Arithmetic Coding with Adaptive Context-Tree Weighting for the H.264 Video Coders

Danny Hong *a, Mihaela van der Schaar †b and Béatrice Pesquet-Popescu ¤c

*aDept. of Electrical Engineering, Columbia University, New York, NY, USA
bDept. of Electrical and Computer Engineering, UC Davis, Davis, CA, USA
cDept. TSI, Ecole Nationale Supérieure des Télécommunications, Paris, France

ABSTRACT

We propose applying an adaptive context-tree weighting (CTW) method in the H.264 video coders. We first investigate two different ways to incorporating the CTW method into an H.264 coder and compare the coding effectiveness of using the method with that of using the context models specified in the H.264 standard. We then describe a novel approach for automatically adapting the CTW method based on the syntactic element to be coded and the encoding parameters. We show that our CTW-based arithmetic coding method yields similar or better compression results compared with the context-based adaptive arithmetic coding method used in H.264, without having to specify so many context models.

Keywords: Context-Tree Weighting, H.264, Arithmetic Coding, Context Modeling, Universal Coding

1. INTRODUCTION

Elias coding [1], the basis of arithmetic coding, has been shown to be very effective as it can compress any stationary source sequence very close to its entropy rate, which is very hard to achieve with Huffman coding [2]. Though it is based on Elias coding, Pasco [3] and Rissanen [4] are considered to be the pioneers of arithmetic coding, as they were able to fix the shortcomings that make Elias coding impractical. Subsequently, Witten et al. have published a paper describing a practical implementation of an arithmetic coder in 1987 [5], and since then, many new schemes have been proposed to decrease the complexity of the coder [6, 7, 8, 9]. As the coder became more practical, international standards began to incorporate arithmetic coding, and some of the standards using arithmetic coding are JBIG [10, 11], JPEG2000 [12], and MPEG-4 [13].

Arithmetic coding can be separated into two main parts: modeling and compression. The modeling part can further be separated into two subparts: context modeling and adaptive modeling. The context modeling part captures the structure of the data to be encoded and help an arithmetic coder to eliminate inter-symbol redundancies. For certain types of data, the structure is generally well known: in a text document, a space almost always follows a period and for images, a pixel has a similar intensity value as its neighboring pixels. With these types of structures known, effective context models can be built to predict the value of the next symbol to be coded with high probability. For example, in the MPEG-4 standard, the shape information is ultimately coded using context-based arithmetic coding, and the context models used in the standard are shown in Figure 1. The models define which set of neighboring pixels (bits) should be used to predict the current pixel (bit). In Figure 1(a), the box marked “?” denotes the current pixel to be coded and the other boxes represent the neighboring pixels used for defining contexts. Since there are 10 pixels in the template (the context model), there are 2^{10} possible contexts, where each context corresponds to a probability distribution of the current pixel to be coded. Note that the template given in Figure 1(a) is the same as the template (three-line model template) defined in JBIG for encoding the lowest resolution layer (in progressive coding).

* danny@ee.columbia.edu
† mvanderschaar@ece.ucdavis.edu
‡ pesquet@tsi.enst.fr
Figure 1: Context models used in MPEG-4 shape coding. The box marked “?” is the current pixel to be coded. (a) The Intra template for coding an Intra-coded BAB. (b) The Inter template where C6 is aligned with the current pixel.

In the H.264 standard [14], there are more than 260 context models defined to account for different structures each sequence of symbols may have. More information about the context-based adaptive binary arithmetic coder (CABAC) used in the standard is given in Section 2.1. In addition to context modeling, adaptive modeling is also incorporated into the CABAC so that the statistics of the source sequence, which closely matches the “true” statistics of the source sequence, is calculated. In the MPEG-4 shape-coding scheme, a fixed distribution is used for each context. Similarly, most of the standards (e.g., Group 3 [15] and Group 4 [16] fax, MPEG-1 [17], MPEG-2 [18], and etc.) specify static Huffman tables corresponding to fixed distributions; however, there will be cases where a given source has a noticeably different statistics from the ones implied from the tables. Consequently, for such cases, the coding will not be as effective. Unlike Huffman coding, arithmetic coding can be easily adapted to different context and adaptive models. Also note that most image/video sources have some non-stationary characteristics and this is why it is important for arithmetic coding to be adaptive accordingly.

