4-1-2006

Lossless Compression of Color Palette Images with One-Dimensional Techniques

Ziya Arnavut
SUNY Fredonia

Ferat Sahin
Rochester Institute of Technology

Follow this and additional works at: <http://scholarworks.rit.edu/article>

Recommended Citation

Ziya Arnavut, Ferat Sahin, "Lossless compression of color palette images with one-dimensional techniques," *Journal of Electronic Imaging* 15(2), 023014 (1 April 2006). <https://doi.org/10.1117/1.2194517>

This Article is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Articles by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

Copyright 2006 Society of Photo-Optical Instrumentation Engineers.

This paper was published in The Journal of Electronic Imaging and is made available as an electronic reprint (preprint) with permission of SPIE. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper are prohibited.

Lossless compression of color palette images with one-dimensional techniques

Ziya Arnavut

SUNY Fredonia

Department of Computer Science

Fredonia, New York, 14063

E-mail: ziya.arnavut@fredonia.edu

Ferat Sahin

Rochester Institute of Technology

Department of Electrical Engineering

Rochester, New York, 14623

Abstract. *Palette images are widely used on the World Wide Web (WWW) and in game-cartridge applications. Many images used on the WWW are stored and transmitted after they are compressed losslessly with the standard graphics interchange format (GIF), or portable network graphics (PNG). Well-known 2-D compression schemes, such as JPEG-LS and JPEG-2000, fail to yield better compression than GIF or PNG due to the fact that the pixel values represent indices that point to color values in a look-up table. To improve the compression performance of JPEG-LS and JPEG-2000 techniques, several researchers have proposed various reindexing algorithms. We investigate various compression techniques for color palette images. We propose a new technique comprised of a traveling salesman problem (TSP)-based reindexing scheme, Burrows-Wheeler transformation, and inversion ranks. We show that the proposed technique yields better compression gain on average than all the other 1-D compressors and the reindexing schemes that utilize JPEG-LS or JPEG-2000. © 2006 SPIE and IS&T.*

[DOI: 10.1117/1.2194517]

1 Introduction

Image compression usually operates on one or more intensity planes of digitized images. For computer graphics images, frequently each color is mapped to an index from a look-up table and the indices of the pixels are then compressed, usually losslessly. The size of the look-up table is much smaller than that of the color intensity space. Hence, this “color palette” approach can generally achieve a better compression ratio for images with limited color.

Highly compressed palletized images are used in many applications. For example, palletized images are widely used on the World Wide Web (WWW). Many images used on the WWW are stored and transmitted as graphics interchange format (GIF) files, which use Lempel-Ziv compression. Lempel-Ziv compression treats the data as a 1-D sequence of index values. The portable network graphics (PNG) format was designed to replace the GIF format. PNG is an improvement over GIF and, fortunately, unlike

GIF, it is also patent-free. An image stored as a lossless PNG file can be 5 to 25% smaller than the GIF file of the same image.

Since palette images are widely used in WWW applications, many techniques have been investigated to improve compression performance of palette images.¹ Several researchers^{1–6} have investigated reindexing color maps to improve the compression gain of predictive techniques, such as JPEG-LS and JPEG-2000. Memon and Venkateswaran³ treated the problem as a general smoothness maximization problem. Zeng *et al.* suggested an index-difference-based reindexing method.⁴ Batatiato *et al.*⁶ and Spira and Malah⁵ suggested color reindexing methods based on the approximate solution to a traveling salesman problem (TSP). Pinho and Neves have given an improved version of Zeng’s algorithm.² Other researchers have also investigated color-palette-based reordering techniques.¹ According to the survey paper of Pinho and Neves,¹ Memon’s and Venkateswaran’s reindexing scheme yields the best compression gain when JPEG-LS or JPEG-2000 is used.

Ausbeck⁷ introduced the piecewise-constant (PWC) image model, which is a two-pass object-based model. Boundaries between constant color pieces and domain colors are determined in the first and second passes, respectively. The PWC image model was shown to yield the best compression on palette images that have more than 32 colors.⁸ Recently, Forchhammer and Salinas⁸ developed a 2-D version of prediction by partial matching (PPM). The coder developed by Forchhammer and Salinas yields better compression gains on images that have less than 32 colors; of course, it is slower than PWC since the technique is based on PPM.

The block-sorting compression (also known as BW94, Burrows-Wheeler compression) technique introduced by Burrows and Wheeler⁹ had a great impact in the data compression area. Since the name Burrows-Wheeler compression (BWC) is more widely used in the literature, we use it here. Compression gains of BWC on text or image data are better than with Ziv-Lempel techniques, with comparable

Paper 05088R received May 17, 2005; revised manuscript received Aug. 30, 2005; accepted for publication Oct. 31, 2005; published online Apr. 18, 2006.

1017-9909/2006/15(2)/023014/11/\$22.00 © 2006 SPIE and IS&T.

Fig. 1 Burrows-Wheeler coder.

speed,^{9–11} while its compression performance is close to that of context-based methods, such as PPM. The scheme of the Burrows-Wheeler coder is presented in Fig. 1. The first step performs the alphabetical (lexical) sorting transformation, which is widely called the Burrows-Wheeler transformation (BWT). The second step is the move-to-front (MTF) coding¹² (transformation). The MTF coding originally was introduced by Bentley *et al.*¹² and was independently discovered by Elias¹³ (called recency ranking by Elias). The last stage utilizes a statistical coder, such as the Huffman or arithmetic coder.

An MTF coder starts with an identity permutation of the size of the alphabet used in the underlying data source. For example, when an MTF coder is implemented for character-based data, the identity permutation is constructed from the set of $\{0, \dots, 255\}$. Whenever a new character or symbol is received from the underlying data string, the coder outputs the index of the symbol. In addition, if the symbol is not at the front of the list, the coder adjusts the permutation (list) by simply moving the symbol to the front of the existing permutation. In essence, a new permutation is generated for each symbol in the data string.

