [17] 2D-PMC scheme, but it considers *any* rectangular block sizes. Moreover, the proposed scheme encodes residuals as well to produce lossless results.

The rest of this paper is organized as follows. Section 2 describes the proposed scheme in detail. Section 3 presents the results. Finally, Sections 4 and 5 offer the suggested future work and the conclusions of this paper, respectively.

2. The proposed GS-2D-LZ scheme

In this paper, a novel grayscale two-dimensional Lempel–Ziv image compression scheme (denoted GS-2D-LZ) is proposed. This scheme is designed to take advantage of the two-dimensional correlations in the image data. It relies on three different compression strategies, namely: two-dimensional block matching, prediction, and statistical encoding. The basic idea of the two-dimensional block matching encoding is to represent a block of uncompressed pixels by a pointer to the best approximate occurrence of that block in the compressed part of the image. This should reduce the interpixel redundancy of the block. In the case of not finding good enough approximate occurrences, prediction is used to reduce the interpixel redundancy of the block. Since GS-2D-LZ is a lossless scheme, residuals generated from approximate matches and predictions are encoded as well. To further compress the image, a statistical encoder is used to reduce the encoding redundancy as much as possible.

2.1. An overview of the GS-2D-LZ scheme

In GS-2D-LZ, an image is encoded in raster scan order processing one block of pixels at each step. For each block of pixels, an approximate match is searched for in previ-

ously encoded data (as with all LZ schemes). The block is encoded as a pointer to the location of this match, the dimensions of the match, and residual information, to ensure that the compression is lossless. If no suitable match can be found, a block of pixels is encoded using a simple prediction scheme. After the entire image is encoded, the match location, match position, residual, and prediction errors are encoded using a statistical compression scheme. A high-level pseudo code for the GS-2D-LZ encoder and decoder are presented in Figs. 1 and 2, respectively. More details on each step of GS-2D-LZ are offered throughout this section.

2.2. Finding and encoding matches

The search area, in which matches are considered for each block, is the rectangular region above and to the left of the block being encoded. The search region is a function of *search-width* and *search-height* parameters, the horizontal and vertical search distances, respectively. The search region is shown in Fig. 3. When searching for a match of a block rooted at the encoder position, each pixel in the search region represents a root of a block at that position. There are search-width × search-height − 1 unique roots to be considered (since the block rooted at the encoder position is not a possibility).

For a particular root pixel in the search region, the algorithm calculates the difference between that pixel and the pixel at the encoder position. To be considered a possible match, the difference between these two pixels cannot exceed the value of a parameter called *threshold*, which is an adjustable parameter that identifies the maximum allowable error between pixels. Fig. 4 shows the procedure for the initial screening of potential matches.

If a pixel is qualified as a root of a potential match, the match is then extended to the right as wide as possible using the same criteria, i.e., in order for the match to be extended one pixel to the right, the difference between corresponding pixels in the potential match and the block being encoded must be less than the *threshold*. Fig. 5 demonstrates extending a potential match to the right.

Once the match has been extended as far as possible to the right (until a mismatch occurs), the match is then extended as far down as possible. For each attempt to extend the height of the match by one, a row of corresponding pixels will be evaluated for a potential match. The pixels in the row are evaluated left to right so that if a mismatch occurs, there is no need to evaluate the rest of the row.

When a mismatch occurs while extending the match downward, the algorithm does not stop the search for a best match rooted in this position. Yet, it reduces the minimum width of the match in order to continue extending downwards.

It is worth mentioning that, in some cases it is possible to extend matches into an un-encoded pixel region, if and only if this un-encoded pixel region is below and to the right of encoder position. In these cases, the decoder can recursively decode these pixels. Fig. 6 shows an example of a match that goes beyond the encoded region and yet the decoder still can reconstruct such block.

```
Begin
  Predict one row of pixels and record each prediction error in
  the prediction error table.
  Advance the encoder in raster scan order to the next un-
  encoded pixel.
  While there are more pixels to be encoded Begin
    Locate the best match in the search region.
    If the match was acceptable Begin
      Record TRUE in the match flag table.
      Update the match dimension, location, and residual tables.
    End
    If the match was unacceptable Begin
      Record FALSE in the match flag table.
      Predict a block of pixels and record the prediction errors in
      the prediction error table.
    End
    Advance the encoder in raster scan order to the next un-
    encoded pixel.
  End
  Statistically encode the matching tables.
  Output the encoded image.
End
```

Fig. 1. High level pseudo code for the GS-2D-LZ encoder.

```
Begin
  Decode the matching tables.
  Predict one row of pixels and correct each prediction error
  using the prediction error table.
  Advance the decoder in raster scan order to the next un-
  decoded pixel.
  While there are more pixels to be decoded Begin
    Read a value from the match flag table.
    If the match flag is TRUE Begin
      Reproduce a block of pixels using the match dimension,
      location and residual tables.
    End
    If the match flag is FALSE Begin
      Predict a block of pixels and correct the prediction errors
      using the prediction error table.
    End
    Advance the decoder in raster scan order to the next un-
    decoded pixel.
  End
  Output the reconstructed image.
End
```

Fig. 2. High level pseudo code for the GS-2D-LZ decoder.