There are many effective adaptive models. For example, in [19] Ghanbari showed that using just a limited number of past symbols, the performance of the adaptive model can be improved to yield compression result better than the first-order entropy of the source. Similar to Ghanbari’s method, Frydrych and Toivanen proposed a sliding window method that uses a window (containing fixed number of recently read symbols), which limits the symbols that are counted in frequency table [20]. As a new symbol is read from the input, its frequency count is incremented and the frequency of the symbol running out of the window is decremented. Zhu also proposed a method that adapts its symbol set to the local statistics of the symbol source [21]. This 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. Depending on the past history, symbols are moved back and forth between the two sets. Note that in H.264, we are dealing with binary arithmetic coding: the source alphabet consists of only two symbols, and thus, Zhu’s method will not apply.

Coming up with effective context and adaptive models is not a straightforward task. It can be very hard or even impossible to find an appropriate model for a source, and the chosen model might not work well for all possible sources. To avoid such downfalls, a universal coding can be applied. For example, the Lempel-Ziv (LZ) method [22], which is (universally) asymptotically optimal, can be applied to obtain good compression results for any type of (stationary) sources. Alternatively, the context-tree weighting (CTW) method [23, 24, 25] can be used along with an arithmetic coder, which gives very good compression results for tree source sequences, of arbitrary length. Good models are needed to obtain effective compression, and making the models general renders arithmetic coding universal. Rather than trying to come up with a general model, we simply use the CTW method, which considers all possible models. Thus, we make the H.264 arithmetic coder universal by replacing the pre-specified models with the CTW method. Note that practically the LZ method never returns asymptotically optimal compression result as many typical sources are non-stationary and data sequences are of arbitrary length. This is especially true with video sequences.

The compression part can be considered to be the same for all arithmetic coders even though different results can arise by using different precisions when representing internal data structure. In this paper we are only concerned with the modeling part. In particular we will focus on context modeling.

The next section briefly describes the CABAC and the CTW method. Then, Section 3 presents our approaches for incorporating the CTW method into the H.264 arithmetic coder. A novel approach for automatically adapting the CTW method based on the syntactic element to be coded and the encoding parameters is presented as well. We have
obtained coding results for a variety of sequences, and we give a detailed explanation of the results. Finally we
conclude with Section 4, summarizing the work presented in this paper.

2. BACKGROUND

We have modified an H.264 coder so that instead of using the context models defined in the standard, the arithmetic
coder uses the CTW method to derive probabilities of the symbols. Before giving detailed information about different
approaches to incorporating the CTW method into the H.264 coder, we give a brief background on both the arithmetic
coder (CABAC) used in H.264 and the CTW method.

2.1 CABAC (H.264)
The CABAC described in the H.264 standard (MPEG-4 Part 10) [14, 26, 27] can be divided into three major parts:
binarization, modeling and compression. First, the values of syntactic elements (e.g., macroblock type, motion vector
differential, transform coefficient level, etc.) are binarized using any one of four binarization schemes: unary, truncated
unary, concatenated unary/kth-order Exp-Golomb, and fixed-length (more information about these schemes can be
found in [14]). For example, using the unary binarization scheme, any non-negative number can be binarized as shown
in Table 1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Binarization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1 1</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Each bit (bin) of a binarized symbol (syntactic element) is denoted by its bin number (bin_no), which is the
position of the bin in the binary code used to represent the given symbol. Each bin_no is associated with at least one
context model, where each model corresponds to a unique probability distribution for the corresponding bin. For
example, when coding the motion vector differential (MVD) data, the context model of the first bin is chosen based on
previously coded MV values. There are three possible context models (probability distributions) for the first bin and a
context model is chosen based on the fact that the current MV will have similar magnitude as the ones corresponding to
the neighboring blocks. The remaining bins are coded using one of four other context models, depending on the bin_no
of the bins. The context models used for the MVD and transform coefficient level (TCL) data are listed in Table 21.

<table>
<thead>
<tr>
<th>bin_no</th>
<th>context models</th>
<th>MVR</th>
<th>context models</th>
<th>TCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0, 1, or 2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 and higher</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After a symbol is binarized, for each bin to be coded, the probability distribution is calculated from the
corresponding context model. For example, whenever the second bin of the binarized MVD data is being coded, the
current distribution corresponding to Context Model 3 is used, and once the bin is coded, the distribution gets updated.
Additionally, the model adapts to non-stationary characteristics of the source data by constantly rescaling the
distribution once certain number of symbols has been encountered.