The first block-sorting-based 1-D scheme for compression of color-mapped images was investigated in Ref. 14, where the author introduced the linear order transformation (LOT). Later, in Ref. 15, it was shown that while the MTF coder may be required to obtain better compression gains when the underlying data stream is text, it may not be necessary for some image data. In particular, the authors showed that by transforming image data with the BWT and utilizing a hierarchical arithmetic coder,¹⁶ about 8% more compression can be attained over the well-known block-sorting coder bzip2. Most recently, our studies have shown that when inversion ranks¹⁷ are used after a block-sorting transformation, such as BWT, followed by a structured arithmetic coder, compression gains improve 14% with respect to bzip2. However, as reported in Ref. 14, the LOT's forward transformation is much faster than the BWT's, while the reverse transformations of both techniques require almost the same amount of time. Furthermore, LOT requires less memory than BWT. When the data string elements are 8 bit, such as ASCII characters, LOT requires 256 additional integer indices (pointers). Yet, for the same ASCII data, BWT requires $4n$ integer indices (pointers), where n is the size of the data string.

This paper is organized as follows. In Sec. 2, we expose the reader to the BWT and the LOT. Section 3 briefly explains inversion ranks. Section 4 describes a TSP-based reindexing algorithm. In Sec. 5, we present our experimental results and compare them to the well-known techniques such as bzip2, GIF, PNG, and other compression techniques that utilize reindexing schemes for reordering palette colors. Section 6 discusses the effect of transformations for image data. Finally, in Sec. 7, we conclude our discussion.

2 Block-Sorting Transformations

Let $\mathcal{A}$ be an ordered alphabet of size m , and let $s = [s_1, s_2, \dots, s_n]$ be a string of size n from the alphabet $\mathcal{A}$. Starting with the original string s we construct the matrix,

$$\mathbf{S} = \begin{pmatrix} s_1 & s_2 & \cdots & s_{n-1} & s_n \\ s_2 & s_3 & \cdots & s_n & s_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{n-1} & s_n & \cdots & s_{n-3} & s_{n-2} \\ s_n & s_1 & \cdots & s_{n-2} & s_{n-1} \end{pmatrix},$$

where the successive rows of $\mathbf{S}$ are consecutive cyclic left-shifts of the sequence $s = [s_1, s_2, \dots, s_n]$. A different matrix $\mathbf{H}$ is formed by sorting the rows of $\mathbf{S}$ lexically,

$$\mathbf{H} = \begin{pmatrix} h_1^1 & h_2^1 & \cdots & h_{n-1}^1 & h_n^1 \\ h_1^2 & h_2^2 & \cdots & h_{n-1}^2 & h_n^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ h_1^{n-1} & h_2^{n-1} & \cdots & h_{n-1}^{n-1} & h_n^{n-1} \\ h_1^n & h_2^n & \cdots & h_{n-1}^n & h_n^n \end{pmatrix}.$$

Let $\triangleleft$ indicate the lexicographic order relationship between any two strings. Then, the rows of $\mathbf{H}$ are in the form of

$$H_1 \triangleleft H_2 \triangleleft \cdots \triangleleft H_n,$$

where H_i denotes the i 'th row of $\mathbf{H}$, $1 \leq i \leq n$. Let $H_{i,k}$ be the k 'th symbol in H_i , $1 \leq k \leq n$. We define column k of matrix $\mathbf{H}$ by the array

$$C_k[1, \dots, n] = [H_{1,k} H_{2,k} \cdots H_{n,k}],$$

and write C_k for $C_k[1, \dots, n]$. Let I ($1 \leq I \leq n$) be the index pointing the row I in $\mathbf{H}$ (H_I), which corresponds to the original string s . When the forward BWT is applied to a string s it yields a pair, namely the index I and the last column C_n of $\mathbf{H}$. The reverse of BWT, given I and C_n , retrieves the original string $H_I = s$. For example, string $s = [c a c a b]$, after constructing the matrix $\mathbf{S}$ and by lexically sorting its rows we obtain the following matrix

$$\mathbf{H} = \begin{pmatrix} a & b & c & a & c \\ a & c & a & b & c \\ b & c & a & c & a \\ c & a & b & c & a \\ c & a & c & a & b \end{pmatrix}.$$

The original string appears in row 5. Hence, the sender sends the index 5, which corresponds to the row index of the original string s in $\mathbf{H}$, as well as the last column ($C_5 = [c c a a b]$) to the receiver. The receiver, on obtaining C_5 , can construct C_1 by sorting the elements of C_5 . Once C_1 and C_5 are both constructed, the receiver obtains the permutation $\pi = [3 \ 4 \ 5 \ 1 \ 2]$, which maps C_5 to C_1 . Applying π to each C_i ($1 \leq i \leq 3$) yields the successive columns of the matrix $\mathbf{H}$, that is C_2 , C_3 , and C_4 . Since the receiver knows the original index of s in $\mathbf{H}$, the receiver can recover the string s .

The block-sorting transformation just described is called BWT. The group-theoretical¹⁸ and the combinatorial properties^{19,20} of the BWT have been studied extensively. Among others, an efficient implementation of the BWT is given in Refs. 9, 11, and 21–23.

A simpler block transformation, LOT, is first introduced in Ref. 14. Later in Ref. 19, it is delineated to the BWT and shown that the data transformed by either transformation is in the inverse set of each other. Here, we briefly describe the LOT transformation.

The linearly ordered matrix $\bar{\mathbf{S}}$ of a string $\mathbf{s}$ is formed as the follows:

1. Starting with $\mathbf{s}=[s_1s_2\dots s_n]$, form an $n \times n$ matrix $\mathbf{S}$ by cyclically left-shifting the successive rows.
2. Sort the rows of $\mathbf{S}$ in ascending order with respect to the first element of each row in $\mathbf{S}$. The rows of $\bar{\mathbf{S}}$ are formed by scanning $\mathbf{s}$ from left to right. For each $v \in \mathcal{A}$, determine the positions $j_1, j_2, \dots, j_{f_v}$, where $v=s_i$ for some i , $1 \leq i \leq n$, and f_v is the number (frequency) of v 's in $\mathbf{s}$.

