Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

On the Lossless Compression of Still Images[♥]

Trees-Juen Chuang and Ja-Chen Lin

Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 30050
R.O.C.

Abstract

This paper presents a new algorithm for lossless still image compression using a new scheme: base-switching (BS). The given image is partitioned into non-overlapping fixed-size subimages. Different subimages then get different compression ratios according to the base values of the subimages. In order to increase the to compress gray-value images (or a color component of color images). In Section 2 we review the JBIG and Lossless compression ratio, a hierarchical technique is also used. It is found that the compression ratio of the proposed algorithm can compete with that of the international standard algorithms known as JBIG and Lossless JPEG. The math theory needed to build up the proposed compression scheme is also provided.

1. Introduction

There are many algorithms for lossy compression of still image compression and they usually achieve very high compression ratio.[1] These algorithms usually assume that the reconstructed images will let human eyes feel no difference. However, in certain situations, lossy compression is inappropriate due to the need of exact fidelity or legality. For example, if we get many unidentified images which cannot be analyzed immediately due to the lack of suitable analyzer on the scene, then we cannot use lossy compression algorithms to compress images. (This kind of application did occur in, say, satellite or medical image processing.) The reason lossy compression is not suitable in this case is that they might ignore some important information imperceptibly, and the lost information cannot be recovered. Note that the need of lossless compression might also arise in the application where some kinds of lossy compression has already been done and further lose is not desired.[2]
Since many lossless compression algorithms have been developed to compress black-and-white (binary) images (the international standards for binary images include the compression algorithms MH,[3] MR,[3] MMR,[4] JBIG,[5,6] and so on), we only discuss in this paper the gray-value images, and present a new lossless method

JPEG for gray-value images. We then present our new algorithm in Section 3. The experimental results and time complexity analysis are provided in Section 4. The comparisons with JBIG and Lossless JPEG are also included there. Concluding remarks are in Section 5.

2. A short review of the JBIG and Lossless JPEG for gray-value images

Two lossless still image compression algorithms, JBIG and Lossless JPEG, have recently become international standards. The algorithms are the special cases of the parameterizable JBIG[5-7] and JPEG[8-11] standards, respectively.

JBIG (Joint Bi-level Image expert Group coding) was defined in CCITT Recommendation T.82, which for gray-level coding breaks images down into the "bit-planes" of the images, and then compresses these bit-planes with its binary algorithm (the algorithm defined in CCITT T.82 for binary compression uses an adaptive 2D coding model, followed by an adaptive arithmetic coder[12]).

JPEG (Joint Photographic image Expert Group) was defined in CCITT Recommendation T.81. For lossless coding (this differs from the lossy mode of JPEG well-known to most of the people; the JPEG that we mention here is in its lossless mode and hence does not require the use of the Discrete Cosine Transform (DCT) coding[13]), JPEG utilizes a customizable from of Differential Pulse Code Modulation (DPCM) coding[14] and a variable-length

[♥]This work was supported by the National Science Council, Republic of China, under contract NSC 86-2213-E009-108.

representation of the DPCM errors.[15] There are two choices — custom Huffman or adaptive arithmetic coder — to follow this model.

3. The proposed algorithm

3.1. *System Overview*

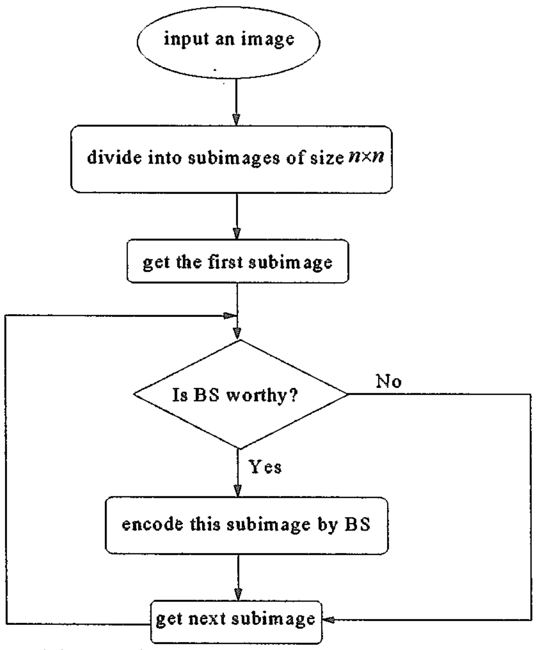


Fig. 1. The flowchart of the proposed Base Switching (BS) method.