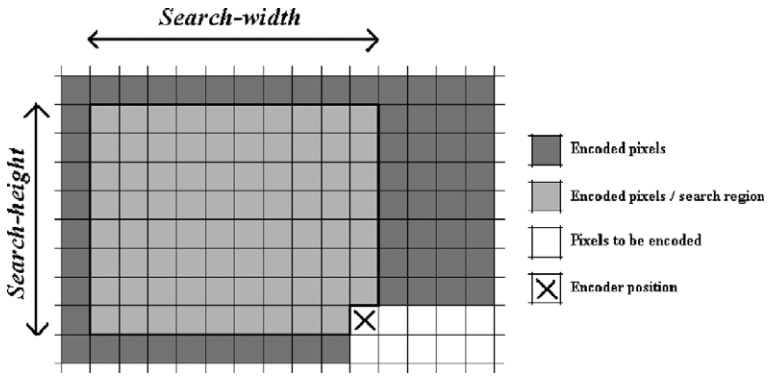

Fig. 3. An example of a search region in GS-2D-LZ.

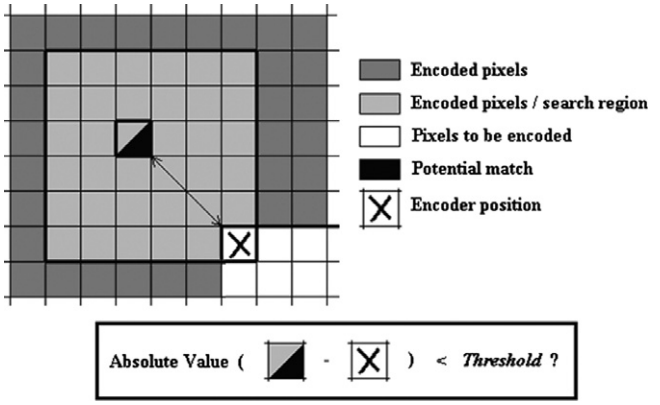

Fig. 4. The initial criteria for screening matches; the roots must match within a certain threshold.

Fig. 7 shows an example of the matching process, where the first row of pixels is extended to a width of seven (the eighth pixel was a mismatch). In each of the second and third rows, all seven pixels of the rows were matches. In the fourth row, the third pixel from the left is a mismatch so the match width is reduced to two. Note that, there is no need to extend the matching width more than the minimum matching width found in the above rows. This is because our goal is to find largest rectangular matching block that has the root pixel in its top left corner. In the fifth, sixth, seventh, and eighth rows the match width is not reduced since no mismatches occur. In the ninth row, the first pixel in the row is a mismatch, so the match cannot be extended downwards any further. In this example, there are two potential matches for this given root pixel; one is seven pixels wide by three pixels high, whereas the other is two pixels wide by eight pixels high.

Potential matches are evaluated according to two measures. The first measure is the size of the matched block, where the match must be large enough such that there are more *new* pixels being encoded than the value of a certain parameter called *minimum-match-size*, which is an adjustable parameter. Because matched blocks can vary in size and shape, encoded blocks will sometimes overlap with previously encoded pixels. In this situation, it does not matter if these pixels match or not, nor do these pixels count as new pixels. For example, in Fig. 8 the last two pixels in each of the first two rows of the match region have already been encoded. Hence, it does not matter if these pixels match or not. Moreover, the number of new pixels in this block is only $(6 \times 4) - 4 = 20$. The second measure for evaluating potential matches is the *mean square error* (MSE), which must be less than the value of a parameter called *max-MSE* (adjustable parameter).