For each occurrence of v , cyclically left shift the string $\mathbf{s}$ (j_k-1) positions ($1 \leq k \leq f_v$). Let $\bar{C}_k$ represent the column k in $\bar{\mathbf{S}}$ and $\bar{C}_{i,k}$ represent the i 'th element from top to bottom in column $\bar{C}_k$. The resulting linearly ordered matrix $\bar{\mathbf{S}}$ consists of the sorted values of $\mathbf{s}$ in its first column, $\bar{C}_1$. An obvious observation about the linearly ordered matrix $\bar{\mathbf{S}}$ is the following: For any given two pairs from the first two columns of $\bar{\mathbf{S}}$, $(\bar{C}_{i,1}, \bar{C}_{i,2})$ and $(\bar{C}_{k,1}, \bar{C}_{k,2})$, with the condition $\bar{C}_{i,1}=\bar{C}_{k,2}$, then the pair $(\bar{C}_{i,1}, \bar{C}_{i,2})$ appears earlier than the pair $(\bar{C}_{k,1}, \bar{C}_{k,2})$ in $\bar{\mathbf{S}}$ (scanning from top to bottom) if and only if the pair $(\bar{C}_{i,1}, \bar{C}_{i,2})$ appears earlier in $\mathbf{s}$ (scanning from left to right). Obviously, the linear ordering induces a particular order on the pairs of the elements. We can recover $\mathbf{s}$ if we know $\bar{C}_2$ and the row index of $\mathbf{s}$ in $\bar{\mathbf{S}}$. Given $\bar{C}_2$ and the row index of $\mathbf{s}$ in $\bar{C}_2$, one can construct $\mathbf{s}$ by obtaining the frequencies of the elements in $\bar{C}_2$ using the count sort²⁴ in one pass. For example, let $\mathbf{s}=[c \ a \ c \ a \ b]$ and suppose $\mathbf{s}$ occurs at position 5 in $\bar{\mathbf{S}}$, and the second column is $\bar{C}_2=[c \ b \ c \ a \ a]$. Then $\bar{C}_1=[a \ a \ b \ c \ c]$; hence the matrix of the first two columns $\bar{C}_1$ and $\bar{C}_2$ of $\bar{\mathbf{S}}$ is

$$(\bar{C}_1, \bar{C}_2) = \begin{pmatrix} a & a & b & c & c \\ c & b & c & a & a \end{pmatrix}^T,$$

where T denotes the transpose of the matrix. Accessing the fifth position of the preceding matrix, $(\bar{C}_1, \bar{C}_2)$, we acquire the first two elements of $\mathbf{s}$, $(\bar{C}_{5,1}, \bar{C}_{5,2})=[c \ a]$. By marking the fifth entry we eliminate it from further consideration [from the preceding matrix $(\bar{C}_1, \bar{C}_2)$]. To find the rest of the elements of $\bar{C}_2$, we determine what follows $\bar{C}_{5,2}=a$ in $\mathbf{s}$. Thus, we scan $\bar{C}_1$ from top to bottom and determine the first unused (unmarked) entry that has a value a . This is the third element of $\mathbf{s}$. In our example, this is the first entry

where $\bar{C}_{1,1}=\bar{C}_{5,2}=a$. Hence, $\bar{C}_{1,2}=c$ should follow $\bar{C}_{5,2}=a$ in $\bar{C}_2$. We then eliminate consideration of the first entry from the matrix $(\bar{C}_1, \bar{C}_2)$. Since $\bar{C}_{1,2}=c$ is determined, the process is repeated to get the fourth element of $\mathbf{s}$. Again, we scan $\bar{C}_1$ to determine the position of the first unused entry that contains $\bar{C}_{1,2}=c$. By finding the first unused entry that contains the value c at position four in $\bar{C}_1$, we discover that the fourth element of $\mathbf{s}$ is $\bar{C}_{4,2}=a$. This entry is eliminated from further consideration. Finally, to find the fifth element of $\mathbf{s}$, we scan to find the first unused entry that has value a in $\bar{C}_1$. Since the first entry that has a value a is used previously, the second entry is considered, $\bar{C}_{2,1}=a$. Therefore, the last element of $\mathbf{s}$ is $\bar{C}_{2,1}=b$, and $\mathbf{s}=[c \ b \ c \ a \ a]$.

The LOT just described has context length one, while the BWT has¹⁰ context length n . One can generalize the block-sorting transformations to have context length m , where $2 \leq m \leq n$. Schindler²¹ used this fact in the implementation of his **szip** compression program, where he uses context level 12. Schindler's work was explained more thoroughly by Yokoo.²⁰ For the discussion of different contextual block-sorting algorithms, interested readers can refer to Refs. 20 and 21.

3 Inversion Ranks

All the variants of the block sorting compressor (e.g., bzip2, szip) utilize the MTF coder on the data stream, transformed with Burrows-Wheeler, before actually sending the data stream to an encoder such as the arithmetic or Huffman encoder.

Recently, we have shown that better compression than GIF, PNG, and bzip2 can be obtained when inversion ranks are used in the second step of a block-sorting based coder.¹⁷ To keep this paper concise, we only briefly describe inversion ranks. For further information, interested readers can consult Refs. 19, 25, and 26.

Definition 3.1. Let $\mathbf{M}$, be a multiset permutation of elements from an underlying set $\mathbf{S}=\{1, 2, \dots, k\}$. Let $\odot$ denote concatenation of data strings. We define the inversion rank vector $\mathbf{D}=\mathbf{D}_k$ for $\mathbf{M}$ as follows:

1. $\mathbf{D}_0=\langle \rangle$.
2. For $1 \leq i \leq k$, $\mathbf{D}_i=\mathbf{D}_{i-1} \odot \mathbf{T}_i$ with $\mathbf{T}_i=\langle x_1, x_2, \dots, x_{f_i} \rangle$, where
 - i. x_1 is the position of the first occurrence of i in $\mathbf{M}$.
 - ii. And for $j>1$, x_j is the number of elements y in $\mathbf{M}$, $y>i$ occurring between the $(j-1)$ 'st and j 'th occurrence of i in $\mathbf{M}$.

