Arithmetic Coding with Dual Symbol Sets and Its Performance Analysis

Bin Zhu, *Member, IEEE,* En-hui Yang, *Member, IEEE,* and Ahmed H. Tewfik, *Fellow, IEEE*

***Abstract*—In this paper, we propose a novel adaptive arithmetic coding method that uses dual symbol sets: A primary symbol set that contains all the symbols that are likely to occur in the near future and a secondary symbol set that contains all other symbols. The simplest implementation of our method assumes that symbols that have appeared in the recent past are highly likely to appear in the near future. It therefore fills the primary set with symbols that have occurred in the recent past. Symbols move dynamically between the two symbol sets to adapt to the local statistics of the symbol source. The proposed method works well for sources, such as images, that are characterized by large alphabets and alphabet distributions that are skewed and highly nonstationary. We analyze the performance of the proposed method and compare it to other arithmetic coding methods, both theoretically and experimentally. We show experimentally that in certain contexts, e.g., with a wavelet-based image coding scheme that has recently appeared in the literature, the compression performance of the proposed method is better than that of the conventional arithmetic coding method and the zero-frequency escape arithmetic coding method.**

***Index Terms*— Arithmetic coding, data compression, entropy coding, image compression, JPEG, wavelet.**

I. Introduction

ARITHMETIC coding [1]–[4] is a very efficient entropy coding technique. It is optimal in theory and nearly optimal in practice, in that it encodes arbitrary data with minimal average code length. It is widely used in text and image compression.

Arithmetic coding has two major advantages over other entropy coding methods, such as Huffman coding. First, its coding efficiency is generally higher than those of other entropy coding methods. This is particularly true when the probability of a symbol is close to one. Second, in arithmetic coding, coding and source modeling are separate. Therefore, an arithmetic coder can work in conjunction with any model that can provide a sequence of event probabilities. The latter advantage is significant because large compression gains can be obtained only through the use of sophisticated models that provide more accurate probabilistic descriptions of the input data. The data model used with arithmetic coding can be adaptive. Adaptive arithmetic coding techniques yield very good compression with a wide variety of source data. Adaptation is particularly useful with sources, such as images, that are generally not statistically stationary.

Note that adaptive arithmetic coding is a one-pass procedure: The coder builds up the statistical model while coding the input data. Static entropy coders need two-passes or use fixed statistical models. Two-pass coding is not efficient for coding large data. Furthermore, the statistical model has to be transmitted to the decoder. This reduces the total compression coding efficiency. Some coders use fixed statistical models and perform only one-pass over the data. However, their compression efficiency can be significantly inferior to coders which use exact models. On the other hand, it has been proven [5] that the compression efficiency of arithmetic coding with an adaptive model is never significantly inferior to arithmetic coding with the exact data model. For these reasons, we concentrate on adaptive arithmetic coding in this paper.

In arithmetic coding, no symbol can have a zero probability of occurrence. Arithmetic coding uses an interval to represent the probability of a sequence. In particular, it uses a number in the interval to differentiate the sequence from all other possible sequences. A zero probability implies zero interval. Thus, it cannot be represented by any number within the interval. In practice, arithmetic coding is usually implemented with incremental transmission and reception and with arithmetic operations of fixed precision [1]–[3]. Fixed precision arithmetic coding simplifies implementation in software and hardware and reduces complexity. However, a fixed precision implementation means that the probabilities of input symbols must be larger than or equal to the smallest nonzero number that can be represented with the given number of bits. This restriction can cause a problem when arithmetic coding is applied to a source that has a lot of symbols with probabilities close to zero. Since arithmetic coding must accommodate all symbols in the alphabet, we have to raise the probabilities of the symbols that do not occur locally to the minimum probability value that can be represented with fixed precision. The resulting distorted source probability model reduces the efficiency of arithmetic coding.

There are several ways of addressing this problem. For example, one can use higher precision arithmetic in the implementation given in [3]. However, high precision arithmetic slows the adaptation of the algorithm described in [3] to the source statistics. Another solution [6] to the problem is to allocate probabilities (intervals) only to symbols that have occurred in the past (instead of to all possible symbols) and

Manuscript received November 17, 1995; revised October 31, 1997. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Michael W. Marcellin.

B. Zhu is with Cognicity, Inc., Edina, MN 55439 USA.

E. Yang is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ont., N2L 3G1 Canada (e-mail: ehyang@bbcr.uwaterloo.ca).

A. H. Tewfik is with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: tewfik@ece.umn.edu).

Publisher Item Identifier S 1057-7149(99)09359-8.

reserve some space for escaping to a new symbol. If the current symbol is new, a special escape symbol is first coded, followed by the new symbol itself. The new symbol is then put into the list of symbols and allocated an interval to represent its probability of occurrence. This technique has been extensively used in the "prediction by partial match" (PPM) method [7], which uses Markov models of different orders to estimate the source statistics. Throughout the rest of the paper, this method will be referred to as *zero-frequency escape arithmetic coding* (ZFEAC). We will refer to Witten *et al.*'s implementation [3] of arithmetic coding as WNCAC.

In image coding, the alphabet typically consists of quantized transform coefficient values or processed pixel values. It may also correspond to pixel values in some high bit rate waveform coders. The alphabet is usually large; a straightforward enumeration of all symbols in it requires several bits per pixel. Many symbols in the alphabet may have a zero or very small probability of occurrence in a given image. In addition, the symbol distribution changes with location within the image and is generally very skewed. For example, a smooth region may produce a small set of symbols while a textured region may yield a large set of symbols.

Note that some image coding procedures use small alphabet sizes. For example, the embedded wavelet zero-tree image coding (EZW) [8] uses a clever iterative thresholding method to reduce the symbol set size to only four. Our emphasis in this paper is on image coding algorithms that do use large alphabets.

ZFEAC can partially address the problems that arise in image coding by effectively reducing the size of the symbol set. However, as we shall see later, the skewedness and spatial variability of the symbol distributions in image coding reduce its coding efficiency. For example, it is not effective when used in conjunction with wavelet-based image coding approaches. In these schemes, some symbols can appear at the beginning of the symbol sequence to be coded and then never appear again.

In this paper, we introduce a novel adaptive arithmetic coding technique called the dual set arithmetic coding (DSAC). The method adapts its *symbol set* to the local statistics of the symbol source. It can effectively deal with the above-mentioned problems with a small increase in complexity. More specifically, the method uses two symbol sets: A *primary symbol set* that contains all the symbols that are likely to occur in the near future and a *secondary symbol set* that contains all other symbols. The primary set acts as the default symbol set. The coding procedure escapes to the secondary symbol set if the current symbol is not in the primary set. This step is similar to ZFEAC except that the new symbol is also entropy-coded in our method. Furthermore, the procedure also removes from the primary set symbols with probabilities of occurrence lower than a prespecified threshold. This allows the proposed approach to better adapt to the local statistics of the symbol source.