Fig. 9 shows a pseudo code for locating the best approximate match for a given root pixel. This module is considered to be the most time consuming step in the 2D-LZ-GS scheme. However, since we used a small constant-size

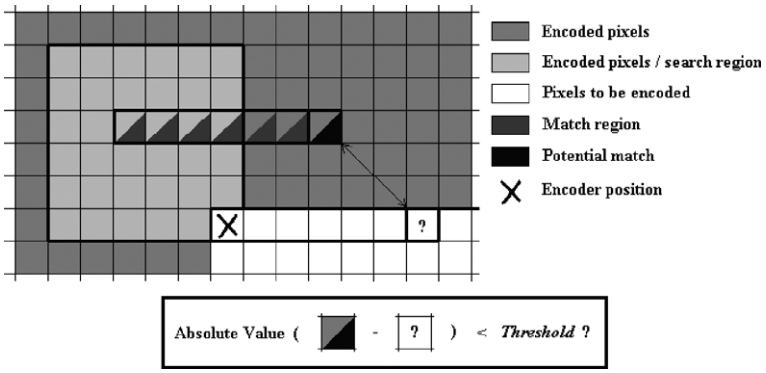

Fig. 5. An attempt to extend a match to the right in GS-2D-LZ; the pixels must match within a certain threshold.

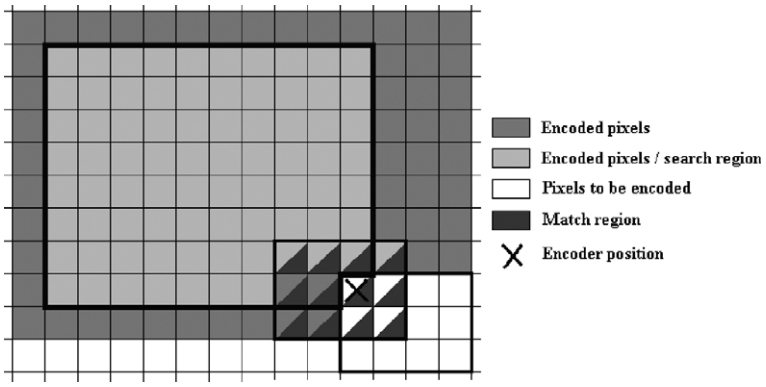

Fig. 6. An example of a match in GS-2D-LZ that extended beyond the encoded pixel region.

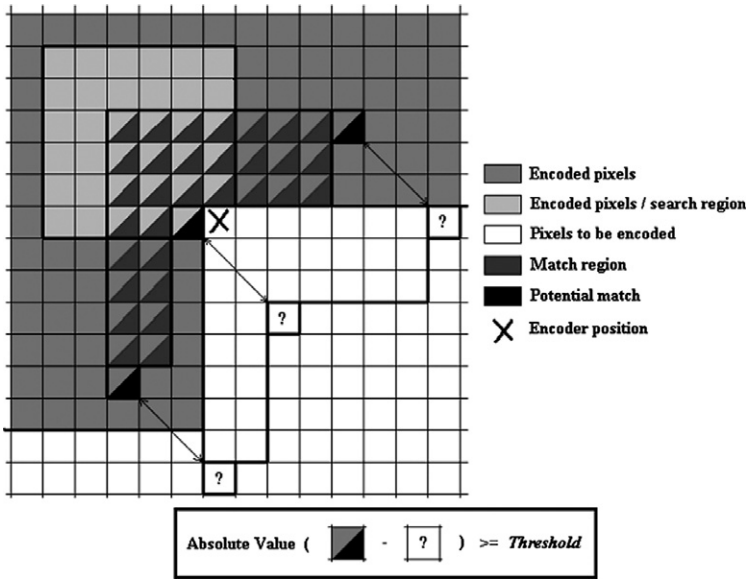

Fig. 7. An example of a match region.

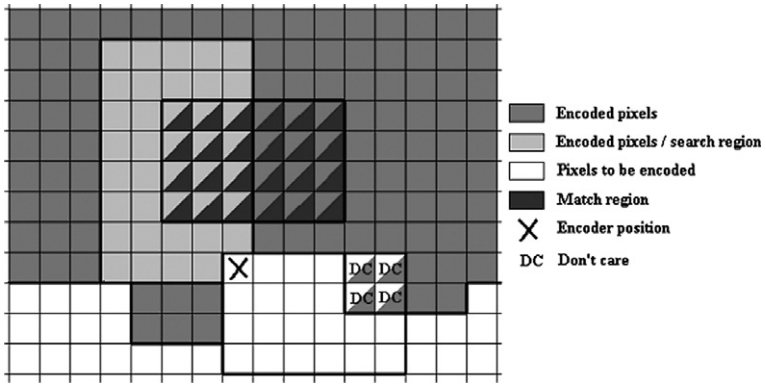

Fig. 8. A match overlaps with previously encoded pixels; in this situation, it does not matter if the overlapping pixels match.

search region, we can state that in the worst case scenario, we only need to apply this module of order O(N), where N is the original image size (i.e., linear running time complexity).