For example, for the multiset permutation $\mathbf{M}=[1, 1, 2, 3, 1, 2, 4, 3, 4, 2, 4]$, we have $\mathbf{S}=(1, 2, 3, 4)$. Initially $\mathbf{D}_0=\langle \rangle$. For $i=1 \in \mathbf{S}$, we have $\mathbf{D}_1=\mathbf{D}_0 \odot \mathbf{T}_1$, where $\mathbf{T}_1=\langle 1, 0, 2 \rangle$, so $\mathbf{D}_1=\langle 1, 0, 2 \rangle$. For $i=2 \in \mathbf{S}$, $\mathbf{D}_2=\mathbf{D}_1 \odot \mathbf{T}_2$, where $\mathbf{T}_2=\langle 3, 1, 3 \rangle$. Therefore, $\mathbf{D}_2=\langle 1, 0, 2, 3, 1, 3 \rangle$. For $i=3 \in \mathbf{S}$, $\mathbf{D}_3=\mathbf{D}_2 \odot \mathbf{T}_3$, where $\mathbf{T}_3=\langle 4, 1 \rangle$, so $\mathbf{D}_3=\langle 1, 0, 2, 3, 1, 3, 4, 1 \rangle$. For $i=4 \in \mathbf{S}$, $\mathbf{D}_4=\mathbf{D}_3 \odot \mathbf{T}_4$, where $\mathbf{T}_4=\langle 7, 0, 0 \rangle$. Hence, $\mathbf{D}=\mathbf{D}_4=\langle 1, 0, 2, 3, 1, 3, 4, 1, 7, 0, 0 \rangle$.

To recover the original multiset permutation $\mathbf{M}$ from $\mathbf{D}$,

we need the knowledge of the multiset described by the frequency vector $F=(f_1, f_2, \dots, f_k)$. We initially let $M=[-, -, \dots, -]$ where $|M|=|D|=\sum_{i=1}^k f_i$. From the definition of inversion ranks, D is built recursively as $D_i=D_{i-1} \odot T_i$, where $T_i=\langle x_1, x_2, \dots, x_{f_i} \rangle$ and x_1 represents the position of the first occurrence of i in M and each $x_j, j>1$, represents the number of elements greater than i , which occur between the $(j-1)$ 'st and j 'th occurrences of i in M . Hence, we can recover the elements of M by first inserting i in location $M(x_1)$ and for $j=2, \dots, f_i$ inserting i in the (x_j+1) 'st empty (dash) position in M from the last inserted i .

For example, for the preceding multiset permutation M , $F=\langle 3, 3, 2, 3 \rangle$. The receiver, on receiving vectors S , F and D , can reconstruct the multiset permutation M as follows. The receiver first creates a vector M of size $\sum_{i=1}^4 f_i=11$. From the ordered F , the receiver determines S and that the first element of M is 1 and there are three 1s in the multiset M . The receiver then knows that, the first three entries in D are the locations related to 1 and in the first pass, the receiver inserts 1s in their location in M correctly. Since the first entry in D is 1, it follows that the location of the first element in D is at position 1, hence $M=[1, -, -, -, -, -, -, -, -, -]$. The second entry in D is 0. This means that, there is no element that is greater than 1, between the first and second occurrences of 1. Hence, the receiver inserts the second 1 in the first blank position next to the first 1, so $M=[1, 1, -, -, -, -, -, -, -, -]$. The third entry in D is a 2. This means that, there are two elements greater than 1, between the second and third 1s. Hence, the third 1 should be placed in the third empty position after the second 1. Therefore, $M=[1, 1, -, -, 1, -, -, -, -, -]$. Again, from S and F , the receiver knows that there are three 2s in M . Accessing the fourth position in D it learns the location of the first 2 in M , should occur in position 3. Hence, $M=[1, 1, 2, -, -, -, -, -, -, -]$.

The receiver then proceeds to insert the second 2 into M . From D , the receiver determines that between the first 2 and second 2, there is one element that is greater than 2. So, starting from the location of the first 2 in M , the receiver skips one blank and inserts the second 2 into the second blank position. Similarly, for the third 2 the receiver determines from D that between the second and third 2s, there are three elements that are greater than 2. Therefore, the receiver inserts the third 2 into the fourth blank position, after the second 2. Hence, $M=[1, 1, 2, -, 1, 2, -, -, -, 2, -]$. Continuing the above procedure, the receiver can fully reconstruct M .

4 Ordering of Color Indices

There are several reindexing techniques in the literature¹ that aim to improve the compression performance of predictive techniques, such as JPEG-LS and JPEG-2000. Some of these reindexing methods utilize approaches based on the TSP to reduce prediction errors.

A recent example of such a technique is given by Battiato *et al.*⁶ They propose an approximate solution to TSP and showed that their technique is faster than other reindexing techniques. They built a TSP model to minimize the sum of the differences between the values of neighboring pixels in a given image.

We explore the possible benefit from reindexing the colors in an image before we apply the BWT and later inversion method ($B+INV$). However, we note that the known reindexing techniques are not designed to optimize the compression performance of $B+INV$. Thus, we strongly believe that finding an optimal reindexing method to improve the compression gain of $B+INV$ is an open problem that requires further investigation.

In this paper, we use a TSP-based reindexing algorithm similar to the one presented in Ref. 6. Let us assume that we have an image of size $n \times m$ with N colors (color indices) and by raster scanning the image, the pixel sequence $p_1, p_2, \dots, p_{n \times m}$ is generated. Let $S(c_i, c_j)$ denote the number of times the pair of colors (c_i, c_j) has been seen over the pixel sequence $p_1, p_2, \dots, p_{n \times m}$, and let $W=\{w_{ij}\}$ be an $N \times N$ matrix representing the edge weights between the colors. The edge weights are calculated based on the number of occurrence of the color pairs. For example, the edge weights between color c_i and color c_j is calculated as

$$\begin{cases} w_{ij} = w_{ji} = S(c_i, c_j) + S(c_j, c_i) & \text{if } i \neq j \\ w_{ij} = 0 & \text{if } i = j, \end{cases} \quad (1)$$

where $i, j=1, \dots, N$.

By calculating edge weights between the colors, we generate the weight matrix W :

$$W = \begin{pmatrix} w_{11} & \dots & w_{1N} \\ \dots & \dots & \dots \\ w_{N1} & \dots & w_{NN} \end{pmatrix}. \quad (2)$$

After calculating the weight matrix, we apply the cross-entropy (CE)-based TSP algorithm²⁷ to find a closed path that covers all the colors. The TSP algorithm is set to maximize the sum of the edge weights in the closed path to cover edges with higher weights. According to Ref. 6, a simple Hamiltonian path in a graph of maximum weight gives an optimal reindexing scheme for differential lossless compression techniques. To achieve this maximization we set the CE-based TSP algorithm to maximize the following cost function:

$$J = \sum_{i=1}^N \sum_{j=1}^N S(c_i, c_j) u_{ij}, \quad (3)$$

where $u_{ij}=1$ when c_i and c_j are adjacent colors in the closed path, and $u_{ij}=0$ otherwise.