We note here that the technique of adapting the symbol set is also used in a lossless image coding method called CALIC [9]. But it is used in a very restricted manner: it lumps the tail of the probability density function into a single escape symbol, and expands and contracts this tail "on the fly."

This paper is organized as follows. In Section II, we propose a novel method which adapts both the symbol set and symbol probabilities. In Section III, we provide a theoretical performance analysis of the proposed arithmetic coding and compare it with conventional arithmetic coding. Finally, experimental results are reported in Section IV in which the proposed method is compared with WNCAC and ZFEAC in several image coding applications.

II. Arithmetic Coding with Dual Sets

A. Arithmetic Coding with Dual Symbol Sets

DSAC uses a special symbol *ESC* in the primary set to escape to the secondary set. Many methods can be used to adaptively shift symbols from one set to the other. Here, we focus on a simple implementation, modeled after WNCAC. Symbols that have appeared in the recent past are assumed to be highly likely to appear in the near future. Further, if the current symbol is not in the primary symbol set, the symbol is moved from the secondary set to the primary. When the frequency counts in the primary set are scaled, the symbols (except *ESC*) with frequency counts less than a threshold are moved to the secondary symbol set.

The algorithm can be described as follows.

1) *Initial State*: Either one of two initial states can be used; (A) is all symbols except *EOF* (end of file) are in the primary symbol set. (B) is all symbols except *ESC* are in the secondary symbol set. State (A) is similar to WNCAC while state (B) is similar to ZFEAC. The frequency counts associated to all symbols are initialized to one. The symbol *ESC* is always in the primary symbol set with frequency count one, while the symbol *EOF* is always in the secondary symbol set since it is used only once.

2) a) If the current symbol is in the primary symbol set, encode the symbol and increase the frequency count associated with it by one. Then go to Step 3.

   b) If the current symbol is not in the primary symbol set, encode the special symbol *ESC*, and use the secondary symbol set to encode the current symbol. The symbol is then moved from the secondary symbol set to the primary set since it is likely to occur in the future.

3) If the sum of all frequency counts in the primary symbol set reaches the maximum frequency count Maxfc, the frequency counts associated with all symbols in the primary symbol set are halved and rounded up to integers. All symbols with a frequency count of one except *ESC* are moved to the secondary symbol set since they are unlikely to occur in the future.

4) Go back to Step 2 to code the next symbol.

The decoding procedure proceeds similarly. Note that if only a small number of symbols occur in the recent past, most symbols will be in the secondary symbol set. Therefore, we can use a smaller Maxfc, low precision integer arithmetic, and thus improve adaptation.

Initial state (B) offers some advantages over initial state (A) when there are many symbols with zero-frequency counts and the length of the symbol sequence to be coded is short. These advantages are essentially similar to those that ZFEAC has over WNCAC. However, these advantages are not as important in DSAC since the effect of the initial state in DSAC disappears once a symbol is moved from the primary set to the secondary set.

B. Comparison with Other Approaches

DSAC adapts the symbol set to the local symbol statistics of the source. A symbol can shift *in* and *out* of the primary symbol set based on the estimated likelihood that it will occur in the near future by using recent observations. This is the key difference between DSAC and other approaches proposed in literature. For example, ZFEAC reduces the symbol set by inserting into the symbol set only those symbols that have appeared in the past. No symbol is removed once it is inserted. Normal arithmetic coding, e.g., WNCAC, uses a fixed symbol set. No symbol can either shift in or out. As we shall see in Section IV, this simple difference can yield an appreciable coding advantage when used in conjunction with certain coding techniques.

When the current symbol is not in the (primary) symbol set, ZFEAC escapes to transmit the new symbol while DSAC escapes to use the second symbol set to encode the symbol. DSAC is more efficient in doing this: Let N be the size of the whole alphabet, and N_s be the size of the secondary symbol set. Note that $N_s \leq N$. ZFEAC encodes a new symbol using $\lceil \log_2 N \rceil$ bits while DSAC encodes it with a *smaller* number $\log_2 N_s$ bits. As the number of symbols in the (primary) symbol set increases, the difference between the number of bits that DSAC and ZFEAC use to encode a new symbol also increases.

C. Variations of the Dual Set Arithmetic Coding

Let us now discuss several variations of DSAC. First we can easily add Markov models to DSAC, in the same way as Markov models are used with normal arithmetic coding. In another variation, the frequency count associated to the escape symbol *ESC* can be chosen to be larger than one. By associating a large frequency count to *ESC*, we reduce the compression effect of coding the extra symbol *ESC* at the expense of distorting the probabilities of the symbols in the primary set. The best choice of the frequency count associated to *ESC* depends on the source to be coded. Note that as the frequency count associated to *ESC* increases, the coding efficiency difference between DSAC and WNCAC decreases. If we set the frequency count of *ESC* to be equal to the size of the secondary symbol set, DSAC uses exactly the same number of bits to encode a sequence of symbols as WNCAC.

The procedure used to shift symbols between the two symbol sets in the proposed method can be replaced by more sophisticated mechanisms. For example, we can use a threshold larger than one, or an adaptive threshold, to move symbols from the primary symbol set to the secondary symbol set or vice versa. In the simple implementation of DSAC described in Section II-A, symbols are shifted from the primary set to the secondary occurs only when frequency counts are scaled. This is simple but not necessary. Symbols can be moved from the primary symbol set to the secondary at any time without scaling frequency counts. For example, we could move a symbol to the secondary symbol set if the symbol does not occur over a certain number of input symbols. This separation of the adaptation of the symbol probabilities and symbol sets could be advantageous.

Other possible modifications include adapting Maxfc to the size of the primary set, using better initial conditions derived from training or information *a priori* to reduce learning time, and using more than two symbol sets.

III. Performance Analysis of DSAC

To analyze the proposed method, we assume that the stream of symbols to be coded is divided into blocks. A block is a sequence of symbols that is processed between two successive scalings of frequency counts. The mth block consists therefore of all symbols that occur between the mth and $(m+1)$st frequency scaling operations. Our analysis can be used regardless of whether or not all blocks have the same size. When scaling is performed in a periodic manner as in [3], all blocks, except for the first one, have the same size. Otherwise, block sizes will vary.

