11-12-1996

Application of inversions to lossless image compression

Ziya Arnavut

Application of inversions to lossless image compression

Ziya Arnavut
University of Nebraska
Geography and Geology Department
Omaha, Nebraska 68182
E-mail: zarnavut@cwis.unomaha.edu

Abstract. Linear prediction schemes, such as that of the Joint Photographic Experts Group (JPEG), are simple and normally produces a residual sequence with lower zero-order entropy. Occasionally the entropy of the prediction error becomes greater than that of the original image. Such situations frequently occur when the image data have discrete gray levels located within certain intervals. To alleviate this problem, various authors have suggested different preprocessing methods. However, the techniques reported require two passes. We extend the definition of Lehmer-type inversions (Lehmer 1960 and 1964) from permutations to multiset permutations and present a one-pass algorithm based on inversions of a multiset permutation. We obtain comparable results when we apply JPEG and even better results when we apply some other linear prediction schemes on a preprocessed image, which is treated as multiset permutation. [S0091-3286(97)02304-0]

Subject terms: permutations; inversions; multiset permutations; data compaction; dynamic range reduction; sparse histogram.

Paper 09046 received Apr. 6, 1996; revised manuscript received July 12, 1996, and Aug. 26, 1996; accepted for publication Nov. 12, 1996.

1 Introduction

The goal of data compression is to find shorter representations for any given data. In a data storage application, this is done to save storage space on an auxiliary device or in the case of a communication scenario, to increase the channel throughput.

Image compression is divided into two main groups, lossy and lossless. Most of the literature in image compression deals with lossy techniques, for which the pixel intensities can not be recovered from an encoded bit stream. Lossless applications, such as digital radiology in the medical field and satellite imaginary in remote sensing applications, can not tolerate such an irreversible process as the images are subject to further processing.

A lossless image compression technique consists of two main components, modeling and encoding.[1] A model captures the structure inherent in the raw data and extracts it. The residual, also called the error, is then encoded using an entropy-encoding technique. Encoding techniques, such as arithmetic and Huffman encoding, are known to perform optimally in terms of the number of bits used to encode a given data source. Hence the critical task in data compression is modeling.

One of the most popular models used in image compression is the linear prediction model. In this model, the current pixel intensity $x_{i,j}$ is predicted using a linear combinations of the neighboring pixel intensities. The residual is then encoded and transmitted. The Joint Photographic Experts Group (JPEG) still picture compression standard[2] uses such predictive schemes in its lossless form. Although linear prediction models, such as JPEG are quite useful, and usually yield a residual sequence with lower zero-order entropy, in some cases, they may not yield significant gain.

Preprocessing an image prior to JPEG or JPEG-like predictive schemes may yield substantial gain. Various authors[3,4] have reported different preprocessing methods. However, their methods require two passes. In this paper, we present a one-pass preprocessing scheme, based on inversions of a multiset, that yields comparable results. We treat an image as a multiset permutation and apply inversions to it. We then apply JPEG-like predictive schemes on preprocessed data. Results indicate that more substantial gains can be obtained by preprocessing and using JPEG-like predictive schemes than when using such schemes alone.

In Sec. 2, we briefly review linear predictive techniques and try to point out their weaknesses. In Sec. 3, we define mathematical concepts, such as permutations, multiset permutations, inversions, and inversions on multiset permutations. We give algorithms that generate inversion vectors of multiset permutations and then methods for recovering a multiset permutation from a corresponding inversion vector. Section 4 discusses our proposed approach. In Sec. 5, we present the results obtained by applying JPEG and JPEG-like predictive schemes on inversion vectors, generated from the image data, which are treated as multiset permutations. In Sec. 5, we give algorithms that generates inversion vectors adaptively. Finally, in Sec. 6, we compare our results with some previous work.

2 Linear Predictive Techniques

For typical images, the values of adjacent pixels are highly correlated; a great deal of information about a pixel value can be obtained by inspecting the neighboring pixel values. Linear predictive techniques try to exploit these correlations between neighboring pixels. They scan the image in a

 0091-3286/97/$10.00

Table 1 Lossless JPEG predictors.