As shown in Fig. 1, we first divide the original image (gray-level data) into subimages of size n×n. The subimages are then processed one by one. For each subimage, we have to determine whether the proposed base-switching (BS) algorithm is worthy to apply to the subimage or not. In other words, if the proposed BS will cause data explosion, i.e., will cause the b.p.p. (bits per pixel) be not less than 8 for this gray level subimage, then we skip this subimage because the whole subimage will be transmitted in the traditional pixel-by-pixel manner (n×n pixels, and each pixel has 8 bits). On the other hand, if the data explosion does not occur, then the proposed BS is used to transmit this subimage. Of course, an extra bit is needed to indicate whether the subimage is encoded (by the proposed BS) or not. (Therefore, the total number of bits needed to transmit a non-BS subimage is one more bit than that of the traditional pixel-by-pixel manner.) Throughout this paper, the subimage size used is 3×3 for the efficiency of compression ratio.

3.2. *Encoding a subimage by BS*

g_0	g_1	g_2
g_3	g_4	g_5
g_6	g_7	g_8

Fig. 2. An arbitrary given subimage A.

g'_0	g'_1	g'_2
g'_3	g'_4	g'_5
g'_6	g'_7	g'_8

Fig. 3. Subimage A' Each g'_i is $g_i - m$, $i = 0,1,\cdots,8$.

Given a 3x3 subimage A, whose nine gray values are $g_0, g_1, \cdots, g_8$ (see Fig. 2). Define the "minimum" m, "base" b, and the "value-reduced subimage" A' (see Fig. 3) of the subimage A by

$$m = \min_{0\le i\le 8} g_i, \qquad (1)$$

$$b = \max_{0\le i\le 8} g_i - \min_{0\le i\le 8} g_i + 1, \text{ and} \qquad (2)$$

$$A'_{3\times3} = A_{3\times3} - \begin{bmatrix} m & m & m \\ m & m & m \\ m & m & m \end{bmatrix}, \qquad (3)$$

respectively. Note that (3) means that

$$g'_i = g_i - m \quad \text{for all } i=0, 1, 2,\cdots,8. \qquad (4)$$

Also denote that

$$\min_{0\le i\le 8} g'_i = 0 \text{ and } \max_{0\le i\le 8} g'_i = b-1. \qquad (5)$$

Therefore, the 9-dimensional vector $A' = (g'_0, g'_1, \cdots, g'_8)$ can be treated as a 9-digit number $(g'_0 g'_1 \cdots g'_8)_b$ in the base-b number-system. For convenience, let $V_{3\times3}$ be the collection of all 3×3 subimage A', and the base-set B = {1, 2, 3, ⋯, 256}. Then we define an integer-value function $f: V_{3\times3}\times B \rightarrow$ {non-negative integers} by

$f(A',b) =$ the decimal integer equivalent to the base-b number $(g'_0 g'_1 \cdots g'_8)_b$

$$= \sum_{i=0}^{8} g'_i \times b^i \qquad (6)$$

$$= (\cdots((g'_0\times b+g'_1)\times b+g'_2)\times b+\cdots)\times b+g'_8. \qquad (7)$$

It is easy to prove the following two properties.

Property 1. The inequality $f(A',b) < b^N$ always holds. Here, $N = n^2$ is the number of pixels in the subimage A'.

Proof of Property 1.

By Equations (6) and (5), we have

$$f(A',b)=\sum_{i=0}^{N-1} g'_i\times b^i \le \sum_{i=0}^{N-1}(b-1)\times b^i=(b-1)\sum_{i=0}^{N-1}b^i=(b-1)\times\frac{b^N-1}{b-1}$$

$$= b^N - 1 < b^N \qquad \square$$

Property 2. For each base b, and for each given integer λ satisfying $b-1 \le \lambda \le \sum_{i=0}^{8}(b-1)\times b^i = b^N - 1$, we can find a unique 3×3 subimage A' such that $f(A',b) = \lambda$.

Proof of Property 2.

Just convert the base-10 number $(\lambda)_{10}$ to a base-*b* number $(g'_0 g'_1 \cdots g'_8)_b$. Note that $\lambda \ge b-1$ is required because Equation (5) have confined the outlook of A'. □

By Property 1, the number of bits needed to store the integer $f(A',b)$ using a binary number is therefore at most

$$Z_b = \lceil log_2 b^9 \rceil. \tag{8}$$

When we want to reconstruct $A' = (g'_0, g'_1, \cdots, g'_8)$, all we have to do is to switch that binary (base-2) number to a base-*b* number $(g'_0 g'_1 \cdots g'_8)_b$.