After considering every match rooted at each pixel in the search region, the largest match that has a MSE smaller than *max-MSE* is considered the best match and hence encoded.

```
Algorithm: Locate_the_best_approximate_match_for_a_given_root_pixel
Input:   Encoder_position_row, Encoder_position_col
         Root_pixel_row, Root_pixel_col
Output: Best_match_ height, Best_match_width, Best_match_size
Begin
Best_match_ height = Best_match_width = Best_match_size = 0
Row_offset = 0
Width_of_the_match_in_the_previous_row = Image_width – Encoder_position_col
Repeat
  Col_offset = 0
  Repeat
    If((pixel[Root_pixel_row + Row_offset][Root_pixel_col + Col_offset] already encoded)
       OR (((Root_pixel_row + Row_offset) ≥ Encoder_position_row) AND
            ((Root_pixel_col + Col_offset) ≥ Encoder_position_col))) Begin
      If(pixel[Encoder_position_row + Row_offset][Encoder_position_col + Col_offset]
        already encoded)
        Increment Col_offset
      Else
        Pixel_to_be_encoded =
           pixel[Encoder_position_row + Row_offset][Encoder_position_col + Col_offset]
        Possible_match_pixel =
           pixel[Root_pixel_row + Row_offset][Root_pixel_col + Col_offset]
        If(|Pixel_to_be_encoder – Possible_match_pixel| ≤ threshold)
          Increment Col_offset
        Else
          Break the internal Repeat loop
        End
      End
    End
  Until(Col_offset == Width_of_the_match_in_the_previous_row)
  Width_of_the_match_in_the_previous_row = Col_offset
  Increment Row_offset
  match_size  = the number of possible newly encoded pixels inside the block
                identified by Width_of_the_match_in_the_previous_row × Row_offset
  match_MSE = the MSE between those possible newly encoded pixels and their
                correspoinding pixels inside the matched area
  If((match_size ≥ Best_match_size) AND (match_MSE ≤  max_MSE)) Begin
    Best_match_height = Row_offset
    Best_match_width   = Width_of_the_match_in_the_previous_row
    Best_match_size    = match_size
  End
Until((Width_of_the_match_in_the_previous_row == 0) OR
       (Encoder_position_row + Row_offset) == Image_height)
End
```

Fig. 9. A pseudo code for locating the best approximate match for a given root pixel.

2.3. Raw pixel encoding (no match)

In the case where no match satisfies the two measures described in Section 2.2, the algorithm encodes a small block of pixels rooted at the encoder position, in order to ensure that progress is made. The dimensions of this block are fixed to no-match-block-width × no-match-block-height, which are two adjustable parameters. See Fig. 10.

To keep the algorithm simple, an uncomplicated predictive scheme is used to predict the value of the pixels being encoded, e.g., the *initial* CALIC predictor [18,19], in which the surrounding pixels at the encoder position are utilized to decides whether the data are horizontally or vertically oriented. Based on this decision, a prediction is made for the current pixel being encoded. Fig. 11 shows the context model for predicting pixels, whereas Fig. 12 shows the used prediction scheme.

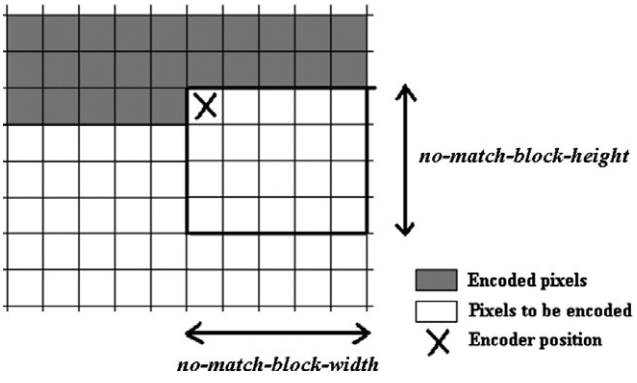

Fig. 10. When no suitable match can be found, a region of pixels specified by *no-match-block-width* and *no-match-block-height* is encoded using a simple predictive encoder.

2.4. Data structures defined

There are five tables that are used to record the matching information. These tables are called: *match flag*, *match location*, *match dimensions*, *residual*, and *prediction errors*.