Mode	Prediction for $x_{i,j}$
0	No prediction
1	$x_{i-1,j}$
2	$x_{i,j-1}$
3	$x_{i-1,j-1}$
4	$x_{i,j-1}+x_{i-1,j}-x_{i-1,j-1}$
5	$x_{i,j-1}+(x_{i-1,j}-x_{i-1,j-1})/2$
6	$x_{i-1,j}+(x_{i,j-1}-x_{i-1,j-1})/2$
7	$(x_{i,j-1}+x_{i-1,j})/2$

fixed order (usually raster order) and predict the current pixel by taking a linear combination of neighboring pixels that have been previously transmitted. The JPEG still picture compression standard[2] uses linear predictive techniques in the prediction step. It has eight different predictive schemes, which are listed in Table 1. The first scheme makes no prediction. The next three are 1-D predictors, which scan the image in raster, vertical, and diagonal order. The last four are 2-D predictors.

An adaptive version of JPEG, which was first proposed by Niss,[5] divides a given image into $m\times m$ blocks and selects the best JPEG predictor for each block by determining least absolute sum of prediction errors. Because the decoder needs to know which JPEG predictor is used for each block, an overhead $3/m\times m$ bits/pixel occurs from transmitting the required information. For a block size of 8×8, this cost is $3/64\approx 0.047$ bits/pixel, however the gain is usually much higher than the overhead. Memon and Sayood[6] suggested using $x_{i,j-1}+(x_{i-1,j+1}-x_{i-1,j-1})/2$ to substitute the trivial predictor, JPEG 0. They further called this version of JPEG, BJPEG. We refer to this method as BJPEG hereafter.

Linear prediction schemes, such as JPEG or BJPEG, are simple and normally produce a residual sequence with lower zero-order entropy. Occasionally, the entropy of the prediction error becomes greater than that of the original image. For example, in the case of the green band of the x-ray image [obtained from the University of Southern California (USC) database], the entropy of the image data is 5.21 bits/pixel. The JPEG predictors yield greater entropy than the entropy of original image, except the first prediction scheme, where the entropy is 5.17 bits/pixel. The BJPEG predictive scheme, suffers similarly. The best entropy of the prediction errors of BJPEG yields 5.81 bits/pixel for the x-ray image. Such situations frequently occur when the image data has discrete gray levels located within certain intervals. For example, in the case of the x-ray image, there are 66 different gray values, distributed in the range of 0 to 255. To alleviate this problem, Kuroki et al.[3] suggest a gray-level conversion method and with a different approach Memon and Ray[4] suggest optimal linear ordering.[7] We obtained comparable results when we applied JPEG and even better results when we applied BJPEG on a preprocessed image, which is treated as a multiset permutation.

3 Mathematical Preliminaries

To set the stage for our later discussions, in this section we define the mathematical concepts.

3.1 *Permutations*

By a permutation of a finite set S, we mean a bijection from S onto itself. For example, in functional notation,

$$\boldsymbol{\pi}=\begin{pmatrix}1&2&3&4&5\\2&5&4&3&1\end{pmatrix}$$

is a permutation on $S=\{1,2,3,4,5\}$. Since we can always arrange the elements of S in a particular order, a permutation π is completely described by the bottom row, for example by [2 5 4 3 1]. The representation $\pi=[2\ 5\ 4\ 3\ 1]$ for the above permutation is called the Cartesian form of π.

Given a set S of size n, there are clearly $n!$ permutations on S if elements of the set S are distinct. The idea of permutations on a set S can be extended to multiset if the elements of the set S are not distinct. By a multiset $\mathcal{M}$ based on set S we mean a pair (S,f), where $f:S\rightarrow\mathbb{N}$ is a function from S into $\mathbb{N}$ and f is called the frequency (multiplicity) function. The size of $\mathcal{M}$ is defined by $|\mathcal{M}|=\Sigma_{x\in S}f(x)$, say $|\mathcal{M}|=m$. A multiset permutation[8] M based on multiset $\mathcal{M}=(S,f)$ is a mapping $M:\{1,2,\ldots,m\}\rightarrow S$, such that if $x\in S$,

$$f(x)=|\{j:1\leqslant j\leqslant m,\ M[j]=x\}|.$$

Intuitively, we can think of a multiset permutation $M:\{1,2,\ldots,m\}\rightarrow S$ as a linear array $[M(1), M(2),\ldots,M(m)]$, where $x\in S$ appears as an element in M exactly $f(x)$ times. If the underlying set has an implicit or explicit linear order, say $S=\{a_1,a_2,\ldots,a_n\}$ with $a_1<a_2<\ldots<a_n$, we sometimes denote the multiset by

$$\mathcal{M}=a_1^{f(a_1)}\cdot a_2^{f(a_2)}\ldots a_n^{f(a_n)}.$$

If $\mathcal{M}=a_1^{f_1}\cdot a_2^{f_2}\ldots a_n^{f_n}$ $[f(a_i)=f_i]$, then it is easy to see that the number of distinct multiset permutations of $\mathcal{M}$ is

$$\frac{(f_1+\ldots+f_n)!}{f_1!\ldots f_n!}.$$

This quantity is also called a multinomial coefficient.[8]