3.2.1. *Several possible ways to represent the subimage A' according to the value of base b*

As stated in (5), for each subimage $A' = (g'_0, g'_1, \cdots, g'_8)$, we always have

$$min\{g'_0, g'_1, \cdots, g'_8\} = 0, \text{ and} \tag{9}$$

$$max\{g'_0, g'_1, \cdots, g'_8\} = b-1. \tag{10}$$

Therefore, at least one of the pixels of A' has gray value 0, and at least one of the pixels of A' has gray value b-1. There are at least two ways to store A'. The first way is as stated at the end of Sec. 3.1, namely, to store

$$b \text{ and a binary-equivalent of } (g'_0 g'_1 \cdots g'_8)_b. \tag{11}$$

The second way is to store

$$\{b;\ i_{min};\ i_{max}\} \text{ and } \{g_i | i \ne i_{min}, i \ne i_{max}\}. \tag{12}$$

(Here, $i_{min} \in \{0,1,\cdots,8\}$ is such that $g'_{i_{min}} = 0$, and $i_{max} \in \{0,1,\cdots,8\}$ is such that $g'_{i_{max}} = b-1$. If more than one i in $\{0,1,\cdots,8\}$ have there g'_i value being 0, say, $g'_2 = g'_3 = g'_5 = 0$, then use the smallest i as i_{min} (hence, $i_{min} = 2$ in this case). An analogous statement making i_{max} unique can be stated likewise.) We analyze below which of the two ways ((11) vs. (12)) would save more storage space. First, we reduce (12) to a simpler form by Lemma 1 below.

Lemma 1. In the storage system (12), we can use 7 bits to indicate the positions of the pair (i_{min}, i_{max}).

Proof.

Because the size of the block A' is 3×3, we have $0 \le i_{min} \le 8$ and $0 \le i_{max} \le 8$. As a result, there are $9 \times 8 = 72$ possible combinations of the pair (i_{min}, i_{max}). Since $2^6 < 72 < 2^7$, we can use 7 bits to indicate the combination (and hence, the location among the nine pixels) of (i_{min}, i_{max}) pair. □

With Lemma 1, we know that (12) can be rewritten as

$$\{b\ ;\ a\ 7\text{-bit key to get } (i_{min}, i_{max})\}, \text{ and a binary-equivalent of the 7-digit base-}b \text{ number } (g_i | i \ne i_{min}, i \ne i_{max})_b. \tag{13}$$

To know when the storage system (13) can save more memory space than (11) does, we notice that: first, both (13) and (11) needs to store *b*; second, (11) needs $\lceil log_2 b^9 \rceil$ bits to represent a 9-digit number $g'_0 g'_1 \cdots g'_8$ in the base-*b* number system, whereas (13) needs 7 bits to indicate the location of the (i_{min}, i_{max}) pair, and $\lceil log_2 b^7 \rceil$ bits to encode a 7-digit number $g'_0 g'_1 \cdots g'_8$ (with $g'_{i_{min}}$ and $g'_{i_{max}}$ taken away) in the base-*b* number system. ($g'_{i_{min}}$ and $g'_{i_{max}}$ needs no storage if we know the position of i_{min} and i_{max} (see Fig. 4), this is because $g'_{i_{min}} = 0$ and $g'_{i_{max}} = b-1$ always hold by (9) and (10).) The next lemma and property are used to compare the storage system (11) and (13).

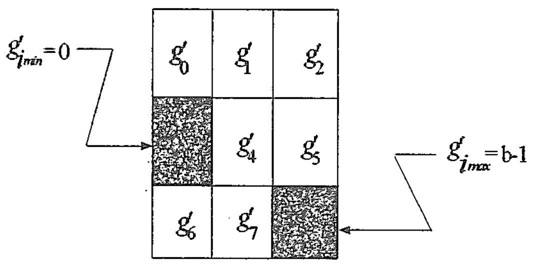


Fig. 4. If the position of i_{min} and i_{max} are known, then only 7 gray values needed to be encoded. Here, $i_{min} = 3$ and $i_{max} = 8$ are known in this example.

Lemma 2. (i). If $b < 2^{3.5} \approx 11.314$, then $7 + log_2 b^7 > log_2 b^9$.

(ii). If $b > 2^{3.5} \approx 11.314$, then $7 + log_2 b^7 < log_2 b^9$.

Proof.

