TABLE I

<table>
<thead>
<tr>
<th>Signal Level</th>
<th>Likelihood Ratio Threshold</th>
<th>Equivalent Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Threshold in parenthesis</td>
</tr>
<tr>
<td>1.0</td>
<td>100</td>
<td>ct(0.6,2.0) Linear Wilcoxon (18)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same as Min (18) or Symmetric (6,6)</td>
</tr>
<tr>
<td>1.0</td>
<td>36</td>
<td>ct(0.5,2.0) Linear Wilcoxon (17)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same as Symmetric (6,6,5)</td>
</tr>
<tr>
<td>1.0</td>
<td>11.39</td>
<td>ct(0.5,1.0) Symmetric (6,6,4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ct(0.5,2.0) Linear Wilcoxon (16)</td>
</tr>
<tr>
<td>2.0</td>
<td>2.55</td>
<td>ct(1.0,2.0) Symmetric (6,6,4)</td>
</tr>
</tbody>
</table>

The purpose of this correspondence is to introduce an efficient method for encoding sparse binary patterns (images), where the term “sparse” implies that the patterns consist of a small number of ones, relative to the number of zeros.

The technique we consider will be referred to as **block coding**. It is shown that block coding enables us to encode sparse binary patterns with average code word lengths $L_w(p)$ that compare very closely to the source entropy $H(p)$ when $p$ is small, where $p$ is the probability of finding a one in the given pattern. Since $L_w(p)$ closely approximates $H(p)$, we can view such block codes as being close to optimum for encoding sparse binary patterns [1].

The sparse pattern we deal with is assumed to be a memoryless binary source. This kind of pattern is found in a 3-D authentication scheme [2]. In data compression, the patterns are usually not memoryless sources. However, when LPC (Linear Prediction Coding) is applied, the resulting error pattern is very close to a memoryless model. Yasuda [3] presented some effective methods to decorrelate 2-D facsimile patterns. For example, Boolean algebra prediction functions [3, p. 834] are shown to be very effective for typical images written English and Japanese documents and weather maps. The error patterns that result via prediction are sparse, and hence our block coding technique may be useful for this application also. After the block coding method is introduced, it is compared to some other existing methods.

## II. BLOCK CODING

For the purposes of discussion, we consider a $(128 \times 128)$ sparse binary pattern in which the probability of finding a one is $p = 0.01$. As such, the probability of finding a zero is $q = 1 - p = 0.99$. If this pattern is scanned on a row-by-row basis, it follows that we obtain a 1-dimensional array consisting of $n = 16384$ bits.

The proposed block coding scheme consists of the following steps.

1. Map a 2-D image into a 1-dimensional array by row-by-row scanning. The 1-dimensional array consists of $n = 2^m$ bits and one.

2. Divide the $2^m$ bit-string obtained in Step 1) into $2^n$ blocks, with each block consisting of $2^n$ bits; it then follows that $M = a + b$.

3. Between any two adjacent blocks we introduce a comma, which is encoded as a “0.”

4a) If there is no one in a block, then no coding is needed for the block.

4b) If there are ones in a block, then assign each one a prefix “1” followed by $b$ bits to indicate its location in the block. This location is with respect to the left end of the $2^n$-bit string and numbered from 0 through $2^n - 1$. The reason for the prefix “1” is to realize an instantaneous code.

5) The bit-string resulting from Step 4 is desired code.

The decoding procedure is just the reverse of the coding procedure. In general, if we have $k$ ones and $n - k$ zeros, the code length, $L(k)$, is given by

$$L(k) = (\text{rows} - 1) + k(\text{coding bits per one}) = (2^m - 1) + k(b + 1) = (2^n - 1) + k(M - a + 1).$$

The probability of $k$ ones and $n - k$ zeros occurring is

$$\Pr(k) = \frac{n!}{k!} b^k q^{n-k}.$$
Given pattern:

\[
\begin{array}{cccc}
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

Step 1 yields:

\[01001100000000\]

Steps 2-4 \(2^n = 2^4 = 4\) yield:

\[
\begin{array}{cccc}
0100 & | & 0110 & | & 0000 & | & 0000 \\
\end{array}
\]

Step 5 yields the desired code to be:

\[1016:011:0000\]

Fig. 1. Illustrative example related to block encoding scheme.