Given a 2-D image P, by raster scanning the image P we can convert it to a linear array P'. We can think of P' as a multiset permutation of the multiset of gray values. In the case of a 256-gray-valued image, we can consider P' to be a multiset permutation on multiset $0^{f_0}\cdot 1^{f_1}\ldots 255^{f_{255}}$, where f_i represents the frequency of gray value i in the multiset. There are

$$\frac{(f_0+\ldots+f_{255})!}{f_0!\ldots f_{255}!}$$

possible different multiset permutations and the same number of images.

3.2 *Inversions*

The notion of inversion for a given permutation was introduced quite early[8] in an effort to provide concise representations of ordinary permutations. Several variants and types of inversions were defined at different times by different authors. Sedgewick[9] gives some other inversion generation methods. We are particularly interested in Lehmer's[10,11] method, which we describe here.

Definition 1. Let $\pi=[\pi_1,\pi_2,\ldots,\pi_n]$ be an arbitrary permutation of an n set S of positive integers. The Lehmer inversion vector I_π, associated with π is the sequence $[I_1,I_2,\ldots,I_n]$ of non-negative integers defined as

$$\text{for} \quad 1\leqslant k\leqslant n \quad I_k=|\{j:k<j\leqslant n \quad \text{and} \quad \pi_k>\pi_j\}|.$$

For instance, the permutation [3,1,5,2,4] yields [2,0,2,0,0] as its inversion vector. Lehmer's method simply counts the number of $\pi_i>\pi_j$, for $i<j\leqslant n$, and generates n elements in the inversion vector. We call this version of Lehmer's method right smaller (RS).

Given a Lehmer-inversion vector $I_\pi=[I_1,I_2,\ldots,I_n]$ for a permutation, one can obtain the permutation $\pi=[\pi_1,\pi_2,\ldots,\pi_n]$ from its inversion vector. Lehmer[9–11] describes a relatively short method for recovering a permutation π from its inversion vector. In this section, we describe another recovery method from the inversion vector $I_\pi=[I_1,\ldots,I_n]$ generated by Lehmer's method. By the definition of Lehmer's RS inversion method, I_i is the number of π_j's, $i<j\leqslant n$, that are less than π_i to the right of π_i in π. Let ℓ_i be a linked list of elements that are candidates for π_i. Thus, initially $\ell_1=(1,2,\ldots,n)$. Since there are I_1 elements smaller than π_1 to the right of π_1 in π, $\pi_1=\ell_1(1+I_1)$. Now, the new linked list ℓ_2 of candidates for π_2 is obtained by deleting π_1 from ℓ_1. Since there are I_2 elements smaller than π_2 to the right of π_2 in π, $\pi_2=\ell_2(1+I_2)$, and so on. We now see that given I_i, $\pi_i=\ell_i(1+I_i)$. We proceed to describe this procedure in the following algorithm. For simplicity, we use ℓ instead of ℓ_i in the actual implementation.

1. Let $i\leftarrow 1$, $\ell\leftarrow(1,2,\ldots,n)$.
2. $\pi_i\leftarrow\ell(I_i+1)$.
3. $\ell\leftarrow\ell-\ell(I_i+1)$ [delete $\ell(I_i+1)$ from ℓ].
4. $i\leftarrow i+1$, if $i>n$ stop, otherwise go to 2.