1 The context models reflect the ones used in [14], and may not be in sync with the latest specification.
2.2 Context-Tree Weighting
Unlike the conventional modeling used in arithmetic coding, the context-tree weighting (CTW) method introduced in [23, 24, 25] does not try to continuously determine the relevant contexts to be used for a particular source, but rather it appropriately averages across all possible contexts for all possible context models. Here, as in the MPEG-4 shape coding, a context refers to a probability distribution and a context model refers to a set of contexts, whereas in H.264, a context model corresponds to a distribution. The CTW method is generally used as a sequential universal source coding procedure, and it can replace the context and adaptive models used in arithmetic coding. Moreover, it has been shown in [24] that the method is optimal in the sense that it achieves the Rissanen lower bound.

The CTW method assumes tree sources where the current symbol to be coded depends only on the most recent source symbols. So for finite memory (say $D$) tree sources, $D$ previous symbols are used to predict the current symbol. Note that the CTW method is based on the observation that if $P_1$ is a good coding distribution for Source 1 and $P_2$ is a good coding distribution for Source 2, then the weighted distribution $(P_1 + P_2)/2$ is a good distribution for both Sources 1 and 2. This weighting is applied to all context models whose memory is less than $D$. This is easily done by using a context tree (a binary tree for binary sources) with depth $D$. As a result, the computational complexity is linear with respect to $D$, and the memory requirement increases exponentially with $D$.

3. ADAPTIVE CONTEXT-TREE WEIGHTING
We have investigated two different ways to incorporating the CTW method into an H.264 coder, and they are described in the next two subsections. Then in the third subsection, we describe a novel approach for automatically adapting the CTW method based on the syntactic element to be coded and the encoding parameters. We have tested the CTW-based modifications and the experimental results are discussed as well.

3.1 H.264 Context Weighting
We first modified an H.264 coder (TML Version 8 [28]) so that rather than using a single context model to derive the probability distribution of the current symbol to be coded, all available probability distributions (corresponding to all available context models) are weighted and used for encoding. For instance, when encoding the TCL data with the modified coder, the distributions corresponding to the three context models (see Table 2) are weighted and used to derive the probability for each bin, instead of using one specific context model. Note that here we are restricted to the context models specified in the H.264 standard.

We have tested the above-mentioned modification with various sequences (three QCIF video sequences – “akiyo_qcif”, “foreman_qcif” and “coastguard_qcif” – and three CIF video sequences – “akiyo_cif”, “foreman_cif” and “stefan_cif”), and in all the cases, the modified coder yielded worse compression results than the original coder. This is due to the fact that we have mixed “bad” and “good” probability distributions. For example, for the MVD data, the context models specified in H.264 already take into account the known structure of the data: it is highly unlikely to have a small MVD value for a block whose neighboring blocks have a large MVD value. However, in our modification, we have mixed the distributions corresponding to both large and small MVD values when coding a small MVD value, and likewise with a large MVD value. Even for the TCL data, which are spatially uncorrelated symbols (due to the transformation coding), our modification yielded worse results. This is because the statistics of bins corresponding to different bit-planes are generally different.

3.2 H.264 Context-Tree Weighting
In our second modification of the H.264 coder, rather than restricting ourselves to the context models specified in the H.264 standard, we considered all possible context models during the weighting procedure. We have completely replaced the context models specified in the standard with generic context trees, assuming the data to be encoded are from finite-memory tree sources. This assumption is generally valid for the syntactic elements defined in the standard, as the next symbol probabilities have some dependence on a finite number of most recently processed symbols. For example, the macroblock type (MBT) data of a macroblock (MB) is highly correlated with the previous MBT data (of the previous MB). The same argument holds for the MVD, coded block pattern (CBP), and so on. In the case of the TCL, it is not obvious if similar dependency exists; however, we let the CTW method find the structure that may be inherent in the data. Note that given enough depth in the context tree, the CTW method can closely capture the structure of the data.
There are several ways to implement the second modification. One way is, similar to H.264, to use different context trees for different bins of each binarized syntactic element. However, this will require at least as many context trees as there are context models defined in the H.264 standard. Another way is to use one context tree for each syntactic element. This only requires 8 context trees for the 8 syntactic elements that are coded using the arithmetic coder. We chose the second option, and the results of using the modification are shown in Figures 2 to 6. Each figure contains three graphs comparing the results of the modification (we refer to the modified arithmetic coder – the coder that uses the CTW method instead of using a specific model – simply as the CTW-based coder) with the original CABAC at three different quantization parameter settings (QP = 24, 14, and 4). We used three QCIF video sequences – “akiyo_qcif” (aq), “foreman_qcif” (fq) and “coastgrd_qcif” (cq) – and three CIF video sequences – “akiyo_cif” (ac), “foreman_cif” (fc) and “stefan_cif” (sc). Each comparison lists the size of the original data after binarization, the size of the binarized data after coded by the CABAC, and the size of the binarized data after coded by the CTW-based coder. For example, for the MBT data extracted from the first three frames of the “stefan_cif” sequence, the size of the binarized data is 473 bytes. Entropy coding this data using the CABAC and reinitializing the coder at the beginning of each frame results in 355 bytes of data, whereas using the CTW-based coder results in 325 bytes of data. This is depicted Figure 2(a). From Figure 2, we can see that the modified coder does not always produce better results than the original coder. However, when a large sequence of the data are coded without reinitialization, as shown in Figures 3 to 6, the modified coder almost always produces better results; and for certain cases, the difference is significant. In Figure 3(c), we see that the MBT data (from the “stefan_cif” sequence) coded using the CTW-based coder is less than half the data coded using the CABAC.