The average code length, \(L_{av}\), is found as

\[
L_{av} = \sum_{k=0}^{n} \Pr(k) L(k)
= \sum_{k=0}^{n} \binom{n}{k} \rho^k q^{n-k} \left[ (2^k - 1) + k(M - a + 1) \right]
= (2^n - 1) + (M - a + 1) np.
\]

(3)

Here, \(L_{av}\) is a function of \(a\). We now choose \(a\) such that \(L_{av}\) achieves its minimum. Setting the derivative of \(L_{av}\) with respect to \(a\) to zero yields

\[
\frac{dL_{av}}{da} = 2^n \ln 2 - np = 0
\]

or

\[
a = \log_2 \left( \frac{np}{\ln 2} \right).
\]

(4)

(5)

For the case we are considering, \(n = 16384\) and \(p = 0.01\). Thus, \(a = 7.885\), which upon rounding yields

\[
a = 8.
\]

(6)

Substituting \(M = 14\), \(n = 2^M = 16384\), \(a = 8\), and \(p = 0.01\) in (3) leads to

\[
L_{av} = 1401.88 \text{ bits.}
\]

(7)

The value of \(L_{av}\) in (7) enables us to locate a point on the \(L^{\text{block}}(p)\) curve in Fig. 2 corresponding to \(p = 0.01\). The remaining points are obtained in a similar manner.

III. COMPARISON

It is well known that the lower bound for the average length of any code is the source entropy \(H(p)\). In our example, the source entropy is given by

\[
H(p) = n \left( p \log_2 \frac{1}{p} + q \log_2 \frac{1}{q} \right).
\]

(8)
Fig. 2 shows that $H(p)$ is indeed the lower bound of all the coding methods.

Obviously, if the pattern is too sparse, one can just code it by the coordinates of its ones. When $n = 2^{14}$, each one requires 14 bits; thus, the average codeword length is

$$L_{ave}^{(Coord)} = 14np$$

which is a straight line in Fig. 2.

Another technique we compare with is the well-known run-length coding [3], [4]. The average codeword length is denoted as $L_{ave}^{(Run)}$ in Fig. 2.

Last, but certainly not least, another block coding [5] will be compared. Since this method was first developed by Kunt [6], we refer to it as Kunt’s method. In Kunt’s method, a pattern of $n$ bits is divided into $n_1$ blocks, each block having $n_2$ bits, i.e., $n = n_1 n_2$. If a block is all zero, it is coded by a ‘‘0,’’ or else it is coded by a prefix ‘‘1’’ and followed by the original block. Clearly, the average codeword length is given by

$$L_{ave}^{(Kunt)} = \frac{n_1 + n_2}{n_2} [1 - \text{Probability (all - zero block)}]$$

As indicated in [5], it is difficult to find the optimum block size. Here, we find the optimum block size by brute force, and plot $L_{ave}^{(Kunt)}$ versus $p$ in Fig. 2.

In Fig. 2, the horizontal axis is from $p = 0.00005$ to $p = 0.291$. This is because when $p < 0.291$, average codeword lengths for all methods are greater than 16384. Observed from Fig. 2, the best coding methods are listed below.

| $0.191 < p < 0.291$ | Kunt’s block coding |
| $0.00018 < p < 0.191$ | Block coding |
| $p < 0.00018$ | Coordinate coding |

When $0.0014 < p < 0.0059$, run-length coding is a little better than block coding. But run-length coding needs storage of a codebook, whereas block coding does not.

IV. Conclusion

It has been shown that the proposed block coding is very efficient for encoding sparse patterns; e.g., $p < 0.191$ when $n = 16384$. Usually error patterns resulting via prediction satisfy $p < 0.191$. Therefore, block coding could find applications in coding such error patterns. Also, as indicated in Fig. 2, $L_{ave}^{(Block)}(p)$ is close to the optimal standard $H(p)$ when $p$ is small. If a block length is optimized for $p = p_o$, but the actual $p$ turns out to be $p_i$, the resulting loss in performance is $|L_{ave}^{(Block)}(p_o) - L_{ave}^{(Block)}(p_i)|$ as indicated in Fig. 2.

References