For example, the permutation

$$\pi=[3,1,5,2,4]$$

generates

$$I_\pi=[2,0,2,0,0].$$

To obtain, the permutation π from I_π, we initialize linked list $\ell=(1,2,3,4,5)$ of size $|I_\pi|=5$. Scanning, from left to right of I_π, we see that $I_1=2$. This means that, there are two elements smaller than π_1 in ℓ. Accessing the $2+1=3$rd position in ℓ, we can retrieve the first element of π, which is $\pi_1=3$. On determining π_1, we drop it from ℓ. Hence, ℓ has four elements now, namely $\ell=(1,2,4,5)$. The next element in I_π is $I_2=0$. Accessing the $0+1=1$st position in ℓ, we obtain the second element of π, namely $\pi_2=1$. After determining $\pi_2=1$, we drop 1 from ℓ, we have $\ell=(2,4,5)$. The third element in I_π is $I_3=2$. Accessing the $2+1=3$rd position in ℓ, we obtain the third element of the permutation π, $\pi_3=5$. Repeating the process in this manner, we can obtain the original permutation π.

Instead of counting the number of elements that are smaller than π_i among the elements to the right of π_i in π, one can count the number of elements that are smaller than π_i to the left of π_i in π and obtain a different inversion vector. We call this method, left smaller (LS). A similar bijection can easily be defined with the permutation π and its inversion vector obtained in this way as well. Thus, the algorithm we have given for RS can be modified for LS. Other variations, which we discuss next, follow the same idea. Two other inversion methods, similar to the methods already mentioned, can be defined as right bigger (RB) and left bigger (LB). These two, instead of counting ''how many smaller elements exist'' to the left (or to the right) of an element π_i, they simply count ''how many bigger elements exist'' to the left (or to the right) of the element π_i in π.

3.3 *Inversions on Multiset Permutations*

We can extend the definition of inversions from permutations to multiset permutations in a similar manner. As we show later, Lehmer-type inversion methods may create more compact data (which has lower dynamic range, with respect to the original data), and hence may help JPEG-like predictive techniques to lower the entropy significantly on some images.

Definition 2. Let $\mathcal{M}=a_1^{f_1}\cdot a_2^{f_2}\ldots a_t^{f_t}$ be a multiset and M be a multiset permutation on $\mathcal{M}$. Let M_i denote the symbol in position i of M, and $n=\Sigma_{i=1}^{t}f_i$.

Initially let $S=(M_1)$, $I\text{LB}_1=0$. Then for $i=2,\ldots,n$,

$$I\text{LB}_i=|\{s\in S|s>M_i\}| \quad \text{and if} \quad M_i\notin S, \quad \text{then} \quad S=S,M_i,$$

where '','' means concatenation operation. We call this the LB inversion method on multiset M.

If we change $s>M_i$ in Definition 2 to $s<M_i$, we obtain the LS inversion method. If we let $S=\{M_n\}$, $I\text{RB}_n=0$, and scan M from right to left, for $i=n-1$ to 1, we define the so-called RB method. Similarly, we define RS. If M is the multiset permutation, $M=[1,1,2,1,3,3,1,2,3,3,5,3]$, then,

LB generates $\mathbf{I}\text{LB}=\langle 0,0,0,1,0,0,2,1,0,0,0,1\rangle$,

LS generates $\mathbf{I}\text{LS}=\langle 0,0,1,0,2,2,0,1,2,2,3,2\rangle$,

RB generates $\mathbf{I}\text{RB}=\langle 3,3,2,3,1,1,3,2,1,1,0,0\rangle$,

and RS generates $\mathbf{I}\mathrm{RS}=\langle 0,0,1,0,2,2,0,0,0,0,1,0\rangle$.

Of all the strings generated by the inversion methods, LB, LS, RB, and RS seem to be more compact, that is, there is no sparse-histogram problem. Since the representations are more compact, is it possible to obtain a lower zero-order entropy for the inversion strings of the multiset permutations? Before we investigate this question, we give an algorithm that shows how to recover an original multiset permutation M from its inversion vector generated by the LB method.

In the previous section, we showed that for any given inversion vector generated by Lehmer variant inversion methods (LB, LS, RB, and RS), we can find the corresponding permutation π. A multiset permutation is like a permutation where the elements are not distinct. By analogy, we can generalize the algorithm given for permutations.