The *match flag* table contains a Boolean value for each block of pixels, where a value of *true* is recorded in the table when a suitable match for the block is found or *false* otherwise. When a suitable match is found for a block, the position of the match, relative to the block being encoded, is recorded in the *match location* table. At the same time,

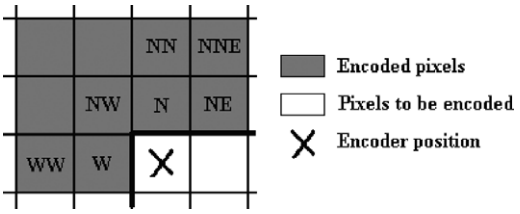

Fig. 11. The context model for predicting pixels.

```
Begin
    Dh = |W – WW| + |N – NW| + |NE – N|
    Dv = |W – NW| + |N – NN| + |NE – NNE|
    Difference_between_Dh_and_Dv = Dh – Dv
    If (Difference_between_Dh_and_Dv > +80) Return N
    If (Difference_between_Dh_and_Dv < –80) Return W
    Prediction = (N + W) / 2 + (NE – NW) / 4
    If (Difference_between_Dh_and_Dv > +32) Return (Prediction + N) / 2
    If (Difference_between_Dh_and_Dv < –32) Return (Prediction + W) / 2
    If (Difference_between_Dh_and_Dv >  +8) Return (3 × Prediction + N) / 4
    If (Difference_between_Dh_and_Dv <  –8) Return (3 × Prediction + W) / 4
    Return Prediction
End
```

Fig. 12. The prediction scheme.

the width and height of the block being encoded are recorded in the *match dimensions* table. The difference between each pixel in the actual block and the corresponding pixel in the matched block is recorded in the *residual* table. When no suitable match could be found for a block, the *prediction errors* table is used to hold the errors between predicted and actual pixel values for each pixel in the block.

2.5. Statistical encoder used

After the entire image has been scanned and the five tables are generated, each table is encoded using a statistical encoder. A benchmark was made using a variety of statistical encoders to choose an encoder that gives the best compression performance for these tables. These encoders include two arithmetic encoder implementations, namely: arib.exe [20] and PAQ6 [21] (predictive arithmetic encoder), two regular Huffman encoder implementations, namely: Huffman.exe and shcodec.exe (static Huffman) [22], and one adaptive Huffman encoder, namely: h2com.exe [23]. It turns out that the best scheme for this job was PAQ6.

The PAQ6 scheme is similar to the well known *prediction by partial matching* (PPM) [24] compression schemes, where both compressors are divided into a predictor and an arithmetic encoder. Yet, the two schemes differ in the way that the next-symbol is predicted, where PAQ6 uses a weighed combination of probability estimates from a large number of models conditioned on different contexts. Moreover, contexts in PPM must be contiguous, whereas in PAQ6 contexts can be any arbitrary functions of the history [25]. These improvements make PAQ6 top ranked on several independent benchmarks. Skibinski et al. [26] empirically compared the performance of various lossless data compression schemes, including PAQ6 and PPMonstr (one of the most efficient PPM implementations). The results (on text data) show that the compression performance of PAQ6 is better than that of PPMonstr, even though it needs more time to execute. In our proposed scheme, PAQ6 is used to compress each of five generated tables.

2.6. Variable settings

There are seven parameters that can be tuned in GS-2D-LZ. These parameters are

- *search-width* and *search-height* (the furthest horizontal and vertical distance a match can be); the larger the match region, the better the approximate match might be found, however a greater number of bits would be required to encode the match pointers,
- *threshold* (the maximum allowable difference between a pixel being encoded and the corresponding pixel within a match); the larger the *threshold* value, the bigger the approximate matches may be found, however more residuals will be needed to be encoded,
- *max-MSE* (the maximum MSE a match can have in order to still be considered a sufficient match), the lower the *max-MSE* value, the fewer the number of bits will be needed to encode the LZ residuals, however LZ compression will be utilized less,
- *min-match-size* (the minimum number of *newly* matched pixels that must be encoded in order to consider the match block sufficiently large enough), the larger the *min-match-size* value, the less of a chance to utilize LZ compression, and the greater the chance to utilize the prediction scheme,
- *no-match-block-width* and *no-match-block-height* (the width and height of the block to be predicatively encoded when no suitable match is found), the larger the *no-match-block*, the less of a chance to utilize LZ compression, and the greater the chance to utilize the prediction scheme.

To find the best setting for these parameters, a training set composed of 24 images is used (six images in four different classes of images namely, *natural scene*, *geographic*, *graphic*, and *medical*). Natural scene images are those of the natural world, e.g., images of people's faces, houses, a hill, and birds. Geographic images are those taken from a high altitude above the earth (usually taken via satellite). Graphic images are those are artificially created by hand or by a computer, e.g., comic strips, charts, graphs, and car-

Table 1
The best settings for the adjustable parameters in GS-2D-LZ

	Natural scene	Geographic	Graphic	Medical
Search-width	4	5	5	4
Search-height	4	5	5	4
Threshold	27	25	25	27
Min-match-size	17	17	21	16
Max-MSE	2.5	3.0	3.8	1.8
No-match-block-width	5	5	5	5
No-match-block-height	5	5	5	5

toons. Medical images are those taken by X-rays, ultrasounds, or magnetic resonance. These classes of images were chosen to represent a variety of image types.

