# Improvement of H.263 Encoding by Adaptive Arithmetic Coding

Eckhart Baum, Volker Harr, and Joachim Speidel

The authors are with the Institute of Telecommunications at the University of Stuttgart, 70569 Stuttgart, Germany.



IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 5, August 2000, pp. 797 - 800

**Abstract**

Arithmetic coding in H.263 is based on models that assign a fixed probability to each possible value of some syntax element. In this paper the effect of adapting the models according to the dynamically changing statistics is analyzed. Simulation results show improvements in all studied cases.

**Keywords**

Adaptive arithmetic coding, entropy coding, H.263, symbol frequencies, video coding.

## I. Introduction

Entropy coding is an effective method to compress data by reducing redundancy. Thus current video coding standards [1]–[4] make use of entropy coding, namely Huffman coding or arithmetic coding. In both cases compression is achieved by transmitting the more probable symbols with fewer bits than the less probable ones. Huffman coding uses variable length code words, that are assigned to each symbol. In contrast arithmetic coding applies models containing the distribution probabilities for each syntax element, e. g. motion vector. The set of all possible motion vectors (symbols) is referred to as syntax element "motion vector". A similar definition holds for the other syntax elements, like run level or type information in a H.263 coder. Instead of using fixed tables like in the above mentioned standards the tables can also be changed to follow the time variant statistics [5] [6].

This paper presents a method for adapting models in image coding. The algorithm has been applied to a software H.263 codec. For a simple realization the algorithm retains the H.263 syntax. Except for one global bit that signals the use of adaptation, no extra information has to be transmitted. The additional computation power is very small so the hardware complexity is not increased.

Section II gives an overview over arithmetic coding in H.263. In section III the proposed adaptation algorithm is described and section IV shows simulation results for different sequences.

## II. H.263 Encoder with Arithmetic Coding

In Fig. 1 the principal block diagram of a coder according to current standards [1]–[4] is shown. With inter frame coding the displaced frame difference is transformed using

discrete cosine transform (DCT) and quantized. This leads to blocks of DCT coefficients which are run-level coded. Together with the motion vectors and some control information they form a complete image representation. In order to reduce redundancy of data to be transmitted all syntax elements are entropy coded. In H.263 either Huffman coding (like in other standards, [1]–[3]) or syntax based arithmetic coding (SAC) can be used. Our study is based on the latter since it offers a higher performance than Huffman coding.

Fig. 1. Source coder according to ITU-T Rec. H.263, T: Transform (DCT), Q: Quantizer, FS: Framestore, Pred: Predictor, ME: Motion Estimator, C: Entropy Coder.

The complete algorithm of arithmetic coding is described in [6]. Instead of using VLC tables where each symbol corresponds to a predefined variable length code, the arithmetic coder uses cumulation tables. These tables contain the cumulated frequencies for all possible symbols of every syntax element. The frequencies are determined by statistical measurements and must be proportional to the corresponding event probability. One table for each syntax element represents its probability distribution. Based on this model the coder generates a bit stream for the incoming sequence of symbols. By doing so it can combine subsequent symbols and assign one code word to them. Therefore it is possible to spend a fractional number of bits for each symbol. Unlike Huffman coding, the arithmetic coder achieves a total number of bits equal to the entropy of the information source.

## III. Adaptive Arithmetic Coding

In H.263 fixed models for the arithmetic coder are used, that is all predefined symbol frequencies remain constant during encoding. This requires time invariant probability distributions, which hold for all successive transmitted symbols, to get optimum results. But in case the statistical characteristics change in time, the models have also to be varied. This study shows the possible improvements of dynamically adapted cumulation tables. As in low bitrate coding the portion of intra coded frames is much smaller than that of inter coded frames, only the latter ones will be further considered.

When designing an adaptive coder, two points must be taken into consideration. First the initial values of the SAC tables are to be defined. One possibility is to choose all frequencies equal, as if there were no a priori information about the probability distributions.

However this leads to a poor performance at the beginning of a sequence, as the presumed equal distributions does not match the actual distributions at all. Some frames later, after the tables have been adapted, the coding efficiency increases. Another approach is to use the fixed tables of the arithmetic coder specified in H.263 as initial values. So the coder starts with proper approximations for the real distributions and the time to adapt the tables decreases. Due to this fact we have chosen this method for our simulations.

The second question is how to adapt the SAC tables. Because transmitter and receiver must always have the same cumulation tables for an error free communication, the encoder may only use information for adaptation that is known also by the receiver. Otherwise additional updating information would have to be sent to the decoder at the receiver side. We are interested in a solution without additional overhead which maintains the bit stream syntax specified in H.263 for compatibility reasons. This requires the same initial tables in encoder and decoder as described above and an updating procedure that only relies on symbol frequencies derived from already transmitted data.