Let $\mathbf{S}=(M_1,M_2,\ldots,M_t)$ be a linked structure that represents the ordered set of elements of M. Let $\mathbf{F}=(f_1,\ldots,f_t)$ be a linked list structure that represents the frequencies (multiplicities) of the distinct elements of M in the order that they appear in **S**. From our definition, $I\mathrm{LB}_i$ is the number of distinct M_j's, $1\leqslant j\leqslant i\leqslant n$, that are bigger than M_i to the left of M_i in M. So, for any given $I\mathrm{LB}_i$, it is sufficient to construct an ordered set, which is represented by the linked list $\mathbf{S}=(M_1,M_2,\ldots,M_k)$, $k\leqslant t$, to determine the corresponding M_i. That is, $I\mathrm{LB}_i$ represents M_i, which is at position $|S|-I\mathrm{LB}_i$. Obviously, we require **F** and **S**, since the elements of M are not distinct, to recover M from **I**LB. So, for any given **I**LB, we can obtain the corresponding M by the following process:

1. Let $i\leftarrow n$.
2. $M_i\leftarrow\mathbf{S}(|\mathbf{S}|-I\mathrm{LB}_i)$.
3. $\mathbf{F}(M_i)\leftarrow\mathbf{F}(M_i)-1$.
4. If $\mathbf{F}(M_i)=0$, delete M_i from **S** and $\mathbf{F}(M_i)$ from **F**.
5. If $i=0$ stop, otherwise go to 2.

The algorithm just given requires $O(t\cdot n)$ time in the worst case, since the **S** and **F** vectors are represented with linked list data structures. Initially, the size of **S** and **F** are t. At each iteration, to obtain the gray value M_i in **S** the process has to move a pointer $|\mathbf{S}|-I\mathrm{LB}_i$ times in **S** (step 2), and to reduce the frequency of M_i by one in **F** the process has to search for the node M_i in **F** (step 3). Both steps 2 and 3 then require $O(t)$ time. If we assume that the pointers in steps 2 and 3 are used, deletion of an element from the linked lists **S** and **F** (step 4) then require constant time. Hence, the worst case time complexity of the recovery algorithm is $O(t\cdot n)$. By using more suitable data structures we can reduce the time complexity. For example, instead of using linked list data structure, we can use array structure to represent the frequency vector **F** and the set of the gray values **S**. The receiver, on receiving the **F** vector from the transmitter constructs **S**. Using a counter variable the process can keep track of the number of gray values in **S**. Whenever frequency count of a gray value becomes zero in **F**, that gray value is deleted from **S**. Figure 1 shows the complete process. Deleting an element from **S** requires at most $O(t)$, since after deleting an element from **S** the process shifts all the gray values that are greater than the deleted gray value to the left by one location. There are at most t values to delete in **S**. Therefore, the time that will be required to delete all the gray values from **S** would be $O(t^2)$. Thus, the worst case total time complexity is $O(t^2+n)$.

The definition given earlier for converting a multiset permutation to its corresponding inversion vector (**I**LB), which is a simple extension of Lehmer method for multiset permutations requires $O(t\cdot n)$. However, again with a simple modification the time complexity can be reduced to $O(t\cdot m+n)$, where $0<t<m$ and m is possible number of

```
Input:   Inversion vector ILB and frequency vector F.
Output:  Multiset permutation M.
Comments:
  F: Frequency vector.
  S: Lookup table. S[i] indicates a gray value.
  Begin

   count:=0 ;  n:=0 ;
   for i := 0 to m − 1 do
        if (F[i] > 0) then begin
            S[count]:= i ;
            n:= n+ F[i] ;
            count:= count +1 ;
         end;
   count:= count -1 ;
   for i := n − 1 to 0 do begin
       M[i]:= S[count-ILB[i]] ;
       F[M[i]] := F[M[i]] -1 ;
       if ( F[M[i]] =  0) then begin
             if ( ILB[i]] <> 0) then
                  for j:= (count-ILB[i]) to (count) do
                      S[j] := S[j+1] ;
             count:= count -1 ;
        end;
   end;
  End.
```

Fig. 1 Algorithm to generate a multiset permutation from its inversion vector.

gray values. For example, in the case of an 8-bit image, m is 256. Hence, let **S**, **F** and **C** be 1-D arrays of size m. Figure 1 shows a conversion algorithm which uses the array structures **S**, **F**, and **C**. From Fig. 1, it is clear that an entry in **S** is updated whenever the process reads a new gray value $M(i)$ that has not been previously seen. It updates the entries of the array **C** whose locations are less than $M(i)$ by simply adding 1. Obviously, the time required to update the entries in **C** is no more than m. This happens at most t times, since the number of different gray values occurred are at most t. Thus, the time complexity for updating the information is $O(t \cdot m)$. Let $t=m$. Then the worst case time complexity to update the information related to the number of gray values that has been previously seen that are greater than the current gray value is $O(t^2)$. Therefore, the total worst case time complexity is $O(t^2+n)$.