Maximizing the sum of the edge weights in a closed path yields a smaller entropy since the colors that appear frequently together will be relabeled with new values such that the difference between those colors will be smaller. For example, in a pixel sequence if colors c_i and c_j appear together the most, then the edge weight between them will be the largest. Thus, this edge will be part of the closed path formed by the TSP algorithm. When colors c_i and c_j are relabeled based on the closed path, they will be labeled with consecutive values.

To relabel the color indices, we must determine where the labeling should start. Since we are trying to label the colors that appear together frequently in the sequence close to each other, we must determine the colors that appear

Table 1 Occurrences of color pairs.

c_i	0	1	0	2	0	3	1	2	1	3	2	3
c_j	1	0	2	0	3	0	2	1	3	1	3	2
$S(c_i, c_j)$	3	1	1	2	1	2	3	2	0	0	1	1

together the least number of times and label them far from each other. After we find the closed path that maximizes the sum of the weights, we remove the edge with the minimum weight. Let us assume that the minimum weighted edge is between the colors c_i and c_j . Then, we create a path from c_i to c_j by removing the edge between these colors. Finally, we start relabeling by labeling c_j as zero and continue relabeling until c_i is reached. Since the closed path has the edges with higher weights, after the relabeling, the sum of differences between the relabeled colors will be reduced dramatically.

We give an example to clarify the algorithm just described. Let us consider an image with four colors and the following pixel sequence

33221211123300100220011200333011.

Based on the sequence, the numbers of occurrences of the color pairs are as shown in Table 1, and for the sequence just given, the following weight matrix is calculated using Eq. (1):

$$W = \begin{pmatrix} 0 & 4 & 3 & 3 \\ 4 & 0 & 5 & 0 \\ 3 & 5 & 0 & 2 \\ 3 & 0 & 2 & 0 \end{pmatrix}.$$

Figure 2 shows the colors and the corresponding weights. Since the TSP algorithm is trying to maximize the sum of the weights, the resulting close path is 0 to 1 to 2 to 3 to 0. The sum of the weights for this route is 14. The edge with

Fig. 2 Example for traveling salesman algorithm utilized in this paper.

Fig. 3 "Sunset" image in gray.

minimum weight is the edge between colors 2 and 3. Thus, we create a path from color 3 to color 2 as 3 to 0 to 1 to 2 by removing this edge. Then, we start relabeling from color 2, we obtain the relabeling, as shown in Fig. 2. Thus, color 0 is relabeled with 2, color 1 is re-labeled with 1, color 2 is re-labeled with 0, and finally color 3 is relabeled with 3. After the relabeling the sequence becomes

33001011103322122002211022333211.

Then, the sum of the differences between the relabeled colors becomes 24. The sum of differences between the colors in the original sequence was 26.

5 Experimental Results

This section presents experimental results and provides discussion on the performance of various techniques. All the test images utilized in our work were obtained from the ftp site, <ftp://ftp.ieeta.pt/~ap/images> and previously were used as test images in Ref. 1. Three of the test images are shown in Figs. 3–5. All of the test images are converted from PPM format to GIF, PNG, and BMP (bitmap) formats using the version 1.2.3 of the *Gimp* program. We notice that the BMP formatted, or GIF compressed images generated by the *Gimp* program are slightly different in size than the ones that may be obtained using the Unix utility function "ppm-togif" or some other software, such as "xv". This may be because each program may encode the header information differently. Thus, for some of the test images the bits per pixel (bpp) values for GIF compression scheme reported in Ref. 2 or some other papers may vary slightly from our results presented in this paper.

Table 2 presents the experimental results of different compression techniques on the set of test images. Columns labeled $L+X$ and $B+X$ present the results of the algorithms

Fig. 4 "Gate" image in gray.

Fig. 5 “Winaw” image in gray.

that first transform the image with LOT and BWT followed by the appropriate ranking scheme X , recency (MTF) or inversion (INV) ranks. Once the ranking technique X is applied, the output is encoded with the structured arithmetic coder that was originally introduced by Moffat *et al.*¹⁶ and improved by Fenwick.¹¹ We did not list the results of $B+MTF$ since $B+MTF$ is a part of bzip2’s implementation. In columns labeled $L-X$ and $B-X$, the corresponding algorithms do not use any ranking method in the second step. Hence, we used the symbol $-X$ to indicate that method X is excluded from the process. In each table, the average and weighted average compression results are presented in bits per symbol (bps) or bpp.

As shown in Table 2, the best weighted average compression in bps is obtained by $B+INV$. $B+INV$ achieves compression gains of 37.9% over GIF, 33.3% over PNG, 25.6% over gzip, and 12.5% over bzip2, while it outperforms all other 1-D techniques. The second-best weighted

Table 2 Results of 1-D compression techniques in bps.