The optimal setting for each parameter was found by exhaustively considering all possible values for a given parameter, while fixing all other parameters. After finding the optimal setting for each parameter, the entire optimization cycle was repeated. This repetition was continued until two consecutive cycles produced no change in the parameters (i.e., optimizing the parameters using successive approximation).

The *search-width*, *search-height*, and *threshold* parameters were tuned between 2 and 63. The *min-match-size* parameter was tuned between 2 and 127. The *max-MSE* parameter was tuned between 1.0 and 5.0. Finally, the *no-match-block-width* and *no-match-block-height* parameters were tuned between 1 and 31. Table 1 shows the best settings for each parameter for each class of images.

3. Experimental results

The compression performance of GS-2D-LZ is compared to other dictionary-based compression schemes as well as state-of-the-art compression schemes. Section 3.1 describes the experimental set-up, while Sections 3.2 and 3.3 present the compression ratio results of GS-2D-LZ versus other dictionary-based schemes and GS-2D-LZ versus state-of-the-art compression schemes, respectively.

3.1. Experimental setup

The GS-2D-LZ scheme, described in Section 2, was tested on a set of 112 different gray scale images divided into: 24 natural scene images (a total of 10.3 MB), 24 geographic images (a total of 12.9 MB), 24 graphic images (a total of 6.88 MB), 24 medical images (a total of 3.31 MB), and 16 standard test images. None of these images were part of the training set that was used to determine the best settings for GS-2D-LZ parameters.

Each test image was compressed and decompressed using GS-2D-LZ and then compared pixel by pixel to the original in order to ensure that the compression was lossless. The compression performance is measured in bits-per-pixel, i.e., the total number of bits in the compressed file divided by the number of pixels in the image.

3.2. Tests versus dictionary-based compression schemes

Since GS-2D-LZ is a dictionary-based scheme, it makes sense to compare its performance with other dictionary-based schemes. In this test, for each of the 4 image classes and the standard test images, the GS-2D-LZ compression performance is compared to that of PNG (which based on LZ77), GIF (which based on LZW), and Unix Compress (which based on LZW). Tables 2 and 3 show the compression performance (in bits per pixel) for the 4 classes of images and the 16 standard test images, respectively.

On average, GS-2D-LZ outperforms each of the other dictionary-based compression schemes in all image classes, as well as on the set of standard test images. Out of 112 images tested, GS-2D-LZ has the best compression performance on 99 of the images. From these results, we can conclude that GS-2D-LZ surpasses the compression performance of any other dictionary-based scheme.

3.3. Tests versus state-of-the-art compression schemes

To demonstrate the potential of dictionary-based compression, the performance of GS-2D-LZ is compared to the current state-of-the-art compression schemes. In this test, for each of the 4 image classes and the standard test images, the GS-2D-LZ compression performance is compared to that of BZIP2 [27], JPEG2000 [28], JPEG-LS [29], and CALIC [18,19]. BZIP2 is based on the Burrows

Table 2
Experimental compression performance (bits per pixel) for GS-2D-LZ and popular dictionary-based compression schemes on 4 classes of images

Image class	GS-2D-LZ	PNG	GIF	Unix-Compress
Natural scene	4.50	4.73	6.83	6.21
Geographic	5.17	5.40	7.37	6.46
Graphic	1.61	1.77	2.73	2.48
Medical	3.25	3.42	4.96	4.58

Table 3
Experimental compression performance (bits per pixel) for GS-2D-LZ and popular dictionary-based compression schemes on 16 standard test images

Image name	GS-2D-LZ	PNG	GIF	Unix-Compress
Baboon—512 × 512	5.84	6.01	8.98	7.84
Barbara—720 × 580	4.75	5.23	8.74	7.73
Boats—720 × 576	4.01	4.33	3.93	6.34
Bridge—512 × 512	5.35	4.94	5.44	5.00
Camera—256 × 256	4.38	4.67	6.77	6.68
Columbia—480 × 480	3.47	3.92	6.61	6.21
Couple—512 × 512	4.69	4.88	7.59	6.82
Crowd—512 × 512	4.05	4.53	6.89	6.01
Lake—512 × 512	5.08	5.37	8.18	7.42
Lax—512 × 512	5.81	5.98	8.64	7.73
Lena—512 × 512	4.06	4.39	7.66	6.68
Man—512 × 512	4.58	4.93	7.90	6.95
Milkdrop—512 × 512	3.74	3.97	6.43	5.97
Peppers—512 × 512	4.65	4.91	4.54	7.22
Woman1—512 × 512	4.75	4.98	6.94	6.14
Woman2—512 × 512	3.37	3.77	6.60	5.73
Weighted average	4.54	4.80	6.99	6.65