We first prove statement (i). Since $b < 2^{3.5}$, we have $log_2 b < 3.5$; i.e., $2\,log_2 b < 7$; i.e.,

$9\,log_2 b - 7\,log_2 < 7$; i.e., $9\log_2 b < 7 + 7\log_2 b$; i.e., $log_2 b^9 < 7 + 7\,log_2 b$. The second statement can be proved likewise. □

Property 3. Using the storage system (13) is more worthy than using the storage system (11) if and only if $b > 11.314$.

The next concern is to find the condition such that using the storage system (11) or (13) is more worthy than using the "raw" storage system in which $9 \times 8 = 72$ bits are used to store the nine (original) gray values (each is 8-bit) $g_0, g_1, g_2, \cdots$, and g_8 of the subimage *A*. After careful checking, we obtain the following rules to encode a 3×3 subimage.

3.2.2. *Format*

There are three formats to be used in the proposed algorithm. They are:

Rule 1: if $b \in \{1,2,\cdots,11\}$, then the coding format is

1 bit	*7 bits*	*8 bits*	$z_b = \lceil log_2 b^9 \rceil$ *bits*
c	*b*	*m*	binary equivalent of $(g'_0 g'_1 \cdots g'_8)_b$

(This format uses at most $1+7+8+\lceil log_2 11^9 \rceil = 48$ *bits since* $b \le 11$ *.)*

Rule 2: if $b \in \{12,13,\cdots,128\}$, then the coding format is

1 bit	*7 bits*	*8 bits*	*7 bits*	$\tilde{z}_b = \lceil log_2 b^7 \rceil$ *bits*
c	*b*	*m*	*P(min,max)*	binary equivalent of $(g'_i \mid 0 \le i \le 8, i \ne i_{min}, i \ne i_{max})_b$

(This format uses at least $+7+8+7+\lceil log_2 12^7 \rceil = 49$ *bits and at most* $1+7+8+7+\lceil log_2 128^7 \rceil = 72$ *bits.)*

Rule 3: if $b \in \{129,130,\cdots,256\}$, then the coding format is

1 bit	*72 bits*
c	the original nine gray values : $g_0, g_1, \cdots, g_8$

(This format always uses 73 bits.)

Note that *c* stands for the category-bit: if *c* is zero then we encode block *A* by Rule 1 or Rule 2, (according to the value of *b*); if *c* is one, however, Rule 3 is need. Also note that *P(min,max)* denotes which of the $9 \times 8 = 72$ possible position-pair is the actual position of the pair (i_{min}, i_{max}). As for $(g'_i | 0 \le i \le 8, i \ne i_{min}, i \ne i_{max})_b$, it is a 7-digit base-*b* number because the two gray values $g'_{i_{min}}$ and $g'_{i_{max}}$ are taken away. Finally, $m = \min_i g_i$ and $b = \max_i g_i - \min_i g_i + 1$ are as defined in Eq. (1) and (2), respectively. Below we explain why we use Rule 3 instead of Rule 1 or Rule 2 when $b > 128$. If $b \ge 129$, then using the format provided in Rule 1 is not worthy because $1+7+8+7+\lceil \log_2 b^7 \rceil \ge 1+7+8+\lceil \log_2 129^9 \rceil$ is longer than the fixed 73 bits needed in the format given in Rule 3. Similarly, if $b \ge 129$, then $1+7+8+7+\lceil log_2 b^7 \rceil \ge 1+7+8+7+\lceil log_2 129^7 \rceil = 73$ suggested that the format in Rule 2 can not be better than that of Rule 3. Moreover, if *b* is large, say, *b*=200, then $1+7+8+7+\lceil log_2 200^7 \rceil = 77$ is even worse than the 73 required in the format of Rule 3.

We also give here another remark about Rules 1 and 2. Some readers might suggest that one more (sub-category) bit is used to distinguish Rule 1 from Rule 2; then , 4 bits (instead of 7 bits) are used to represent *b* for Rule 1 (whereas 7 bits are still used to represent *b* for Rule 2). However, according to our experiments, this modified approach was found not better than the old one which uses 7 bits to represent *b* for both Rules 1 and 2, especially if the hierarchical structure introduced in Sec. 3.4 was used. The only case that this modified approach (the one using one more (sub-category) bit to distinguish Rule 1 from Rule 2) could perform better occurred only when the hierarchical structure was not used and the image had many large smooth regions. However, since the hierarchical structure can improve the compression ratio, and we wish to handle images of any kind without judging in advance whether the image have large smooth regions or not, we do not intend to use this modified approach.