Image	Number of colors	GIF	PNG	gzip	bzip2	$L-MTF$	$L+MTF$	$L+INV$	$B-MTF$	$B+INV$
“Clegg”	256	5.678	4.838	4.862	4.240	4.788	4.111	3.974	4.760	3.866
“Cwheel”	256	2.763	2.242	2.253	2.068	3.063	1.827	1.774	2.956	1.722
“Fractal”	256	6.923	5.878	5.903	5.171	5.327	5.013	4.964	5.335	4.893
“Frymire”	256	2.674	2.181	2.190	1.944	2.483	1.865	1.733	2.457	1.664
“Ghouse”	256	4.988	4.320	4.322	4.065	4.375	3.891	3.728	4.407	3.697
“Serrano”	256	2.877	2.428	2.450	2.276	3.031	2.102	1.960	3.051	1.935
“Yahoo”	229	1.989	1.762	1.758	1.760	2.017	1.625	1.552	2.001	1.496
“Sunset”	204	2.602	2.143	2.122	1.929	2.286	1.797	1.658	2.300	1.614
“Decent”	122	2.927	2.647	2.639	2.608	2.708	2.432	2.246	2.730	2.257
“Gate”	84	2.939	2.601	2.577	2.420	2.667	2.317	2.225	2.647	2.154
“Benjerry”	48	1.239	1.144	1.135	1.066	1.124	1.142	0.996	1.059	0.889
“Sea-dusk”	46	0.323	0.102	0.104	0.092	0.328	0.093	0.073	0.317	0.065
“Netscape”	32	2.113	2.005	2.012	1.810	1.848	1.924	1.755	1.823	1.577
“Party8”	12	0.427	0.369	0.363	0.330	0.385	0.394	0.318	0.369	0.283
“Winaw”	10	0.499	0.510	0.510	0.433	0.470	0.454	0.394	0.453	0.372
“Music”	8	1.241	1.061	1.036	1.112	1.231	1.114	1.038	1.191	0.939
“Books”	7	1.516	1.483	1.476	1.414	1.351	1.411	1.289	1.349	1.252
“PC”	6	0.843	0.538	0.538	0.469	0.723	0.823	0.677	0.505	0.417
Average bpp.		2.600	2.125	2.125	1.956	2.233	1.909	1.797	2.206	1.727
Weighted average		2.104	2.035	1.917	1.717	2.095	1.792	1.666	1.997	1.526
Normalized		1.379	1.333	1.256	1.125	1.372	1.174	1.092	1.309	1.000

Fig. 6 Segment of BWTed "Gate" image.

Fig. 7 Segment of LOTed "Gate" image.

average compression result is achieved by $L+INV$. The compression performance of the $L+INV$ technique is approximately 9% worse than $B+INV$. However, as mentioned earlier, $L+INV$ is much faster than $B+INV$ since LOT is a context-level one transformation as opposed to context level n and uses less memory than the BWT transformation.

The third-best weighted average compression result is obtained by *bzip2*. This well-known coder is composed of the BWT and MTF transformations in addition to a modified Huffman coder. The fourth-best weighted average compression is achieved by $L+MTF$. Note that all the transformation-based techniques yield better compression results, in terms of weighted average bit per symbol, than GIF and PNG, with the exception of the $L-MTF$ technique. Apparently, $L-MTF$ and the $B-MTF$ perform comparably. Although $B-MTF$ yields approximately 1.2% better compression than $L-MTF$ in terms of weighted average bps, $L-MTF$ performs better than $B-MTF$ for most of the images that have a high number of colors. As we can see in Figs. 6 and 7, both transformations yield similar output as expected. Although in images, the neighboring pixel values are highly correlated, it is hard to find similar long patterns in different regions of an image. For images with fewer colors this may not be true. Thus, $B-MTF$ performs slightly better than $L-MTF$ on images that have fewer colors.

Unlike the results presented in Table 2, where the results were presented in bps, the results in Table 3 are presented in bpp so that we can have a fair comparison with the results presented in Ref. 1. Table 3 shows the results of reindexing schemes. The results presented under the columns titled Memon, Zeng, mZeng, and Battiato are taken

from Table 1 of the recently published survey paper of Pinho and Neves.¹ In Table 3, under the column titled $T+B+INV$, we present the results of the $B+INV$ technique, which is combined with our TSP-based reindexing scheme. In the proposed column, we present the results of our new technique. Our new technique is simply a combination of $T+B+INV$ and $B+INV$. When the number of colors in an image is less than 32, the TSP-based reindexing scheme helps $B+INV$ to yield better compression gains. Our algorithm uses the $B+INV$ technique for images that have more than 31 colors and for images that have less than 32 colors it uses the $T+B+INV$ technique. This improves our compression gain close to 1% for the 18 test images used in Table 3.

From the compression results shown in Table 3, it is clear that our proposed technique yields the best compression gain in terms of weighted average bpp. Our proposed scheme achieves 73.0% better compression gain over Battiato's technique, while it yields 46.5% better compression gain over Memon's technique, which is the best method among the reindexing schemes presented in Ref. 1.

In Table 4, we present the compression results of images with and without dithering for the test sets of "Natural1" and "Natural2." These test images were also obtained from the same ftp site mentioned earlier. They are color quantized. Also, in each set there are those different groups of images, namely, those with 256, 128, and 64 colors, both with and without dithering. In each color group (with and without dithering), there are 23 images in Natural1, while there are 12 images in Natural2. Our proposed technique yields 15.1 to 16.5% better compression gains on average for images without dithering, and 18.0 to 26.0% for images with dithering.

Table 3 Compression results of reindexing techniques and our method in bpp.

Image	Number of colors	Memon	Zheng	mZheng	Battiato	T+B+INV	Proposed
"Clegg"	256	5.22	5.863	5.456	6.075	3.971	3.881
"Cwheel"	256	2.724	3.058	2.878	3.374	1.729	1.726
"Fractal"	256	5.763	6.193	5.828	7.234	4.935	4.913
"Frymire"	256	3.259	3.619	3.376	3.946	1.692	1.668
"Ghouse"	256	4.229	4.841	4.541	5.418	3.715	3.705
"Serrano"	256	3.001	3.393	3.273	3.779	1.963	1.948
"Yahoo"	229	1.743	1.798	1.789	2.008	1.565	1.550
"Sunset"	204	2.407	2.570	2.307	2.647	1.626	1.619
"Decent"	122	2.817	2.943	2.854	3.531	2.290	2.276
"Gate"	84	2.548	2.587	2.566	3.116	2.175	2.167
"Benjerry"	48	1.133	1.154	1.137	1.186	0.909	0.900
"Sea_dusk"	46	0.197	0.189	0.189	0.189	0.069	0.065
"Netscape"	32	1.745	1.791	1.752	1.907	1.592	1.582
"Party8"	12	0.321	0.318	0.318	0.319	0.284	0.284
"Winaw"	10	0.450	0.450	0.450	0.464	0.376	0.376
"Music"	8	1.051	1.060	1.051	1.071	0.961	0.961
"Books"	7	1.453	1.469	1.469	1.453	1.282	1.282
"PC"	6	0.748	0.743	0.745	0.743	0.417	0.417
Average		2.267	2.447	2.332	2.692	1.753	1.740
Weighted average		2.243	2.452	2.333	2.65	1.546	1.532
Normalized		1.465	1.601	1.521	1.730	1.009	1.000

It is clear that our proposed algorithm yields slightly better compression gain over $B+INV$, as expected. The re-indexing algorithm aims to minimize the sum of differences between the values of the neighboring pixels. This minimization does not boost the performance of $B+INV$ because $B+INV$ does not depend on the differences between the values of the neighboring pixels. If a reindexing algorithm is specially modeled for the $B+INV$ technique, the results might be improved. At this time, we are not aware of any preprocessing technique that can help inversion ranks to increase compression gain. This requires further investigation and should be material for future research.