To simplify our analysis, we also assume that coding is performed with exact rational arithmetic. Integer arithmetic with a large fixed range will introduce roundoff errors. Fortunately, it is well known [5] that roundoff errors have very small effects on compression performance and integer arithmetic yields nearly the same coding efficiency as exact rational arithmetic. This is also confirmed by the results presented at the end of this section.

We first introduce the following notation.

- F_m and S_m: Primary and secondary symbol sets, respectively, at the beginning of block m.
- $F_m(i)$ and $S_m(i)$: Primary and secondary symbol sets, respectively *before* coding the ith symbol in block m. Note that $F_m = F_m(1)$, and $S_m = S_m(1)$.
- $n_{a,m}$: The frequency count associated with a symbol $a \in F_m$ at the beginning of block m.
- $T_m = \sum_{a \in F_m} n_{a,m}$: Sum of the frequency counts corresponding to the symbols in the primary set at the beginning of block m.
- B_m: Size of block m. Note that B_m is equal to Max$fc - T_m$.
- $c_{a,m}$: Number of occurrences of a symbol $a \in F_m \cup S_m$ in block m.
- V_m: Subset of S_m that consists of all $a \in S_m$ which appear in block m.
- p_i: Weighted probability used to encode the ith symbol in block m.
- $|G|$: Cardinality of a finite set G.
- $A^{\overline{B}} = A(A+1)\cdots(A+B-1)$, with $A^{\overline{0}} = 1$.
- $A^{-\overline{B}} = A(A-1)\cdots(A-B+1)$, with $A^{-\overline{0}} = 1$.

Let us begin by noting that exact arithmetic coding uses the estimated weighted probability[1] p_a of a symbol a to code a. The estimated weighted probability p_a of symbol a in a set G

[1] In the remainder of this paper, we shall use the term "probability of a symbol" by abuse of terminology to denote the conditional probability of the symbol given the image or source sequence to be coded.

is shown in (1) at the bottom of the page. We need $-\log_2 p_a$ bits to code p_a with exact arithmetic coding.

Let the ith symbol in block m be a_i. To determine the probability p_i that the proposed arithmetic coding procedure uses to encode a_i, we need to distinguish three cases: 1) $a_i \in F_m$; 2) $a_i \in S_m(i)$; and 3) $a_i \in F_m(i)$ and $a_i \notin F_m$.

Consider first the case where $a_i \in F_m$. The sum of the frequency counts for all symbols in $F_m(i)$ is $T_m + i - 1$. Since no symbol is moved from the primary symbol set to the secondary symbol set inside a block, $a_i \in F_m(i)$. Hence, the frequency count corresponding to a_i is $n_{a_i,m} + k_{a_i,m}$, where $k_{a_i,m}$ is the number of occurrences of a_i before time i in block m. Thus, in case 1), the probability p_i that the proposed arithmetic coding procedure uses to encode a_i is

$$p_i = \frac{1}{T_m + i - 1} \cdot (n_{a_i,m} + k_{a_i,m}). \tag{2}$$

If $a_i \in S_m(i)$, the proposed procedure encodes two symbols: *ESC* and a_i. Since *ESC* is in the primary set, the probability associated to it is $1/(T_m + i - 1)$. On the other hand, since a_i is in the secondary symbol set the probability associated to it is $1/|S_m(i)|$. Therefore, the probability p_i that the proposed arithmetic coding procedure uses to encode a_i is given by

$$p_i = \frac{1}{T_m + i - 1} \cdot \frac{1}{|S_m(i)|}. \tag{3}$$

The symbol a_i is then moved to the primary symbol set.

Case 3) ($a_i \in F_m(i)$ and $a_i \notin F_m$) implies that a_i was moved from the secondary symbol set to the primary symbol set at an earlier time during the coding of block m. The weighted probability p_i of a_i is therefore given by

$$p_i = \frac{1}{T_m + i - 1} \cdot k_{a_i,m}. \tag{4}$$

Note that (4) is identical to (2) with $n_{a_i,m} = 0$ since $a_i \notin F_m$. Note also that in all three cases, the denominator of p_i contains the factor $(T_m + i - 1)$.

Now note that DSAC and WNCAC assume that the source is memoryless. Therefore, the probability P_m associated to the sequence of symbols in block m is the product of all p_i in that block. Note that P_m is independent of the order in which the symbols appear. In particular, by reordering the terms p_i in P_m, we can write P_m as in (5), shown at the bottom of the page.

Let L_m denote the total number of bits used to code the symbols in block m with DSAC with exact arithmetic, then $L_m = -\log_2 P_m$, or as in (6), shown at the bottom of the next page. Let L'_m be the total number of bits used by WNCAC to encode the symbols in block m. By using a similar approach, we find that L'_m is given by (7), shown at the bottom of the next page. Note that the denominator in the above equation starts with $T_m + |S_m| - 1$ rather than $T_m + |S_m|$. This is due to the fact that the special symbol *ESC* is not used in conventional arithmetic coding.

Therefore, the total saving in bits that results from using the proposed procedure to encode block m is

$$\begin{aligned} L'_m - L_m &= \log_2 \{(T_m + |S_m| - 1)(T_m + |S_m|) \\ &\quad \cdots (T_m + |S_m| + B_m - 2)\} \\ &\quad - \log_2 \{T_m(T_m+1)\cdots(T_m + B_m - 1)\} \\ &\quad - \log_2 \prod_{a \in V_m} c_{a,m} - \log_2 \{|S_m|(|S_m| - 1) \\ &\quad \cdots (|S_m| - |V_m| + 1)\}. \end{aligned} \tag{8}$$

In summary, we have established the following result.

Theorem 1: The total saving in bits that results from using the proposed procedure, rather than the conventional arithmetic coding WNCAC [3], to encode a stream of symbols is given by

$$\begin{aligned} L' - L &= \sum_m \{L'_m - L_m\} \\ &= \sum_m \log_2 \left[(T_m + |S_m| - 1)^{\overline{B}_m}\right] - \log_2 \left[T_m^{\overline{B}_m}\right] \\ &\quad - \log_2 \prod_{a \in V_m} c_{a,m} - \log_2 \left[|S_m|^{-\overline{|V_m|}}\right] \end{aligned} \tag{9}$$

where it is assumed that the last two terms $\log_2 \prod_{a \in V_m} c_{a,m}$ and $\log_2 [|S_m|^{-\overline{|V_m|}}]$ are zero if V_m is null.