3.3. *Decoding*

Without the loss of generality, we show below how to reconstruct (decode) the first subimage of an image which has been encoded using Rules 1~3 presented above in Section 3.2. (The remaining subimages can be reconstructed similarly.)

We first check the first bit c. If $c = 1$, then we use the next $8 \times 9 (= 72)$ bits to reconstruct the nine gray values, each is 8-bit, of the subimage. However, if $c = 0$, we use the next 7 bits to obtain the base value b. According to the value of b, there are two subcases to proceed. (Subcase 1): if $b \le 11$, then we take the next 8 bits to obtain the value m; and after that, we take another $\lceil log_2 b^9 \rceil$ bits of the received code to know the binary equivalent of $(g'_0 g'_1 \cdots g'_8)_b$. We can therefore obtain the nine gray values $\{g'_i\}_{i=0}^{8}$ of the subimage A'. Then, with the help of Eq. (4), we can obtain the nine gray values $\{g_i\}_{i=0}^{8}$ of the subimage A. (Subcase 2): if $12 \le b \le 128$, then get the next 8 bits, 7 bits, and $\lceil log_2 b^7 \rceil$ bits, to obtain the values of m, *P(min,max)*, and $(g'_i | 0 \le i \le 8, i \ne i_{\min}, i \ne i_{\max})_b$, respectively. With the help of a predefined position codebook, we can use the value of *P(min, max)*, which is a codeword, to recover the positions of the two pixels where the (reduced) gray values g'_i are minimum and maximum, respectively. By (4), the positions where g'_i become minimum or maximum are also the positions where g_i become minimum or maximum. Therefore, on the two pixels just recovered by the value of *P(min, max)*, the "original" gray values g_i should then be m and $m + b - 1$, respectively, by Eq. (1) and (2). As for the remaining $9 - 2 = 7$ pixels, we can use the next $\lceil log_2 b^7 \rceil$ bits to obtain a binary number. Convert this $\lceil log_2 b^7 \rceil$-digit number in the base-2 system to obtain a 7-digit number in the base-b system. After adding the value m to each of these seven digits, we obtain the seven gray values needed.

3.4. *Hierarchical use of the techniques introduced in Sec. 3.2~3.3*

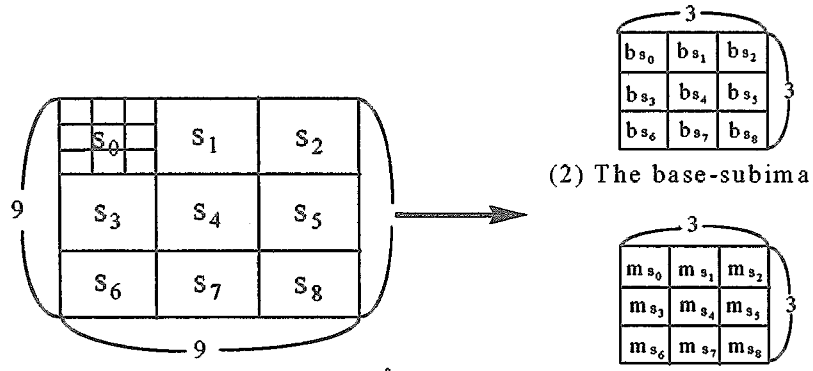
The encoded result of Sec. 3.2 can be compressed further in a hierarchical manner. Consider $3 \times 3 = 9$ adjacent subimages, each subimage is of size 3×3. Then, since each subimage has its own base b, we have nine bases. (If some of these 9 subimages were encoded using Rule 3, for convenience, just "assign" a fixed number to the corresponding bases. In this paper, we set this fixed number as 128, and modify the base-value range of using Rule 2 as 12~127, so that we may completely discard the category bit "c" (see Sec. 3.2.2) for all subimages (because whether Rule 1 (or 2, or 3) is used to encode a specified subimage can be completely determined by the value of base). We then can imagine that there is a so-called "base-image", whose gray values are $b_0, b_1, b_2, \cdots, b_8$; then, since it is a kind of image (except that each value is a base value of a subimage rather than a gray value of a pixel), we can use the technique introduced in Sec. 3.2 to compress these nine base values. The details are omitted.

Besides b, the minimal value m of each block can also be grouped and compressed similarly. In other words, for every $3 \times 3 = 9$ adjacent subimages, we compress their $\{m_0, m_1, \cdots, m_8\}$ by treating $m_0 \sim m_8$ as the nine gray values of an imaginary 3×3 "super" image. (If some of the $3 \times 3 = 9$ subimages that form the super image were encoded using Rule 3, the missing m_i can be arbitrarily assigned, because the decoding of those subimages using Rule 3 will not use m_i at all.)