6 Effect of Transformations

It is well known that neighboring pixel values in an image are highly correlated. Most often, in a given gray-colored image, the difference between the values of two neighboring pixels will not exceed ± 10 . Of course, this is not true

when neighboring pixel values are in different regions or edges. In general, if a pixel's neighbor is on an edge, the absolute difference of neighboring pixel values may be quite high (>30).

The forward BWT sorts the lexical sequences in increasing order. When an image is BWT transformed, frequently, pixel values tend to group together "nicely," except for some peaks that may occur due to edges. Figures 6 and 7 display the output of 3000 elements of the BWTed and LOTed gate image, respectively. These 3000 elements are taken from the same exact coordinates of BWTed and LOTed gate image data. Notice the similarities between Figs. 6 and 7. In both figures, most of the values occur between 30 and 45. However, it appears that similar values tend to group together in Fig. 6 better than in Fig. 7.

Contrary to the nature of image data, in text data, frequently, the values between neighboring characters may differ significantly. For any given sentence, we have spaces

Table 4 Compression results of ours and reindexing techniques in bpp.

Image Set		Memon	Zheng	mZheng	Battiato	Proposed
"Natural1"	256	4.203	4.907	4.475	5.530	3.549
	128	4.441	3.844	3.552	4.287	2.954
	64	2.641	2.804	2.709	3.144	2.327
	Weighted average	3.428	3.852	3.579	4.320	2.944
	Normalized	1.165	1.311	1.218	1.470	1.00
"Natural1"	256d	4.870	5.959	5.120	6.232	4.1243
	128d	4.112	4.537	4.225	5.146	3.363
	64d	3.341	3.590	3.420	3.915	2.327
	Weighted average	4.108	4.695	4.255	5.098	3.272
	Normalized	1.260	1.440	1.300	1.560	1.00
"Natural2"	256	4.607	5.491	4.881	5.983	4.023
	128	3.771	4.339	3.979	4.729	3.220
	64	2.793	3.060	2.912	3.281	2.497
	Weighted average	3.724	4.297	3.924	4.664	3.247
	Normalized	1.151	1.328	1.213	1.441	1.00
"Natural2"	256d	5.063	6.006	5.322	5.455	4.345
	128d	4.205	4.766	4.423	5.314	3.557
	64d	3.417	3.712	3.548	4.127	2.842
	Weighted average	4.228	4.828	4.431	4.965	3.582
	Normalized	1.180	1.348	1.237	1.386	1.00

between words. For example, for the word "zone" the difference of the first two ASCII values is 90 because the ASCII values of space (" ") and *z* are 32 and 122, respectively, while the difference between the last two ASCII values is 55 because the ASCII values of *e* and "." are 101 and 46. Figure 8 shows a segment of the BWTed "Book1" (from Calgary Corpus) text data. Apparently, ASCII values of characters that are similar tend to group together nicely. Most of the 3000 BWTed values are in the range of 100 and 122. Notice the difference between BWTed and LOTed "Book1" text data, as shown in Figs. 8 and 9 respectively. From Fig. 8 we can easily observe that the elements of BWTed "Book1" are grouped together more, and most of the values are between 105 and 120. On the contrary, in Fig. 9, we see an oscillation between neighboring data elements and the values are in the range of 30 and 120. Obviously, neighboring values of BWTed text data tend to group together and are in close range to each other. Hence, when an arithmetic coder is utilized after the MTF or inversion ranks, we obtain better compression. However, when the same process is applied to LOT-transformed text data we do not obtain as good compression.

In general, we observed that after the BWT transformation is applied to an image, similar pixel values tend to group together. This grouping is smoother for images that have less than 32 colors. For images that have more than 31 colors, groups may not be as smooth since it is hard to find the same exact pattern in different regions in images that have a high number of colors. Therefore, in image data, dependencies are more on neighboring pixels, and LOT-transformed images, such as the "Gate" image, resemble the BWT transformed data, as can be seen in Figs. 6 and 7. Hence, when MTF or inversion ranks is applied to BWTed or LOTed image data, in terms of weighted average bps, $L+MTF$ and $bizp2$ ($B+MTF$) or $L+INV$ and $B+INV$, differs approximately 5 to 9%, respectively.

It is apparent from the results shown in Table 2 that inversion ranks are more appropriate for image compression than the MTF. In addition, the inversion ranks definitely improves the compression gain of both transformations, namely, the LOT and the BWT.

Fig. 8 Segment of BWted "Book1" text data.

7 Conclusion

All variants of the block sorting coder (e.g., bzip2, szip) first transform a given data stream with the BWT and then

Fig. 9 Segment of LOTed "Book1" text data.

utilize the MTF transformation before coding the resulting data stream with an arithmetic or Huffman coder.

In this paper, we investigated, various 1-D compression techniques for color palette images. We showed that among the 1-D techniques, blocks-sorting-based compressors that utilize inversion ranks in the second step yield better compression than all the other 1-D compressors. Moreover, our proposed technique provides compression gains in terms of weighted average bps over the set of test images on the order of 38.9% over GIF, 33.3% over PNG, and 12.5% over bzip2.

We also showed that for various sets of test images our proposed reindexing-based scheme yields results superior to the best reindexing scheme, namely, that of Memon and Venkateswaran. Our technique achieves 46.5% better compression on average on synthetic images, while it yields 15.1 to 16.5% better compression for various nonsynthetic, natural images, and 18.0 to 26.0% for the corresponding set of dithered images.