If (9) is positive, the proposed algorithm is more efficient than WNCAC. Otherwise, it is less efficient. The first and second terms in (9) represent the saving that is due to the more accurate probability model used by the proposed algorithm. The last two terms in (9) correspond to the overhead associated with using dual symbol sets. When the size of the secondary set is large and few symbols are moved to the primary set, i.e., when $|S_m|$ is large and $|V_m|$ and $\prod_{a \in V_m} c_{a,m}$ are small for all m, $L' - L$ is always positive, and the proposed algorithm provides better compression than WNCAC. This is often the case in lossy image coding, and has been confirmed by the simulations that we shall present in Section IV.

Theorem 1 implies that the performance margin that DSAC enjoys over WNCAC increases when the ratio of the size of the alphabet to the *local* symbol set increases. This is because the difference between the first term and the second term in (9) increases when $|S_M|$ increases. In other words, the performance advantage of DSAC over WNCAC increases as

$$p_a = \frac{\text{frequency count of the symbol } a}{\text{sum of the frequency counts of all the symbols in set } G} \tag{1}$$

$$P_m = \frac{\left[\prod_{a \in F_m - \{ESC\}} n_{a,m}(n_{a,m}+1)\cdots(n_{a,m} + c_{a,m} - 1)\right] \cdot \prod_{a \in V_m} (c_{a,m} - 1)!}{T_m(T_m+1)\cdots(T_m + B_m - 1) \cdot |S_m|(|S_m| - 1)\cdots(|S_m| - |V_m| + 1)} \tag{5}$$

the symbol distribution is more skewed and nonstationary, i.e., as more symbols in the alphabet do not appear locally.

Theorem 1 also implies that the performance difference $|L' - L|$ increases when Maxfc decreases. This is due to the fact that rescaling must be done more often and thus the number of blocks increases. Note that (9) does not depend on the statistical properties of the symbols in F_m for block m. It implies that, among all sources that differ from one another in the probabilities of the symbols in F_m for some block m, the relative saving is higher for sources that can be encoded with less bits.

Finally, Theorem 1 allows us to easily predict when DSAC is more efficient in compression than WNCAC. To illustrate this, we use the following artificial example. Suppose that we need to encode 30 000 symbols in a 256 × 256 image coding application. We assume that the alphabet size is 500, and the *local* symbol set from an image is kept at a size of 50 for simplicity. This is not true in general in real applications, since the local symbol set usually changes with position within the image. Suppose that five symbols are in the secondary symbol set for each block, and 10 b are used for frequency counts. Then, Theorem 1 tells us that the savings that result from using DSAC amount to about 0.64 b/symbol, or 0.30 b/pixel for the image. This translates into 20% saving if the bit rate with conventional arithmetic coding is 1.5 b/pixel.

Let us now develop lower and upper bounds for the number of bits L_m that DSAC uses to encode block m. To this end, we rewrite L_m as shown in (10) at the bottom of the next page (all subscripts m are omitted). Let

$$p^-_{a,m} = \frac{n_{a,m}}{T_m - 1}, \; a \in F_m - \{ESC\}$$

and

$$p^+_{a,m} = \begin{cases} \frac{n_{a,m}+c_{a,m}}{T_m+B_m-1}, & \text{if } a \in F_m - \{ESC\} \\ \frac{c_{a,m}}{T_m+B_m-1}, & \text{if } a \in V_m. \end{cases}$$

Note that $p^-_{a,m}$ and $p^+_{a,m}$ are the weighted probabilities of symbol a at the beginning and end of block m, respectively. Denote by H^-_m and H^+_m the empirical Shannon entropies of the primary symbol set (excluding *ESC*) at the beginning and at the end of block m, respectively. We have

$$H^-_m = \sum_{a \in F_m - \{ESC\}} -p^-_{a,m} \log_2 p^-_{a,m},$$

and

$$H^+_m = \sum_{a \in (F_m - \{ESC\}) \cup V_m} -p^+_{a,m} \log_2 p^+_{a,m}.$$

We now invoke an inequality from information theory [10], which states that if $n = m_1 + m_2 + \cdots + m_k$, where $m_i \geq 1$ for $i = 1, 2, \cdots, k$, then

$$\begin{aligned} &\frac{1}{\binom{n+k-1}{k-1}} 2^{nH((m_1/n), \cdots, (m_k/n))} \\ &\leq \frac{n!}{m_1! \cdots m_k!} \\ &\leq 2^{nH((m_1/n), \cdots, (m_k/n))}. \end{aligned} \tag{11}$$

Since

$$\sum_{a \in F - \{ESC\}} (n_a + c_a) + \sum_{a \in V} c_a = T + B - 1$$

and

$$\sum_{a \in F - \{ESC\}} n_a = T - 1,$$

we obtain by applying (11) to (10)

$$\begin{aligned} L_m &\leq (T+B-1)H^+_m - (T-1)H^-_m \\ &\quad + \log_2 \binom{T+|F|-3}{|F|-2} \\ &\quad + \log_2 \left[\prod_{a \in F - \{ESC\}} (n_a + c_a) \prod_{a \in V} c_a \right] \\ &\quad - \log_2 \left[\prod_{a \in F - \{ESC\}} n_a \right] + \log_2 \left[|S|^{-\overline{|V|}} \right] \\ &\leq (T+B-1)H^+_m - (T-1)H^-_m \\ &\quad + \log_2 \binom{T+|F|-3}{|F|-2} \\ &\quad + (|F|+|V|-1) \log_2 \frac{T+B-1}{|F|+|V|-1} \\ &\quad + \log_2 \left[|S|^{-\overline{|V|}} \right]. \end{aligned} \tag{12}$$

The last inequality follows from the concavity of the logarithmic function ("log(·)").

$$L_m = -\log_2 \left\{ \frac{\left[\prod_{a \in F_m - \{ESC\}} n_{a,m}(n_{a,m}+1) \cdots (n_{a,m}+c_{a,m}-1) \right] \cdot \prod_{a \in V_m} (c_{a,m}-1)!}{T_m(T_m+1) \cdots (T_m+B_m-1)} \right\} + \log_2 \left[|S_m|(|S_m|-1) \cdots (|S_m|-|V_m|+1) \right] \tag{6}$$

$$L'_m = -\log_2 \left\{ \frac{\left[\prod_{a \in F_m - \{ESC\}} n_{a,m}(n_{a,m}+1) \cdots (n_{a,m}+c_{a,m}-1) \right] \cdot \prod_{a \in V_m} c_{a,m}!}{(T_m+|S_m|-1)(T_m+|S_m|) \cdots (T_m+|S_m|+B_m-2)} \right\} \tag{7}$$