Table 4
Experimental compression performance (bits per pixel) for GS-2D-LZ and state-of-the-art compression schemes on 4 classes of images

Image name	GS-2D-LZ	BZIP2	JPEG2000	JPEG-LS	CALIC-h	CALIC-a
Natural scene	4.50	4.88	4.66	4.71	4.50	4.27
Geographic	5.17	5.24	5.31	5.24	5.23	5.06
Graphic	1.61	1.77	2.61	1.90	2.11	1.73
Medical	3.25	3.48	3.22	3.22	3.39	3.23

Table 5
Experimental compression performance (bits per pixel) for GS-2D-LZ and state-of-the-art compression schemes on 16 standard test images

Image name	GS-2D-LZ	BZIP2	JPEG2000	JPEG-LS	CALIC-h	CALIC-a
Baboon—512 × 512	5.84	6.38	5.88	5.82	5.80	5.66
Barbara—720 × 580	4.75	5.92	4.69	4.74	4.63	4.52
Boats—720 × 576	4.01	5.00	4.07	3.93	3.93	3.83
Bridge—512 × 512	5.35	4.30	5.74	5.50	5.53	5.37
Camera—256 × 256	4.38	5.12	4.54	4.31	4.25	4.20
Columbia—480 × 480	3.47	4.45	3.52	3.43	3.55	3.43
Couple—512 × 512	4.69	5.37	4.84	4.68	4.70	4.59
Crowd—512 × 512	4.05	4.71	4.20	3.91	3.87	3.77
Lake—512 × 512	5.08	5.68	5.15	4.98	5.01	4.91
Lax—512 × 512	5.81	6.38	5.96	5.76	5.77	5.63
Lena—512 × 512	4.06	5.07	4.06	3.99	3.93	3.87
Man—512 × 512	4.58	5.49	4.69	4.50	4.43	4.37
Milkdrop—512 × 512	3.74	4.37	3.77	3.63	3.65	3.57
Peppers—512 × 512	4.65	5.37	4.63	4.51	4.55	4.42
Woman1—512 × 512	4.75	5.00	4.81	4.67	4.67	4.55
Woman2—512 × 512	3.37	4.19	3.32	3.30	3.30	3.21
Weighted average	4.54	5.18	4.62	4.48	4.45	4.34

Wheeler transformation, JPEG2000 is the current JPEG lossy compression standard but is operated in its lossless mode, JPEG-LS is the current JPEG lossless compression standard, and CALIC is a Context-based Adaptive Lossless Image Coding scheme. In CALIC-h, a Huffman encoder is internally used, whereas in CALIC-a, an arithmetic encoder is internally used. Tables 4 and 5 show the compression performance (in bits per pixel) for the 4 classes of images and the 16 standard test images, respectively.

On average, GS-2D-LZ outperforms each of the state-of-the-art compression schemes in the graphic image class. It also outperforms each of the state-of-the-art compression schemes (except CALIC-a) in natural scene and geographic image classes. In the class of medical images, GS-2D-LZ is outperformed (on average) by only a margin of 0.03 bits-per-pixel by JPEG2000, LPEG-LS, and CALIC-a. Yet, it outperforms both BZIP2 and CALIC-h. On the set of standard test images, GS-2D-LS scores lower than JPEG-LS and CALIC, but only by a small margin of bits-per-pixel. From these results it can be concluded that the compression of GS-2D-LZ is at least comparable to that of the state-of-the-art compression schemes, if not better.

4. Future work

It is worth mentioning that, the compression/decompression processes in the GS-2D-LZ scheme are slower than that in the schemes mentioned in Section 3. This flaw can be attributed to the time complexity of the PAQ6 scheme and the fact that all other compression schemes are heavily optimized, while the proposed scheme is not, since it was implemented as a proof of concept. Currently, we are working on optimizing the code, especially the pattern matching module, to reduce the execution time. We also are experimenting with other arithmetic encoders to improve both the compression and the time performance of the proposed scheme.

Currently, the 2D-LZ-GS scheme has seven user defined adjustable parameters. Further work need to be done to automate the selection of these parameters.