The compression layer described in the above two paragraphs are called Pass 2, and we can repeat the same procedure to encode in Pass 3 the result of Pass 2. Of course, the higher a layer is, the less the data to be processed.

For decoding, we first decode the highest pass, Pass k, using the method presented in Sec. 3.3, and then decode Pass k−1, and then decode Pass k−2, and so on. For example, look at the 9×9 image S sketched in Fig. 5(a). For Pass 1 encoding, nine 3×3 subimages $s_0 \sim s_8$ are encoded by slightly modifying the non-hierarchical formats of Rule 1~3 given in Sec. 3.2.2. Note that there is no category bit "c"; Rule 1 is still with $1 \le b \le 11$; but Rule 2 is with $12 \le b \le 127$; and Rule 3 (which handles the case $128 \le b \le 256$) now uses the artificial format

7 bits	8 bits	72 bits
$b=128$	$m=$ an arbitrary number	the original nine gray values : $g_0, g_1, \cdots, g_8$



(1) A 9-by-9 image $S = \bigcup_{i=0}^{8} s_i$ (2) The base-subimage of S (3) The minimum-subimage of S

Fig. 5. A two-pass BS system. (a) A 9×9 image S consists of 9 subimages s_i ($i = 0,1,\cdots,8$), and each s_i is 3×3 . (b) The base-subimage of S where b_{s_i} means the base-value of the subimage s_i. (c) The minimum-subimage of S where m_{s_i} means the minimum-value of the subimage s_i.

After that, each subimage drops the first 7+8=15 bits from its storage format by sending these 15 bits (a base-value b (7 bits) and a minimum-value m (8 bits)) to Pass 2 encoder. The base-values of each nine adjacent subimages constitute a "super" image (see Fig. 5(b)), and the minimum-values of each nine adjacent subimages also constitute a super image (see Fig. 5(c)). For Pass 2 encoding, these super images are encoded respectively using the original Rules 1~3 stated in Sec. 3.2.2.

For decoding, we first decode Pass 2, and recover the base-subimage (Fig. 5(b)) and the minimum-subimage (Fig. 5(c)). Then we decode Pass 1 according to these base-values and minimum-values. For example, the 3×3 subimage S_0 in Fig. 5(a) is reconstructed with the help of the b_{s_0} and m_{s_0} just obtained. The original image S (Fig. 5(a)) is thus recovered.

4. Experimental results and complexity analysis

Although the techniques introduced in Section 3 are explained in terms of gray-level images, we can of course use these techniques to handle color images by applying the techniques three times to each of the three color components.

In this section, we use six color images (shown in Fig. 6) to test the proposed Base Switching (BS) algorithm. In order to compare our results with the results of JBIG and Lossless JPEG which was reported in reference (12), we used the same color components that was used in reference (12), i.e., we used the "YUV" components of the color images. All of the compression ratios presented below express the averages of the corresponding results of the six images, and each of them is again the average compression ratio of the three color components (therefore, we took the average of $6 \times 3 = 18$ data sets to obtain a compression ratio). In these experiments, we used 3-pass BS algorithm (see Section 3.4) to compress each color image component, and the subimage sizes for each pass were 3×3. Table 1 shows how the color image compression ratio varied for the BS, JBIG, and Lossless JPEG algorithms. It was found that our new algorithm, BS, can compete with the international standard algorithms, JBIG and Lossless JPEG. Their compression ratios are very close.

Table 1. The average compression ratio of the six test images.