Acknowledgments

We would like to thank Dr. Joseph Straight for his valuable comments and suggestions, Dr. Armando J. Pinho for supplying the test images and answering our several questions, Dr. Thomas Taimre for letting us modifying his code to implement our TSP-based reindexing algorithm, Dr. Wenjun Zeng for supplying us another set of images, and to the unknown referees for their valuable suggestions to improve the quality of this work.

References

1. A. J. Pinho and A. J. R. Neves, "Survey on palette reordering methods," *IEEE Trans. Image Process.* **13**(11), 1411–1417 (2004).
2. A. J. Pinho and A. J. R. Neves, "A note on Zeng's technique for color reindexing of palette-based images," *IEEE Signal Process. Lett.* **11**(2), 232–235 (2004).
3. N. D. Memon and A. Venkateswaran, "On ordering color maps for lossless predictive coding," *IEEE Trans. Image Process.* **5**(11), 1522–1527 (1996).
4. W. Zeng, J. Li, and S. Lei, "An efficient color reindexing scheme for palette-based compression," in *Proc. IEEE Int. Conf. on Image Processing (ICIP-2000)*, Vol. 3, pp. 476–479 (2000).
5. A. Spira and D. Malah, "Improved lossless compression of color-mapped images by an approximate solution of the traveling salesman problem," in *Proc. Acoustics, Speech, and Signal Processing (ICASSP '01)*, Vol. 3, pp. 1797–1800 (2001).
6. S. Battiato, G. Gallo, G. Impoco, and F. Stanco, "An efficient reindexing algorithm for color-mapped images," *IEEE Trans. Image Process.* **13**(11), 1419–1423 (2004).
7. P. J. Ausbeck Jr., "The piecewise-constant image model," in *Proc. IEEE on Lossless Data Compression: Special Issue*, pp. 1779–1789 (2000).
8. S. Forchhammer and M. J. Salinas, "Progressive coding of palette images and digital maps," in *Proc. Data Compression Conf., DCC-2002*, pp. 362–371 (2002).
9. M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," SRC Research Report 124 (1994) (ftp site: gatekeeper.dec.com, /pub/DEC/SRC/research-reports/SRC-124.ps.Z).
10. J. G. Cleary, W. J. Teahan, and I. H. Witten, "Unbounded length contexts for PPM," in *Proc. Data Compression Conf., DCC-95*, pp. 52–61 (1995).
11. P. Fenwick, "The Burrows-Wheeler transform for block sorting text compression: principles and improvements," *Comput. J.* **39**(9), 731–740 (1996).
12. J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "A locally adaptive data compression scheme," *Commun. ACM* **29**(29), 320–330 (1986).
13. P. Elias, "Interval and recency Rank source coding: two on-line adaptive variable-length schemes," *IEEE Trans. Inf. Theory* **IT-33**(1), 3–10 (1987).
14. Z. Arnavut, "Lossless compression of pseudo-color images," *Opt. Eng.* **38**(6), 1001–1005 (1999).
15. Z. Arnavut and H. Otu, "Lossless compression of color-mapped images with Burrows-Wheeler transformation," in *Proc. Signal Process-*

- ing Conf., IASTED, pp. 185–189 (2001).
16. A. Moffat, R. Neal, and I. H. Witten, "Arithmetic coding revisited," in *Proc. Data Compression Conf.*, DCC-95, pp. 202–211 (1995).
17. Z. Arnavut and F. Sahin, "Inversion ranks for lossless compression of palette images," in *Proc. IEEE Electro/Information Technology Conf.*, CDROM (2005).
18. Z. Arnavut and S. S. Magliveras, "Lexical permutation sorting algorithm," *Comput. J.* **40**(5), 292–295 (1997).
19. Z. Arnavut and M. Arnavut, "Investigation of block sorting transformations," *J. Comput. Math.* **2**(1), 46–57 (2004).
20. H. Yokoo, "Notes on block-sorting data compression," *Electron. Commun. Jpn.* **82**(6), 18–25 (1999).
21. M. Schindler, "A fast block sorting algorithm for lossless data compression," in *Proc. Data Compression Conf.*, DCC-97, p. 469 (1997).
22. N. J. Larsson, "The context trees of block sorting compression," in *Proc. Data Compression Conf.*, DCC-98, pp. 189–198 (1998).
23. J. Seward, "The Bzip2 program," vers. 0.1pl2, <http://www.muraroa.demon.co.uk> (1997).
24. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York (1986).
25. Z. Arnavut, "Inversion coder," *Comput. J.* **47**(1), 46–57 (2004).
26. R. Sedgewick, "Permutation generation methods: a review," *ACM Comput. Surv.* **9**(2), 137–164 (1977).
27. P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the gross-entropy method," *Ann. Operat. Res.* **134**(1), 19–67 (2005).

Ziya Arnavut received his BSc degree in applied mathematics from Ege University, Turkey, in 1983, his MSc degree from the University of Miami in 1987, and his PhD degrees in computer science from the University of Nebraska, Lincoln, in 1995. During 1996 and 1997, he was a research analyst with the Remote Sensing Lab, at the University of Nebraska, Omaha. Since August 1997, he has been with the State University of New York (SUNY), Fredonia,

where he is an associate professor in the Computer Science Department. His research interests are data compression, algorithms, image processing, and remote sensing. He is a member of the IEEE Computer Society.

Ferat Sahin received his BSc degree in electronics and communications engineering from Istanbul Technical University, Turkey, in 1992 and his MSc and PhD degrees in electrical engineering from Virginia Polytechnic Institute and State University in 1997 and 2000, respectively. His MSc thesis concerned a radial basis function networks solution to a real-time color image classification problem. During his PhD study, he performed research in biological decision-theoretic intelligent agent design and its application to mobile robots in the context of distributed multiagent systems. In September 2000, he became an assistant professor with the Rochester Institute of Technology. His interests are biologically inspired algorithms, decision theory, Bayesian networks, artificial immune systems, microactuators, microelectro mechanical systems (MEMs), autonomous robots, intelligent control, and nonlinear control. He is a member of the IEEE Systems, Man, and Cybernetics Society, Robotics and Automation Society, and the American Association for Engineering Education (ASEE). He was the student activities chair for the IEEE SMC society from 2001 to 2003 and he is currently the secretary.