Whenever a certain symbol occurs, its measured frequency increases and thus the estimated probability changes. Because of that it seems reasonable to re-evaluate the cumulation tables after each transmitted symbol. Unfortunately, updating the model is quite expensive, as there are cumulative totals to be maintained. Since the occurence of one single syntax element contains only little information about its actual probability, there is no need to update immediately. Our investigations have shown that an update of the tables only once per frame is sufficient.

Let us assume now that both encoder and decoder have determined how often each syntax element has been set to certain values. Generally the amount of data is quite small during one frame in a statistical sense. Thus generating completely new tables only based on this information would cause a bad statistical prediction for the next frame. A better solution is to include the previous frequency tables into the computation. Both old and new information can be weighted in a suitable manner. In other words, for each possible value of some syntax element the old and new counted frequencies have to be added with a certain weighting factor $w$. The results for all values will then be used to create a new SAC table.

The choice of $w$ corresponding to one syntax element respectively its frequency table needs some considerations. Suppose frame $i$ has just been transmitted using a SAC table $i-1$ based on a total number of $N_{i-1}$ symbols. In this table each value $v \in \mathbf{V}$ has a frequency $n_{i-1,v}$, with $N_{i-1} = \sum_{v \in \mathbf{V}} n_{i-1,v}$, where $\mathbf{V}$ is the set of all possibles values for this syntax element. To generate the new SAC table $i$, frame $i$ provides an additional amount of $K_i$ symbols, where each value $v$ has occured $k_{i,v}$ times ($K_i = \sum_{v \in \mathbf{V}} k_{i,v}$). To compute the prediction $n_{i,v}$ for the next frame $i+1$ we use the following formulation:

$$n_{i,v} = \frac{w n_{i-1,v} + k_{i,v}}{r_i} \tag{1}$$

where $w$ weights the previous information. There is no need to have a separate weight for $k_{i,v}$ because of the two parameters $w$ and $r_i$. $r_i$ is a scaling parameter to be chosen such that the total number $N_i$ equals to a fixed number $N$ for all frames $i$ to avoid permanent growing or shrinking of $n_{i,v}$ from frame to frame. This is no restriction, because multiplying all $n_{i,v}$ by a factor neither changes the probability distribution nor consequently the number of bits per symbol. Summing up $n_{i,v}$ for all possible values $v \in \mathbf{V}$ in Eq. (1) and setting $N = N_i = N_{i-1}$ leads to

$$r_i = w + \frac{K_i}{N} \tag{2}$$

which depends on the total number $K_i$ of symbols transmitted in frame $i$ and changes after every frame. With Eq. (2) we obtain from Eq. (1)

$$n_{i,v} = \frac{w n_{i-1,v} + k_{i,v}}{w + \frac{1}{N} K_i}. \tag{3}$$

Obviously, the denominator in Eq. (3) can be considered as a scaling factor for $n_{i,v}$, which can be varied by $N$. If $N = 1$, the $n_{i,v}$ represent the measured probabilities, as $\sum_{v \in \mathbf{V}} n_{i,v} = 1$ holds. The weight $w$ ($0 \leq w < \infty$) for one syntax element, and accordingly all weights for the other syntax elements that can also be considered as "forgetting factors", are important for optimizing the adaptation. For the two special cases, $w = 0$ (the adaptation totally "forgets" any information of previous frames) and $w \to \infty$ (the adaptation does not "forget" anything), we obtain from Eq. (3)

$$n_{i,v} = k_{i,v}\frac{N}{K_i} \qquad (w = 0) \tag{4}$$

$$n_{i,v} = n_{i-1,v} \qquad (w \to \infty). \tag{5}$$

As can be seen from Eq. (4), the prediction for frame $i+1$ is based only on the transmitted symbols of frame $i$ and thus it may not always be reliable. On the other hand Eq. (5) shows, that the frequencies $n_{i,v}$ do not change at all for $w \to \infty$, therefore no adaptation takes place. Obviously, an optimum $w$ must lie in between these two bounds.

Optimization of $w$ can be done once and off-line by performing computer simulations with a representative set of different video sequences. By varying $w$ step by step and measuring the resulting bitrates for all video sequences, a value $w_o$ can be found that is nearly optimum for each sequence. Simulations have shown, that small variations of $w_o$ do not have much effect on coding efficiency, so optimization is not critical. As the different syntax elements are entropy coded with independent SAC tables, an optimum value of $w$ can be allocated for each syntax element. Once the optimum $w$ for each SAC table has been found it is used for all video transmissions without change. As described in Section IV, these values are 0.1 and 0.2, resp., depending on type of syntax element. An optimization of $w$ during transmission would give only slight improvements. However, it would be expensive and is therefore not recommended for a real time video codec.