Both the modified conversion and recovery algorithms have $O(t^2+n)$ worst case time complexity. For 8-bit images, where $n>t^2$ ($t\leqslant 256$) the overall time complexity is linear with respect to n.

The preceding algorithms can be easily modified for LS, RB, and RS.

4 Simulation Results

For our simulations, we used some images (green band) from the USC database and some others from the University of Nebraska at Lincoln (UNL) compression laboratory of size 256×256.

Note that the inversion vectors obtained from the LB, LS, RS, and RB methods generate different sequences and different entropies. For example, we observe that when we scan the images with raster scan, the entropies generated by the preceding methods vary by approximately ± 0.4 bits/pixel for the same image. However, the entropies generated by the best JPEG predictor, when applied to the data generated by LB, LS, RB, and RS, vary approximately by ± 0.04 bits/pixel. Therefore, in Table 2 we present only the results obtained from applying best JPEG and BJPEG on **I**LB. As can be seen from the results, for certain images, such as the ''USC-girl,'' ''x-ray,'' and ''couple'' images, the performance of the JPEG and BJPEG predictive schemes is quite substantial on preprocessed data. To reconstruct the multiset permutation (image) from its inversion vector, we need to transmit vectors **S** and **F**. This can be achieved by using run-length coding. For each gray value i, we can transmit $\log_2 n$ bits to indicate its frequency f_i, where n is the size of image. Whenever $f_i=0$, it indicates that the gray value i does not exist in the image. Thus, the receiver can construct **F** and **S** from the bit stream. For our images, $n=2^{16}$. Therefore, with an overhead of 0.062 bits/pixel, the information needed to construct **S** and **F** can be transmitted. For images, $n>2^{16}$, overhead would be even smaller. In the case of a 512×512 image the overhead would be $(256\times 18)/(512\times 512)\approx 0.02$ bits/pixel.

For the ''USC-girl,'' ''x-ray,'' and ''couple'' images we obtain a solid gain of 0.87, 0.82, and 0.32 bits/pixel instead of using JPEG alone. Notice that, instead of using JPEG on the entire inversion vector, if we use JPEG on blocks

```
Input:    Multiset permutation M.
Output:   Inversion vector ILB and frequency vector F.
Comments:
  F: Frequency vector.
  S: Lookup table. If S[i] = 1, gray value has occurred.
  C: Counter table. C[i] indicates how many gray values
     bigger than i has seen so far.

Begin
 for i := 0 to m − 1 do begin
     S[i]:= -1; C[i]:= 0 ; F[i]:=0;
   end;
 for i := 0 to n − 1 do begin
     if (S[M[i]] = −1) then begin
        S[M[i]] := 1 ;
        for j:= 0 to M[i] − 1 do
            C[j] := C[j] + 1 ;
      end;
      ILB[i] := C[M[i]] ;
      F[M[i]] := F[M[i]] + 1 ;
 end;
End.
```

Fig. 2 Algorithm to convert a multiset permutation to its inversion vector.

(BJPEG) of the inversion vector we obtain a further gain at a small cost such as 0.047 bits/pixel for a block size 8×8. This overhead is required if we choose a block size 8×8 to indicate to the receiver which JPEG method is used for each block. The cost that is incurred from BJPEG is also added to the entropy obtained for each image, as well as the cost needed to transmit **S** and **F** vectors in Table 2. On the remaining test images, the best JPEG or BJPEG predictor, applied directly to the image, produces almost the same results as the best JPEG or BJPEG applied to the inversion

5 Adaptive Inversions

Inversions on multiset permutations generate more compact data and eliminate the sparse-histogram problem. To determine inversion value of a data element, the transmitter must maintain an ordered set of the previously seen elements. The receiver, on the other hand, must receive the frequency vector, in addition to the inversion vector, to construct the original data. Although the transmission cost of the frequency vector using the run-length coding requires $n\log_2 n$ bits, it can be reduced to $t\log_2 m$, where m is the set of

Table 2 Entropy of best JPEG and BJPEG predictor on inversion vectors of images.