Similarly, we have

$$\begin{aligned} L_m \geq{} & (T+B-1)H_m^+ - (T-1)H_m^- \\ & - \log_2 \binom{T+B+|F|+|V|-3}{|F|+|V|-2} \\ & + \log_2 \left[\prod_{a \in F-\{ESC\}} (n_a + c_a) \prod_{a \in V} c_a \right] \\ & - \log_2 \left[\prod_{a \in F-\{ESC\}} n_a \right] + \log_2 \left[|S|^{-\overline{|V|}} \right] \\ \geq{} & (T+B-1)H_m^+ - (T-1)H_m^- \\ & - \log_2 \binom{T+B+|F|+|V|-3}{|F|+|V|-2} + \log_2 \left[|S|^{-\overline{|V|}} \right]. \end{aligned} \tag{13}$$

Therefore, the following bounds characterize the code length that corresponds to the proposed algorithm:

Theorem 2: The code length L of the proposed DSAC for a stream of symbols is bounded by

$$\begin{aligned} & \sum_m \left\{ H_m - \log_2 \binom{T_m + B_m + |F_m| + |V_m| - 3}{|F_m| + |V_m| - 2} \right. \\ & \qquad \left. + \log_2 \left[|S_m|^{-\overline{|V_m|}} \right] \right\} \\ & \leq L \\ & \leq \sum_m \left\{ H_m + \log_2 \binom{T_m + |F_m| - 3}{|F_m| - 2} \right. \\ & \qquad + (|F_m| + |V_m| - 1) \log_2 \frac{T_m + B_m - 1}{|F_m| + |V_m| - 1} \\ & \qquad \left. + \log_2 \left[|S_m|^{-\overline{|V_m|}} \right] \right\} \end{aligned} \tag{14}$$

where $H_m = (T_m + B_m - 1)H_m^+ - (T_m - 1)H_m^-$.

Theorem 2 enables us to estimate the code length without actually applying DSAC to code the sequence of symbols. It is especially useful when we want to find quickly the optimal parameters, such as block sizes, scaling method, etc, for DSAC.

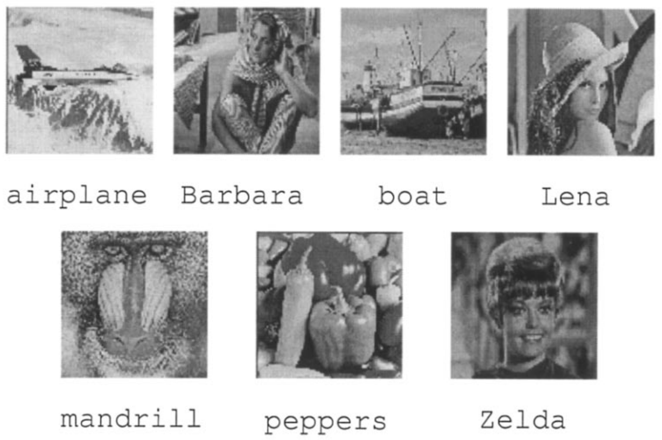


Fig. 1. Images used in the experiments.

Note that expressions (9) and (14) in Theorems 1 and 2 can be further simplified with Sterling's formula $n! \approx \sqrt{2\pi n}(n/e)^n$ for $n \gg 1$.

Similar results can be derived to compare DSAC and ZFEAC, and for the variations of DSAC discussed in Section II-C.

We conclude this section by comparing the predictions of Theorems 1 and 2 to experimental results. Consider the problem of entropy-coding the symbols corresponding to the DCT ac coefficients in JPEG coding [11]. JPEG uses 176 different entropy-coding symbols for the DCT ac coefficients. To perform our experiment, we used the seven standard 512 $\times$ 512 gray-scale (8 b/pixel) shown as thumbnails in Fig. 1 and listed in Table II. Each image was coded at four different quality levels. Further, we set the number of bits for frequency counts $f = 10$. A typical symbol distribution is shown in Fig. 2. Note that the primary symbol set is much smaller than the alphabet. Fig. 3 shows the relative errors between the predictions of Theorem 1 and actual results. The relative error is defined as

$$\text{relative error} = \frac{\text{actual saving bits} - \text{Theorem 1's saving bits}}{\text{actual saving bits}} \times 100\%.$$

$$\begin{aligned} L_m ={} & -\log_2 \left\{ \frac{\left[\prod_{a \in F-\{ESC\}} n_a(n_a+1)\cdots(n_a+c_a-1) \right] \cdot \prod_{a \in V} (c_a - 1)!}{T(T+1)\cdots(T+B-1)} \right\} + \log_2 \left[|S|^{-\overline{|V|}} \right] \\ ={} & -\log_2 \frac{\prod_{a \in F-\{ESC\}} (n_a + c_a - 1)! \prod_{a \in V} (c_a - 1)!}{(T+B-1)!} + \log_2 \frac{\prod_{a \in F-\{ESC\}} (n_a - 1)!}{(T-1)!} \\ & + \log_2 \left[|S|^{-\overline{|V|}} \right] \\ ={} & -\log_2 \frac{\prod_{a \in F-\{ESC\}} (n_a + c_a)! \prod_{a \in V} c_a!}{(T+B-1)!} + \log_2 \frac{\prod_{a \in F-\{ESC\}} n_a!}{(T-1)!} \\ & + \log_2 \prod_{a \in F-\{ESC\}} (n_a + c_a) \prod_{a \in V} c_a - \log_2 \prod_{a \in F-\{ESC\}} n_a + \log_2 \left[|S|^{-\overline{|V|}} \right] \end{aligned} \tag{10}$$

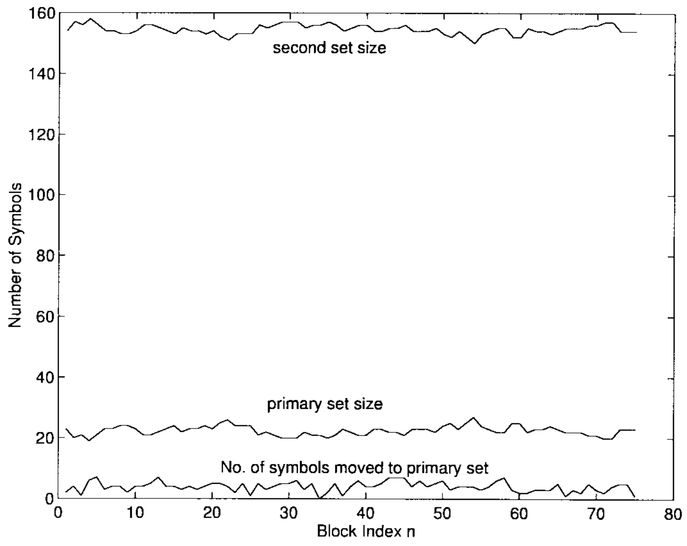


Fig. 2. Typical distribution of symbols.