5. Conclusions

In this paper, a novel *true* two-dimensional dictionary-based scheme is introduced. Experimental results showed that the compression performance of the GS-2D-LZ scheme outperforms and surpasses any other dictionary-based compression scheme. Furthermore, its performance is comparable to JPEG2000, JPEG-LS, and CALIC. This implies that dictionary-based compression schemes can be as efficient as the current state-of-the-art compression schemes. Hence, further research in this area would definitely be worth while.

Acknowledgments

This research is partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

This support is greatly appreciated. Authors thank anonymous reviewers for their constructive comments that indeed help improving the presentation of this paper big time.

References

[1] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Transactions on Information Theory 23 (3) (1977) 337–343.
[2] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory 24 (5) (1978) 530–536.
[3] M. Rodeh, V. Pratt, S. Even, Linear algorithm for data compression via string matching, Journal of the ACM 28 (1) (1981) 16–24.
[4] J. Storer, T. Syzmanski, Data compression via textual substitution, Journal of the ACM 29 (4) (1982) 928–951.
[5] T. Bell, Better OPM/L text compression, IEEE Transactions on Communications 34 (12) (1986) 1176–1182.
[6] R. Brent, A linear algorithm for data compression, Australian Computer Journal 19 (2) (1987) 64–68.
[7] D. Huffman, A method for the construction of minimum redundancy codes, Proceedings of the IRE 40 (9) (1952) 1098–1101.
[8] T. Welch, A technique for high-performance data compression, IEEE Computer 17 (6) (1984) 8–19.
[9] V. Miller, M. Wegman, Variations on a scheme by Ziv and Lempel, Combinatorial Algorithms on Words, NATO ASI Series, 1984, F12:131–140.
[10] M. Jakobsson, Compression of character strings by an adaptive dictionary, BIT Numerical Mathematics 25 (4) (1985) 593–603.
[11] P. Tischer, A modified Lempel–Ziv–Welch data compression scheme, Australian Computer Science Communications 9 (1) (1987) 262–272.
[12] E. Fiala, D. Greene, Data compression with finite windows, Communications of the ACM 32 (4) (1989) 490–505.
[13] A. Amir, G. Landau, D. Sokol, Inplace 2D matching in compressed images, Journal of Algorithms 49 (2) (2003) 240–261.
[14] J. Storer, H. Helfgott, Lossless image compression by block matching, The Computer Journal 40 (2-3) (1997) 137–145.
[15] F. Rizzo, J. Storer, B. Carpentieri, LZ-based image compression, Information Sciences 135 (1–2) (2001) 107–122.
[16] V. Dai, A. Zakhor, Lossless layout compression for maskless lithography, Proceedings of the SPIE 3997 (2000) 467–477.
[17] M. Alzina, W. Szpankowski, A. Grama, 2D-pattern matching image and video compression: theory, algorithms, and experiments, IEEE Transactions on Image Processing 11 (3) (2002) 318–331.
[18] X. Wu, An algorithmic study on lossless image compression, Proceedings of the 1996 IEEE Data Compression Conference (1996) 150–159.
[19] X. Wu, N. Memon, Context-based, adaptive, lossless image coding, IEEE Transactions on Communications 45 (3) (1997) 437–444.
[20] D. Scott, David Scott's bijective arithmetic encoder, http://bijective.dogma.net/compres10.htm, 2001.
[21] M. Mahoney, The PAQ6 data compression program, http://www.cs.fit.edu/~mmahoney/compression, 2004.
[22] A. Moffat et al., SHCODEC, http://webcenter.ru/~xander, 2002.
[23] D. Scott, David' Scott's Bijectified Vitter Adaptive Compression, http://bijective.dogma.net/compress2vh.htm, 2002.
[24] T. Bell, J. Cleary, I. Witten, Data compression using adaptive coding and partial string matching, IEEE Transactions on Communications 32 (4) (1984) 396–402.
[25] M. Mahoney, Adaptive Weighing of Context Models for Lossless Data Compression, Florida Tech. Technical Report CS-2005-16, 2005.
[26] P. Skibinski, S. Grabowski, S. Deorowicz, Revisiting dictionary-based compression, Software—Practice and Experience 35 (2005) 1455–1476.
[27] M. Burrows, D.J. Wheeler, A block-sorting lossless data compression algorithm, Digital SRC Research Report 124 (1994).
[28] D. Taubman, M. Weinberger, JPEG2000: image compression fundamentals, Standards and Practice, Kluwer Academic Publishers, Dordrecht, 2002.
[29] M. Weinberger, G. Seroussi, G. Sapiro, The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS, IEEE Transactions on Image Processing 9 (8) (2000) 1309–1324.