Image	**H** (Image)	JPEG	JPEG on **I**LB	BJPEG	BJPEG on **I**LB
"Couple"	5.96	4.27	**3.95**	4.05	**3.79**
"Lady"	5.37	3.81	**3.80**	3.60	**3.59**
"Moon"	6.71	5.11	**5.07**	5.14	**5.10**
"Sat1"	7.31	5.89	**5.88**	5.81	5.87
"Tree"	7.41	5.49	5.49	5.36	5.40
"USC-girl"	6.42	4.82	**3.94**	4.72	**3.85**
"x ray"	5.21	5.17	**4.35**	5.86	**4.34**

```
Input: Inversion vector ILB and Gray value vector G.
Output: Multiset permutation M.
Comments:
  G: Gray value table
  S: Lookup table. Initially it is empty.
  Begin
    index := 0
    for i := 0 to n − 1 do begin
        if (ILB[i]] < index) then
            M[i] := S[ILB[i]] ;
        else begin
            M[i] := G[index]
            S[index] := G[index]
            If the elements of S are not in decreasing
              order from left to right, swap the new element
              to preserve the decreasing order.
            index := index + 1 ;
        end;
    end;
  End.
```

Fig. 3 Algorithm to adaptively generate a multiset permutation from its inversion vector.

possible gray values, using adaptive techniques. In Fig. 3, an adaptive version of Fig. 1 is presented. The difference between Figs. 1 and 3 are the following:

1. Instead of always copying the value of $\mathbf{C}[M(i)]$ to $\mathbf{I}\mathrm{LB}(i)$, we copy the value of $\mathbf{C}[M(i)]$ to $\mathbf{I}\mathrm{LB}(i)$ whenever $M(i)$ has been previously seen. Otherwise, we copy the value of the index variable into $\mathbf{I}\mathrm{LB}(i)$.
2. We maintain a gray-value vector **G**, which stores the first occurrences of different gray values in the order that they appear.
3. We no longer need to maintain frequency vector **F**.

The transmitter, after generating **I**LB and **G** vectors, transmits them to the receiver. Transmission of the **G** vector is less costly than for the **F** vector, since $n\times\log_2 m$ bits must be transmitted instead of $n\times\log_2 n$ bits. The inversion vector generated by the adaptive scheme will have at most t $(t\leqslant m)$ values different from the inversion vector generated by the method described earlier.

The receiver having the first occurrences of t different gray values can easily simulate the behavior of the transmitter. Figure 3 presents an adaptive recovery method for the receiver. In Fig. 4, whenever a new gray element is seen, the transmitter inserts the value of the index variable into the **I**LB vector, copies the newly seen gray value into the **G** vector, and increments the value of the index variable by 1. The receiver uses the value of the index variable to determine whether a gray value has been previously seen, or to access into the **G** vector and retrieve a new gray value that has not been seen. A new gray value from **G** is retrieved if the inversion value is equal to the index value. Otherwise, the inversion value is less than the index value. This implies that the value has been seen and should be in the set **S**. The gray values in **S** are maintained in decreasing order from left to right to simulate the ''left bigger'' property.

The inversion vector that will be generated by the transmitter, will have at most t elements that will be different from those in the inversion vector generated by the LB method described earlier. However, the transmission overhead is less while the worst case time complexity is the same $[O(t^2+n)]$. Experimental results on our test images showed that the results of the JPEG and BJPEG predictive schemes on the inversion vectors generated by the described adaptive method is almost the same as the results presented earlier.

6 Discussion

We see that viewing an image as a multiset permutation, and applying inversions to it, can improve the performance of JPEG-like prediction schemes on some images. Conventional prediction schemes such as JPEG or BJPEG, are simple and useful, but in some cases, e.g., the ''x-ray'' image, the entropy of prediction error becomes greater than that of the original image data. Using inversions on multiset permutations cures this problem by generating a more compact representation. It reduces the gap between neighboring

```
Input:    Multiset permutation M.
Output:   Inversion vector ILB.
Comments:
  G: Gray value table
  S: Lookup table. If S[i] = 1, gray value has occurred.
  C: Counter table. C[i] indicates how many gray values
     bigger than i has seen so far.

Begin
 for i := 0 to m − 1 do begin
     S[i]:= -1; C[i]:= 0 ; G[i]:=0;
   end;
   index:= 0 ;
 for i := 0 to n − 1 do begin
    if (S[M[i]] = −1) then begin
       S[M[i]] := 1 ;
       for j:= 0 to M[i] − 1 do
            C[j] := C[j] + 1 ;
       ILB[i] := index ;
       G[index] := M[i] ;
       index := index + 1 ;
     end;
    else ILB[i] := C[M[i]] ;
 end;
End.
```