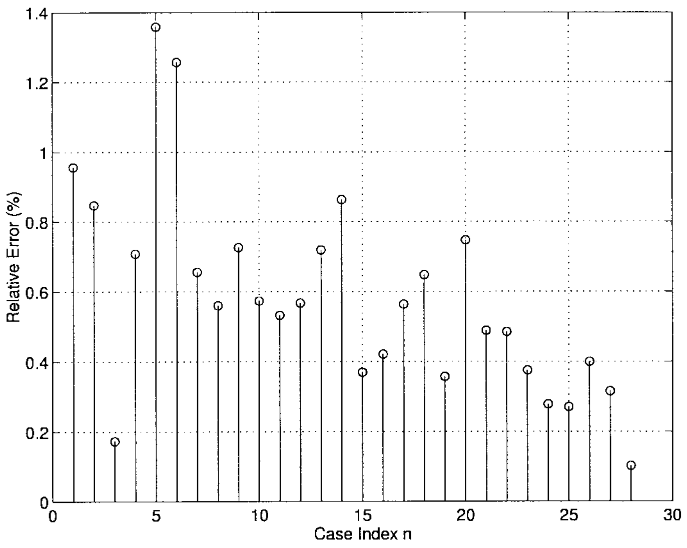


Fig. 3. Comparison between the actual number of saving bits and Theorem 1's prediction.

TABLE I
CASE INDEXES

		Case Indexes			
	JPEG Quality (%)	90	80	60	40
IMAGE	Airplane	1	2	3	4
	Barbara	5	6	7	8
	Boat	9	10	11	12
	Lena	13	14	15	16
	Mandrill	17	18	19	20
	Peppers	21	22	23	24
	Zelda	25	26	27	28

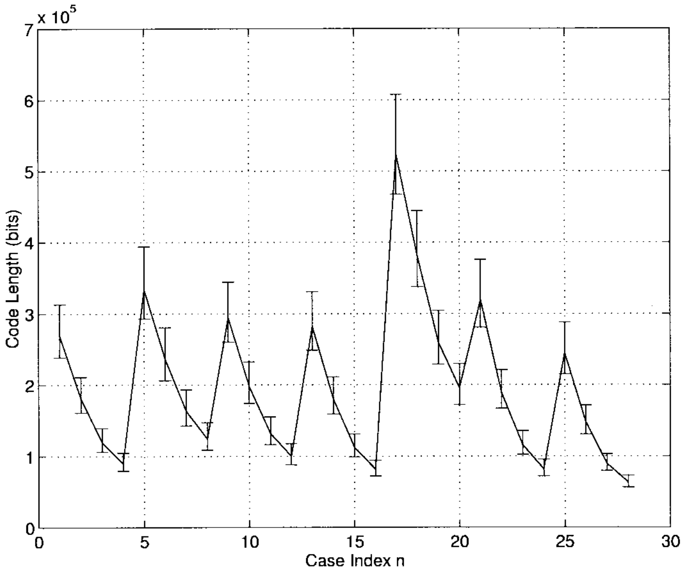


Fig. 4. Actual number of coding bits with the lower and upper bounds from Theorem 2.

The horizontal axis denotes the case number. Each case corresponds to a given image coded at a given quality level as shown in Table I. The corresponding bounds computed from Theorem 2 are compared to the actual number of coding bits used by DSAC in Fig. 4.

Note that the predictions of Theorem 1 are very accurate. Similarly, the bounds given by Theorem 2 are excellent. It is interesting to note that the coding improvements predicted by Theorem 1 are always lower than the actual observed coding improvement values.

IV. Experimental Results

To gain some insight into the performance of the proposed DSAC, we have applied it to several image coding schemes and compared its performance with that of WNCAC [3] and ZFEAC [6]. We measure the coding performance enhancement that results from using the proposed approach with the following dimensionless percentage variable as shown in (15), at the bottom of the next page. Once more, we used the seven standard 512 $\times$ 512 gray-scale (8 b/pixel) shown as thumbnails in Fig. 1 and listed in Table II. We used the same parameters with DSAC, WNCAC, and ZFEAC. Our implementation of DSAC chooses the number of bits f used to represent frequency counts automatically given a user selected range for f and a desired minimum number of blocks. The user selected range for f is chosen such that Max$fc = 2^{f-1}$ is much larger than the size of alphabet. Such a choice provides the proposed method with enough opportunities to potentially shift symbols from the primary symbol set to the secondary symbol set for most of the images that we have tested. (Recall that in the simplest implementation of the proposed method (Section II-B), shifting of symbols from the primary set to the secondary occurs only at scaling time when the total frequency count reaches its maximum value of Maxfc). Further we did not use the special EOF (end of file) symbol since image sizes are known to arithmetic coders.

We first applied the three procedures to the wavelet-based image coding scheme reported in [12]. We used the 9-7 biorthogonal wavelet filters given in [13] to implement the

TABLE II
COMPARISON FOR WAVELET-BASED CODER

Images	WNCAC Bitrate (bpp)	WNCAC Performance Q-coder (%)	WNCAC Performance Overall (%)	DSAC Bitrate (bpp)	ZFEAC Bitrate (bpp)	ZFEAC Performance Q-coder (%)	ZFEAC Performance Overall (%)	PSNR (dB)
Airplane	0.509	16.41	10.07	0.458	0.480	7.36	4.46	34.91
	0.355	21.95	13.27	0.308	0.326	9.73	5.65	32.67
Barbara	0.856	25.76	15.07	0.744	0.783	8.77	5.25	33.47
	0.609	29.71	18.62	0.495	0.536	12.89	7.52	30.89
Boat	0.657	36.24	19.41	0.550	0.574	8.01	4.43	33.80
	0.393	17.47	9.54	0.356	0.369	6.52	3.51	31.55
Lena	0.438	30.60	16.34	0.376	0.398	10.41	5.72	34.87
	0.311	31.85	19.38	0.250	0.267	11.12	6.17	33.00
Mandrill	1.515	13.27	7.61	1.408	1.459	6.06	3.62	30.43
	0.978	16.71	9.24	0.887	0.934	9.07	4.97	27.59
Peppers	0.441	26.25	13.79	0.387	0.408	9.92	5.35	34.12
	0.318	28.29	16.74	0.265	0.282	11.15	6.14	32.64
Zelda	0.235	19.26	10.63	0.213	0.225	10.24	5.83	36.10
	0.160	21.33	12.63	0.140	0.150	11.87	6.91	34.55