Figure 2: The MBT data from 3 frames of each sequence (aq = akiyo_qcif, fq = foreman_qcif, cq = coastgrd_qcif, ac = akiyo_cif, fc = foreman_cif, sc = stefan_cif). For each sequence, the size of the data after binarization, the size of the data coded using the CABAC, and the size of the data coded using the CTW-based coder are compared. The coders are reinitialized at the beginning of each frame. The size is in bytes. (a) QP = 24. (b) QP = 14. (c) QP = 4.

Figure 3: The MBT data from 30 frames of each sequence where the arithmetic coder is not reinitialized. (a) QP = 24. (b) QP = 14. (c) QP = 4.

2 The 8 syntactic elements are: macroblock type, reference frame number, motion vector differential, delta quantization parameter, prediction modes, coded block pattern, transform coefficient level and run of zero coefficients. This reflects TML Version 8. More recent specification may include a different set of syntactic elements that are coded using an arithmetic coder.
3.3 H.264 Adaptive Context-Tree Weighting

The compression efficiency and complexity of the CTW method are highly dependent on the depth \( D \) of the context tree used – the results described in the previous subsection are obtained by using a context tree with \( D = 16 \). Generally, the method yields better results for bigger values of \( D \) until \( D \) reaches a certain value. This value of \( D \) can be called \( D_{\text{sat}} \) (saturation value). For a finite memory tree sequence, that value of \( D_{\text{sat}} \) will be the memory of the sequence. So, \( D_{\text{sat}} \) is the smallest value of the depth \( D \) for the context tree that includes the structure of the sequence. Therefore, we would want to set the value of \( D \) as large as possible, without going over the memory limit. But, the drawback of having the value of \( D \) too large is that it will decrease the coding speed. Larger depth means more updates in the CTW algorithm, so we may want to set the value of \( D \) just right. Also note that the memory requirement increases exponentially with increasing value of \( D \). Hence, given an upper bound on the decoding speed and an upper bound on the memory usage,
we can choose the largest value of $D$ satisfying both constraints. However, this is not an efficient way of choosing the value of $D$: once $D$ reaches a certain value, the compression gain obtained by further increasing the value of $D$ is negligible. In this subsection, we describe a way to automatically adapt the value of $D$, without sacrificing too much compression.

The values of syntax elements such as the MBT, the MVD and the CBP of current MB are highly correlated with those of previously coded MB. For the CTW method to capture the structure for the data, without wasting too much memory, the depth of the context tree (or the memory of the tree source) can be set to equal to the maximum codeword length of the binary code representing the syntactic element. For example, for the MBT data, if the context models are reinitialized at beginning of each frame, then when encoding an Intra frame, the depth $D$ of the corresponding context tree can be set to 6, and for the other frame types, $D$ can be set to 10. Note that we need at most 6 bits to represent the value of a MBT data when encoding in the Intra mode, and at most 10 bits are needed when encoding the data in the Inter mode. In the case where reinitialization is not required, one context tree with $D = 10$ can be used. Similarly, for coding the CBP data, we need a context tree with $D = 6$ for the same reason as previously described.

Unlike the MBT and the CBP, the size of a MVD value can be quite large, and the value depends on both the video sequence characteristics as well as the search range. As such, there is no simple way to derive a good value for $D$, where $D$ is of a reasonable size. However, after testing the CTW method with various values of $D$, we chose 8 as the value of $D$ when coding the MVD data. For each of the six sequences (aq, fq, cq, fc, sc), the MVD data from the first 30 frames were extracted and then using the CTW method (with different values of $D$), the size of the coded data is obtained. The coding results for the MVD data obtained with QP set to 24 are shown in Figure 7. For both QCIF and CIF sequences, as can be seen from the figure, the coding gain obtained by changing the value of $D$ from 8 to any value larger than 8 is negligible. Hence, $D_{sat}$ in this case, is 8. Similar results are obtained for QP = 14 and 4. Based on the results, we set the value of $D$ used when coding the MVD data to 8.