Fig. 4 Algorithm to adaptively convert a multiset permutation to its inversion vector.

pixel elements. This process effectively reduces the range of prediction error by curing the sparse-histogram problem. Hence, it is logical to expect a lower entropy. But, as shown by Table 2, this does not happen for all the images, even though all test images we are using have less than 256 gray values. Location of gray values and their relationship to the neighboring gray values affects the prediction error in the image. Some pixel elements may preserve their absolute gray differences between neighboring elements.

Memon and Ray[4] posed the question of exchanging gray values and the effect of this on the entropy. They formulated the problem as optimal linear ordering of gray values and suggested two algorithms to attack the problem. Kuroki et al.[3] reported similar results, where the authors are performing simple gray-level conversion. The later method is also suggested by Wang.[12] The methods just mentioned require two passes. For example, the gray-level conversion method requires a prescan of the image to determine the gray values. It then does a gray-level conversion before decorrelating the data with predictive operators. Optimal linear ordering also requires two passes and has a much higher time complexity requirement. In the first pass, it constructs the adjacency matrix of the gray values of an image data. It then determines the optimal linear ordering of the gray values, and finally decorrelates the data. However, inversions on multiset permutations can operate with one pass. Also, our algorithms have a linear time complexity with respect to the size of the image and are suitable for multiprocessing or parallel processing. While the current portion of the image data is being processed, previously operated data can be sent to another process or processor to apply predictive operators onto the processed data. This speeds up the operation and reduces the total compression time, especially when parallel processing is employed.

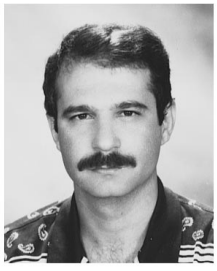
We have not yet established whether any further gain can be achieved by applying our method of inversions in addition to an optimal linear ordering. It is worth investigating whether applying optimal linear ordering to each of the inversion sequences will yield any gain.

Acknowledgment

The author is thankful to Prof. Dr. David Klarner, Prof. Dr. Spyros S. Magliveras, and Dr. Nasir Memon for their valuable help and comments in preparation of this work.

References

1. J. J. Rissanen and G. G. Langdon, ''Universal modeling and coding,'' *IEEE Trans. Inf. Theory* **27**(1), 12–22 (1981).
2. G. K. Wallace, ''The JPEG still picture compression standard,'' *Commun. ACM* **34**(4), 31–44 (1991).
3. N. Kuroki, T. Nomura, M. Tomita, and K. Hirano, ''Lossless image compression by two-dimensional linear prediction with variable coefficients,'' *IEICE Trans. Fund.* **E75-A**(7), 882–889 (1992).
4. N. D. Memon and S. Ray, ''Ordering color maps for lossless compression,'' in *Visual Communications and Image Processing, Proc. SPIE* **2308**, 1192–1203 (1994).
5. W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Compression Standart*, Van Nostrand Reinhold, New York (1993).
6. N. D. Memon and K. Sayood, ''Lossless image compression—a comparative study,'' in *Still Image Compression, Proc. SPIE* **2418**, 8–20 (1995).
7. M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York (1979).
8. D. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA (1973).
9. R. Sedgewick, ''Permutation generation methods: a review,'' *ACM Comput. Surv.* **9** (2), 137–164 (1977).
10. D. H. Lehmer, ''Teaching combinatorial tricks to a computer,'' in *Proc. of Symp. Applied Mathematics, Combinatorial Analysis*, Vol. 10, pp. 179–193, American Mathematical Society (1960).
11. D. H. Lehmer, *The Machine Tools of Combinatorics, Applied Combinatorial Mathematics*, pp. 5–31, John Wiley and Sons, New York (1964).
12. Y. Wang ''A set of transformations for lossless image compression,'' *IEEE Trans. Image Process.* **4**(5), 677–679 (1995).



Ziya Arnavut a research analyst and lab manager at the University of Nebraska at Omaha, Department of Geography and Geology. He received his BS in mathematics from Ege University in Turkey, his MS in computer science from the University of Miami, and his PhD in computer science from the University of Nebraska-Lincoln. His research interests are data compression, algorithms, and remote sensing analysis.