approach of [12]. The run-length and nonzero quantized wavelet coefficient values were coded separately with arithmetic coding. The maximum run-length was set to 15 to balance the requirements of the small subimages that are produced by the coarse low frequency stages of the wavelet transform and the large subimages produced by the finer high frequency stages of the wavelet transform. As a result, the run-length coder uses a total of 18 symbols (run-lengths from 0–15, an end-of-subimage symbol and a repeating-run-length symbol for run-length larger than 15). The symbol set corresponding to the nonzero quantized wavelet coefficient values was designed to be as tight as possible in a continuous range. The symbols with minimum and maximum value are transmitted to the decoder. The experimental results for the three arithmetic coding procedures are reported in Table II. All three arithmetic coding procedures yield similar coding results for run-length coding because the size of the symbol set associated with run-length coding is small. Therefore, we reported the performance enhancement in Table II in two forms. The first column is labeled as "Q-coder." It compares the performance of the three coders when used to encode the nonzero quantized wavelet coefficient values only. The second column compares the overall coding performance of the three coders, i.e., it includes the results corresponding to the Q-coder *and* the run-length coder. As we can see from Table II, the proposed DSAC has a significant coding advantage over conventional arithmetic coding (WNCAC). This coding advantage is in the range of 7% to over 19% in overall coding performance (Q-coder + run-length coder) and 13% to over 30% for the "Q-coder" only. The coding advantage of DSAC over ZFEAC is smaller but is still nonnegligible. Both the adaptation of symbol sets and removal of zero-frequency symbols from the symbol set contribute to the performance advantage of DSAC over WNCAC and ZFEAC. The PSNR of the compressed images are also listed in the table.

TABLE III
COMPARISON FOR JPEG CODEC

Images	WNCAC (bpp)	DSAC (bpp)	Performance Entropy (%)	Performance Overall (%)	ZFEAC (bpp)	Performance Entropy (%)	Performance Overall (%)
Airplane	1.197	1.147	6.45	4.22	1.147	0.07	0.04
	0.785	0.749	6.78	4.58	0.749	0.04	0.03
	0.588	0.559	7.07	4.89	0.559	-0.04	-0.03
Lena	1.173	1.122	6.45	4.28	1.121	-0.32	-0.12
	0.739	0.704	6.86	4.66	0.703	-0.27	-0.14
	0.543	0.517	7.06	4.88	0.516	-0.21	-0.14
Mandrill	2.474	2.363	6.97	4.50	2.362	-0.08	-0.05
	1.639	1.561	7.09	4.76	1.560	-0.20	-0.13
	1.215	1.154	7.27	5.01	1.152	-0.34	-0.23
Zelda	0.973	0.928	6.96	4.61	0.927	-0.21	-0.14
	0.602	0.571	7.59	5.15	0.571	-0.04	-0.03
	0.436	0.413	7.94	5.47	0.412	-0.10	-0.06

If we compare the DSAC results with the next-generation wavelet-based image codec, the EZW [8], we find that their performances are very close: For Lena at 0.250 b/pixel, the PSNR corresponding to DSAC is 0.17 dB worse than that corresponding to EZW, while for Barbara at 0.50 b/pixel, it is 0.46 dB better.

Next we applied the three arithmetic coding methods to the JPEG image coding standard. Specifically, we implemented the baseline JPEG procedure [11] and replaced the Huffman entropy coding step by adaptive arithmetic coding. The results of our experiments are shown in Table III. Note that the JPEG algorithm uses a two-step coding procedure to encode each

$$\text{performance} = \frac{\text{coding length with coder A} - \text{coding length with DSAC}}{\text{coding length with coder A}} \times 100\% \qquad (15)$$

TABLE IV
COMPARISON FOR JPEG-LIKE CODEC

Images	WNCAC (bpp)	DSAC (bpp)	Performance (%)	ZFEAC (bpp)	Performance (%)
Airplane	1.138	1.053	7.43	1.051	-0.19
	0.741	0.680	8.29	0.679	-0.15
	0.570	0.508	10.89	0.509	0.20
Lena	1.112	1.029	7.48	1.027	-0.19
	0.701	0.643	8.22	0.641	-0.31
	0.532	0.475	10.61	0.474	-0.21
Mandrill	2.333	2.123	8.99	2.130	0.33
	1.535	1.383	9.89	1.388	0.36
	1.169	1.017	12.97	1.024	0.68
Zelda	0.934	0.847	9.26	0.846	-0.12
	0.581	0.522	10.15	0.520	-0.38
	0.437	0.379	13.23	0.378	-0.26

quantized DCT coefficient: The first step encodes the number of bits used to represent the coefficient. The second step simply represents the coefficient as a variable-length integer (VLI) whose length is specified in the first step. Entropy coding is used only to encode the number of bits used to represent the coefficients. All three arithmetic coding techniques were used to encode these entropy coding values. We used the same VLI number representation in all three implementations. Columns 4 and 7 in Table III show the performance advantage of DSAC over WNCAC and ZFEAC when the entropy coding part only is considered. Columns 5 and 8 show the overall relative coding performances of the three methods. This includes the effects of both the coded entropy values and the variable length integers.

To better compare the three methods, we replaced the JPEG symbol coding scheme with direct entropy coding of all quantized dc and ac DCT coefficients and run-lengths of zeros. We encoded each type of information with a different adaptive model. The sizes of the resulting alphabets depend on the quantization table and the quality scale. Each alphabet was designed to include only the possible symbols in a continuous range for the specific quantized table and the quality scale used. For example, the quantization table we used gives a smallest value of 11 for the ac DCT coefficients. If the quality scale is chosen to be 0.8, the smallest quantization step is 8.8. Since the range of ac DCT coefficients is from -1023 to 1023 (from Table III of paper [11]), the alphabet used to encode these ac coefficients with arithmetic coding includes all integers from -117 to 117 ($117 = \lceil 1023/8.8 \rceil$). We shall refer to this modified coder as the JPEG-like coder. A comparison of the three arithmetic coding procedures for this JPEG-like coder are shown in Table IV. The JPEG-like coder gives the exact same decompressed image as that from the JPEG coder on the same row of Table III. Note also that, as expected, the direct entropy coding of quantized DCT coefficients in the JPEG-like coder usually results in better compression than the two-step approach used in the JPEG standard (Table III).