Methods	BS	JBIG	JPEG
Average compression ratio	2.00	1.99	2.04

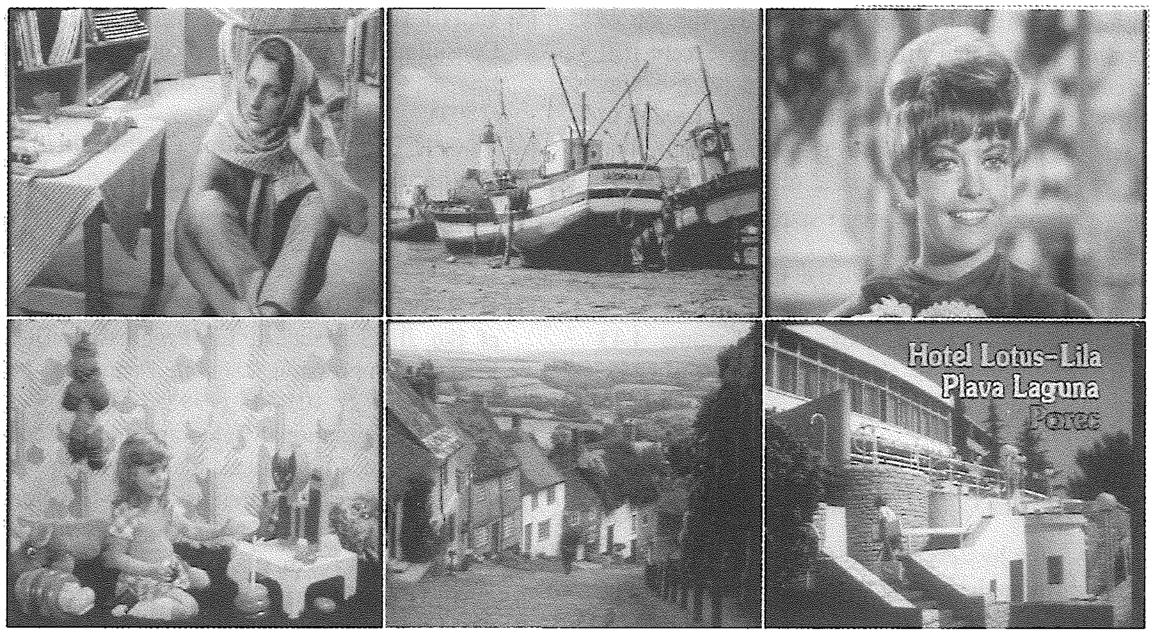


Fig. 6. The test image set (actual size at 720×576 pixels/image).

In Table 1, we found that the average compression ratio of the BS algorithm is a little better than that of the JBIG and a little inferior to that of the Lossless JPEG. On the other hand, although the average compression ratio of the proposed method is a little ($(2.04\text{-}2.00)/2.00 \approx 2.0\%$) inferior to that of the Lossless JPEG, the proposed method is about ($5\text{-}3.94)/3.94 \approx 27\%$ faster than the Lossless JPEG. In the encoding, for example, the BS algorithm requires about 3.33~4.55 clock cycles (the average is 3.94 clock cycles) for each pixel (we will analyze the detail in next paragraph); whereas the Lossless JPEG requires 4~6 clock cycles (the average is 5 clock cycles) for each pixel. On the average, the BS algorithm is therefore a little faster than the Lossless JPEG. The reason that that the Lossless JPEG requires 4~6 clock cycles for each pixel is explained as follows: first, for each pixel, the Lossless JPEG requires 3~5 clock cycles for the predictor part (used for some arithmetic operations such as addition, subtraction, arithmetic-right-shift, and one's complement operation; the detail is given in Sec. 2.10.3 of reference (16) and H.1.2.1 of Appendix A of reference (11)); after that, 1 complete clock cycle for the adaptive arithmetic coder part is needed (see Sec. 13.7 of reference (11)).

We discuss below in detail the time complexity of the BS algorithm. Without the loss of generality, we only analyze the single-pass system (or the first pass of the hierarchical system). To encode a 3×3 subimage, we need 8~15 comparisons (8 comparisons for the best case and 15 comparisons for the worst case) to obtain $\min_{0\le i\le 8} g_i$ and $\max_{0\le i\le 8} g_i$; 1 subtraction and 1 addition to compute the value of $\max_{0\le i\le 8} g_i - \min_{0\le i\le 8} g_i + 1$ (= b, see Equations (1) and (2)); and 8 subtractions to obtain $A'_{3\times 3}$ (because we had known the location of $\min_{0\le i\le 8} g_i$ in the process of finding $\min_{0\le i\le 8} g_i$ and $\max_{0\le i\le 8} g_i$, we could save 1 subtraction, see Equation (4)). After that, if Rule 1 (Rule 2) is applied, then we need 8 (6) additions and 8 (6) multiplications to compute Equation (7). Since an arithmetic operation such as addition, subtraction, comparison, shift, one's complement, and multiplication could be accomplished during one complete cycle under the modern technology of VLSI (see Sec. 2.2 of reference (17)), the BS algorithm requires 30~41 clock cycles to encode a 3×3 subimage. In other words, it takes 3.33~4.55 clock cycles to encode a pixel (the average is 3.94 clock cycles). On the other hand, because the encoded length of the JBIG, the Lossless JPEG, and our proposed algorithm are all variable instead of being fixed, we do not consider the computations of the transformation from decimal values to binary values, because this kind of computations are common for all three methods. Finally, the job of decoding is similar to that of encoding, except that the computation of Equations (1) and (2) are now disappeared. As for the computation loads needed in Passes 2 and 3, they are relatively negligible, because the whole image size of Pass 2 is only $\frac{1}{3 \times 3} = \frac{1}{9}$ of the whole image size of Pass 1; not to mention the even smaller image in Pass 3. (If we consider the work needed in Passes 2 and 3, the 3.94 clock cycles mentioned above will become 4.43 clock cycles, which is about $(5\text{-}4.43)/4.43 \approx 13\%$ faster than the Lossless JPEG.)