In the case of the TCL data, the number of bits needed to represent each element varies depending on the quantization parameter (QP). Generally, for bigger QP values, fewer bits are needed to represent each TCL. Consequently, we can adapt the CTW method by changing the depth of the context tree used based on the chosen parameter. For example, Figures 8 and 9 show how the compression result changes as different values of $D$ are used. For QP = 24, if any value larger than 6 is used for $D$, then the coding improvement, compared with using $D = 6$, is negligible. Hence, in this case, $D_{sat} = 6$. Also from Figure 9, for QP = 4, we see that $D_{sat} = 10$. This means $D$ is inversely related with the QP value; bigger QP value implies smaller $D$. This is as expected as the QP value increases, the variation of the TCL decreases, and the value of each TCL data also decreases. As a result, the structure of the data
can be described with fewer bits. In our case, we set $D = 6$ for $QP \geq 24$, $D = 8$ for $24 > QP \geq 14$, and $D = 10$ for $14 > QP$.

![Figure 8: The TCL data from 3 frames of each sequence where the arithmetic coder is reinitialized for each frame. The size of the data (in bytes) coded using the CTW method with various context tree depths ($D$) is depicted. (a) QCIF. (b) CIF.](image)

![Figure 9: The TCL data from 3 frames of each sequence where the arithmetic coder is reinitialized for each frame. The size of the data (in bytes) coded using the CTW method with various context tree depths ($D$) is depicted. (a) QCIF. (b) CIF.](image)

For the run data (the number of zeroes before a significant coefficient), which is binarized using the unary binarization scheme, the maximum number of bits needed to represent a value is 16 (15 zeroes). However, it is highly unlikely that the only significant coefficient of a 4x4 block corresponds to the highest frequency signal component. In fact, most of the significant coefficients will correspond to low frequency components. As a result, the run value will usually be significantly less than 15. For this reason, we chose $D = 10$, and using the CTW method yielded better results than using the H.264 context models. The CTW method produced about 1 to 2% less bits than the H.264 method when encoding the run data.

Note that there is not much correlation in the sequence of run values, and consequently, the latest H.264 standard [29] uses a different method when coding the transform coefficients. Instead of using run length coding, a significance map is used to specify the positions of significant coefficients. The TCL data is still coded the same way as described above.
Using the adaptive context tree weighting method described in this section, we generally get a total of 1% to 3% fewer bits than using the CABAC when encoding first 30 frames of each of the 6 video sequences described in this paper.

4. CONCLUDING REMARKS

The benefit of using the CTW method over a given set of contexts and context models is that the CTW method works well for all types of sources, whether the structure of the source is known or not. Additionally, this greatly simplifies the specification as well as the development process of the specification-compliant applications. For example, in the H.264 specification, there are more than 260 context models defined and keeping track of different context models for different types of syntactic elements is very tedious. By using the CTW method, defining such models can be avoided, making the standard less complex.

We have shown that the CTW-based arithmetic coding yields similar or even better results compared with the CABAC. The resulting compression efficiency and complexity of the CTW method are highly dependent on the depth $D$ of the context tree used. We showed that the CTW method generally yields better compression result for bigger values of $D$. However, the value of $D$ should not be too large because both the time and space complexities increase with increasing value of $D$. As described in [25], $D$ can be made infinite and this results in optimal redundancy behavior for all tree sources. Also, the storage complexity is upper-bounded by $2^T-1$ number of nodes, where $T$ is the length of the sequence. However, the computational complexity is upper-bounded by $T$, and this may not be acceptable for a large sequence of data. We have described a simple way to find an effective value of $D$ based on the syntactic element to be coded and the quantization parameter.

The sequence of each syntactic element is not a true tree source as the syntactic elements of a macroblock depends not only on those of the macroblock located at left of the current one, but also on those of the macroblocks located on top of the current one. Since the macroblocks are processed in a raster scan order, the syntactic elements of a macroblock at the left edge of the frame are preceded by those of the one at the right edge of the frame. Hence, those syntactic elements are not efficiently processed. Nevertheless, as shown in the previous section, the CTW method was able to capture whatever the structure that was available and produce coding results better than the CTW method. A better way to process the macroblocks, when using the CTW method, would be to encode them in a zigzag scanning order.

REFERENCES

17. ISO/IEC 11172 International Standard (MPEG-1), Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage at up to about 1.5 Mbit/s, 1993.