The performance of ZFEAC with both the JPEG and JPEG-like coders is very close to that of DSAC. The relative coding advantage of DSAC over WNCAC is also smaller for the JPEG and JPEG-like coding algorithms as compared to the wavelet coding procedure of [12] (cf. Tables II and III). This is due to the fact that the wavelet transform yields a better compaction of the signal energy. DSAC is able to better exploit the resulting localized and skewed symbol distribution. In JPEG, the image is divided into 8×8 blocks. These image blocks are independent of each other. The raster scan of the 8×8 image blocks in JPEG and JPEG-like coders reduces the ability of DSAC to efficiently exploit the localization of symbols in the image.

We can conclude from the experimental results that the performance gain of DSAC over WNCAC comes from removal of zero-frequency symbols for both DCT based image coding methods. For the wavelet-based coder, the performance gain comes from removal of zero-frequency symbols *and* the adaptation of the symbol set to the local statistics of the source.

V. CONCLUSION

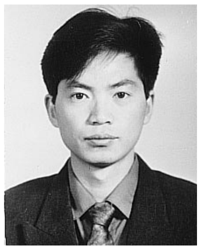
We have proposed a novel arithmetic coding which works well for sources that produce locally a small subset of all possible source outputs. We provided theoretical and experimental comparisons between the proposed algorithm, the conventional adaptive arithmetic coding procedure of [3] and the zero-frequency escape arithmetic coding of [6]. We showed experimentally that the proposed procedure outperforms the other two arithmetic coding techniques when used in conjunction with wavelet-based image coding. This coding advantage is due to the fact that the proposed scheme is better able to exploit the skewedness and variability of the symbol distributions associated with wavelet based image coders. For DCT-based image coding schemes, its performance is still better than that of [3] and comparable to that of [6].

REFERENCES

[1] G. G. Langdon, Jr., and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Commun.*, vol. COMM-29, pp. 858–867, 1981.
[2] C. B. Jones, "An efficient coding system for long source sequences," *IEEE Trans. Inform. Theory*, vol. IT–27, pp. 280–291, 1981.
[3] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, 1987.
[4] P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression," *Proc. IEEE*, vol. 82, pp. 857–865, 1994.
[5] ——, "Analysis of arithmetic coding for data compression," *Inform. Process. Manage.*, vol. 28, pp. 749–763, 1992.
[6] I. H. Witten and T. C. Bell, "The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1085–1094, 1991.
[7] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. COMM-32, pp. 396–402, 1984.
[8] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, pp. 3445–3462, 1993.
[9] X. Wu and N. Memon, "Context-based, adaptive, lossless image codec," *IEEE Trans. Commun.*, vol. 45, pp. 437–444, 1997.
[10] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991.
[11] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, pp. 31–44, 1991.
[12] H. Gharavi and A. Tabatabai, "Sub-band coding of monochrome and color images," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 207–214, 1988.
[13] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using the wavelet transform," *IEEE Trans. Image Processing*, vol. 1, pp. 205–220, 1992.

Bin Zhu (M'97) received B. S. degree in physics from University of Science and Technology of China in 1986 and the M.S. and Ph.D. degrees in electrical engineering from the University of Minnesota, Minneapolis, in 1993 and 1998, respectively.

Since 1996, he has been with Cognicity, Inc, Edina, MN, where he presently holds the position of Lead Scientist. He is one of the founders of Cognicity, Inc. His research interests include multimedia watermarking, compression, and communications, content-based visual data retrieval, and multiscale signal processing.



En-hui Yang was born in Jiangxi, China, in 1966. He received the B.S. degree in applied mathematics from Hua Qiao University, Qianzhou, China, and the Ph.D. degree in mathematics from Nankai University, Tianjin, China, in 1986 and 1991, respectively.

He joined the faculty of Nankai University in June 1991 and was promoted to Associate Professor in 1992. From January 1993 to July 1993, and from January 1995 to August 1995, he was a Research Associate in the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis. During the summer of 1994, he was a guest of the Sonderforschungsbereich "Diskrete Strukturen in der Mathematik," University of Bielefeld, Bielefeld, Germany. From October 1993 to May 11997, he was a Visiting Scholar in the Department of Electrical Engineering–Systems, University of Southern California, Los Angeles. Since June 1997, he has been with the Department of Electrical and Computer Engineering, Faculty of Engineering, University of Waterloo, Waterloo, Ont., Canada. His current research interests are multimedia data compression, digital wired and wireless communications, information theory, Kolmogorov complexity theory, source coding, and applied probability theory and statistics.

Dr. Yang is the recipient of the 1992 Tianjin Science and Technology Promotion Award for Young Investigators and a co-recipient of the third Science and Technology Promotion Award of Chinese National Education Committee in 1992.

Ahmed H. Tewfik (F'95) was born in Cairo, Egypt in 1960. He received the B.Sc. degree from Cairo University, Cairo, Egypt, in 1982, and the M.Sc., E.E., and Sc.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1984, 1985, and 1987, respectively.

He was with Alphatech, Inc., Burlington, MA, in 1987. He is currently the E. F. Johnson Professor of Electronic Communications with the Department of Electrical Engineering, University of Minnesota, Minneapolis. He served as a consultant to MTS Systems, Inc., Eden Prairie, MN, and is a regular consultant to Rosemount, Inc., Eden Prairie. His current research interests are in signal processing for multimedia (in particular watermarking, data hiding and content-based retrieval), low power multimedia communications, adaptive search and data acquisition strategies for world wide web applications, radar and dental/medical imaging, monitoring of machinery using acoustic emissions, and industrial measurements.

Dr. Tewfik is a Distinguished Lecturer of the IEEE Signal Processing Society for the period July 1997 to July 1999. He was a Principal Lecturer at the 1995 IEEE EMBS summer school. He received a Taylor Faculty Development Award from the Taylor Foundation in 1992 and an NSF research initiation award in 1990. He gave a plenary lecture at the 1994 IEEE International Conference on Acoustics, Speech, and Signal Processing in 1994, and an invited tutorial on wavelets at the 1994 IEEE Workshop on Time-Frequency and Time-Scale Analysis. He was selected to be the first Editor-in-Chief of the IEEE SIGNAL PROCESSING LETTERS in 1993. He is a past Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING and was a Guest Editor of two special issues of the TRANSACTIONS on wavelets and their applications.