5. Concluding remarks

A new lossless compression algorithm in the spatial domain has been proposed along with the experimental results and time complexity analysis. The compression ratios using the proposed BS algorithm were found to be competitive to those of the international standard algorithms JBIG and Lossless JPEG. The math theory needed to derive the proposed encoding format is also provided.

In our experiments, we also tested some other subimage sizes such as 4×4, 6×6, and 8×8, and found that the subimages of size 3×3 can usually achieve higher compression ratios. 3×3 is therefore suggested. The reason why larger sizes did not give better compression ratios is that: as the subimage size increases, the base value b (which indicates how wide the gray value variation of a subimage is) also increases, and the compression ratio is down because the frequency that Rule 3 occurs will increase.

References

[1] J.M. Shapiro, An Embedded Hierarchical Image Coder Using Zero Trees of Wavelet Coefficients, *Proceeding of the IEEE Data Compression Conference*, 214-223 (1993).

[2] S.J. Lee, K.H. Yang, C.W. Kim, and C.W. Lee, Efficient Lossless Coding Scheme for Vector Quantization Using Dynamic Index Mapping, *Electronics Lett.* 31(17), 1426-1427 (1995).

[3] CCITT (International Telegraph and Telephone Consultative Committee), Standardization of Group 3 Facsimile Apparatus for Document Transmission, Recommendation T.4 (1980).

[4] CCITT (International Telegraph and Telephone Consultative Committee), Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, Recommendation T.6 (1984).

[5] CCITT (International Telegraph and Telephone Consultative Committee), Progressive Bi-level Image Compression, Recommendation T.82 (1993).

[6] ISO/IEC (International Organization for Standards/International Electrotechnical Organization), Progressive Bi-level Image Compression, International Standard 11544 (1993).

[7] H. Hampel, R.B. Arps, et al., Technical Features of the JBIG Standard for Progressive Bi-level Image Compression, *Signal Process.: Image Commun.* 4(2), 103-111 (1992).

[8] CCITT (International Telegraph and Telephone Consultative Committee), Digital Compression and Coding of Continuous-tone Still Images, Recommendation T.81 (1992).

[9] ISO/IEC(International Organization for Standards/International Electrotechnical Organization), Digital Compression and Coding of Continuous-tone Still Images, International Standard 10918-1 (1993).

[10] G. Wallace, Overview of the JPEG (ISO/CCITT) Still Image Compression Standard, *Proc. SPIE (Image Processing Algorithms and Techniques)*, 1244, 220-233 (1990).

[11] W.B. Pennebaker and J.L. Mitchell, *JPEG Still Image Data Compression Standard*, New York: Van Nostrand Reinhold (1993).

[12] R.B. Arps and T.K. Truong, Comparison of International Standards for Lossless Still Image Compression, *Proceedings of the IEEE*, 82(6), 889-899 (1994).

[13] A.G. Tescher, *Transform Image Coding in Image Transmission Technique*, W.K. Pratt., ED. New York: Academic Press, Ch. 4 (1979).

[14] H.G. Musmann, *Predictive Image Coding in Image Transmission Techniques*, W.K. Pratt, ED. New York: Academic Press (1979).

[15] G.G. Langdon, Sunset: A Hardware-oriented Algorithm for Lossless Compression of Grey-scale Images, *Proc. SPIE (Medical Imaging V: Image Capture, Formatting, and Display)*, 1444, 272-282 (1991).

[16] V. Bhaskaran and K. Konstantininides, *Image and Video Compression Standards: algorithms and Architectures*. Boston/Dordrecht/London: Bluwer Academic Publishers (1995).

[17] J.L. Hennessy and D.A. Patterson, *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers, INC. San Mateo, California, third printing (1993).