Problems can occur, if the received bitstream contains an error and the receiver decodes a wrong symbol. Apart of the fact that this will likely cause additional failures due to variable length coding, the decoder will use wrong information to update its SAC tables. This can lead to different tables at the coder and decoder which will have tremendous consequences for the subsequent decoding process. To reduce this effect of different tables it is recommended to reset the adaptation process from time to time by setting all tables to the initial values. This can be done by setting $w$ temporarily equal to 0 in both the coder and decoder. A simple solution for a real codec is to reset $w$ periodically.

## IV. Simulation Results

Simulations have been carried out with the H.263 encoder from Telenor, version 2.0, which has been extended by the described framewise adaptation of SAC tables. For an

accurate comparison of normal SAC and adaptive SAC the rate control is turned off. In this case changing the entropy coding procedure has no effect on the resulting picture quality and the peak signal to noise ratio (PSNR) at all. So bitrates can be compared under equal conditions. The frame rate is held constant at 7.5 frames per second. All special modes defined in ITU Rec. H.263 are turned on, namely unrestricted motion vector mode, syntax-based arithmetic coding, advanced prediction mode and PB-frames mode.

The results for sequences *Claire*, *Carphone*, *Foreman*, *Miss America*, *Salesman* and *Trevor* with normal SAC and adaptive SAC are provided in Tab. I. An improvement is always observable which goes up to 3.5%. The adaptation gets more efficient with higher bitrates. All simulations have set parameters $w$ to 0.1 for each syntax element (COD, MCBPC, MODB, YCBPB, UVCBPB, CBPY, DQUANT, MVD, TCOEF1, TCOEF2, TCOEF3, SIGN, LAST, RUN, LEVEL) except for TCOEFr, whose $w$ is set to 0.2.

TABLE I

OVERALL BITRATES FOR NORMAL SAC (H.263) AND ADAPTIVE SAC.

| Sequence | normal SAC (kbit/s) | adaptive SAC (kbit/s) | bitrate reduction for given PSNR | PSNR (dB) |
|---|---|---|---|---|
| Claire | 18.6 | 18.4 | 1.1 % | 38.8 |
| Carphone | 50.7 | 49.7 | 2.0 % | 33.9 |
| Foreman | 59.4 | 57.7 | 2.9 % | 33.3 |
| Miss America | 16.7 | 16.5 | 1.2 % | 39.5 |
| Salesman | 48.5 | 46.8 | 3.5 % | 37.0 |
| Trevor | 56.6 | 55.4 | 2.1 % | 35.1 |

Fig. 2 shows the number of bits per picture for the first 150 frames of sequence *Salesman*. It can be seen, that the improvement by adaptive SAC is comparable to the difference between normal SAC and Huffman coding. In addition there is no frame, where adaptive coding is outperformed by another mode. This holds for all other sequences, too, even for sequences with a scene change, like *Trevor*.

Fig. 2. Number of bits per frame versus frame number for *Salesman* sequence encoded at 48 kbit/s.

The influence of weighting factor $w$ is shown in Fig. 3. The curve plotted with squares represents the case, where all syntax elements have parameter $w$ set to 0, that is each SAC table is generated only based on the information of the previous frame. The second curve with circles holds for $w \to \infty$. In this case all tables remain constant and they are not adapted. By setting $w$ to the optimized values (0.2 for TCOEFr and 0.1 for other syntax elements, as listed above), the results, drawn with full dots, become superior to both other cases in almost all frames.

Fig. 3. Number of bits per frame versus frame number for *Foreman* sequence at 58 kbit/s.

## V. Conclusion

In this paper a method for adapting the cumulation tables in H.263 arithmetic coding is presented. After each frame the tables are re-evaluated based on the measured statistics of preceding frames, that have already been transmitted to the receiver. Therefore coder and decoder can synchronously update their tables, without transmitting additional information. So H.263 syntax needs just another flag that signals the use of adaptive arithmetic coding. As adaptation takes place only once per frame, the computational amount is quite small. Simulations of the adaptive encoding have shown a bitrate reduction of up to 3.5 % at the same PSNR. Adaptive encoding has never been inferior to normal H.263 coding.

## References

[1] ISO/IEC JTC1/SC29/WG11, *Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s: Video*, November, 1991.

[2] ISO/IEC JTC1/SC29/WG11/N802, ITU-T H.262, Draft International Standard *Generic coding of moving pictures and associated audio information: Video*, March, 1995.

[3] ITU-T Recommendation H.261, *Video codec for audiovisual services at p x 64 kbit/s*, December, 1990.

[4] Draft ITU-T Recommendation H.263, *Video Coding for Low Bitrate Communication*, May, 1996.

[5] L.-Y. Liu, J.-F. Wang, and J.-Y. Lee, *On the concurrent update and generation of the dynamic Huffman code*, IEEE Trans. Signal Proc., vol. 44, pp. 2082–2085, August 1996.

[6] I. H. Witten, R. M. Neal, and J. G. Cleary, *Arithmetic coding for data compression*, Commun. ACM, vol. 30, pp. 520–540, June 1987.

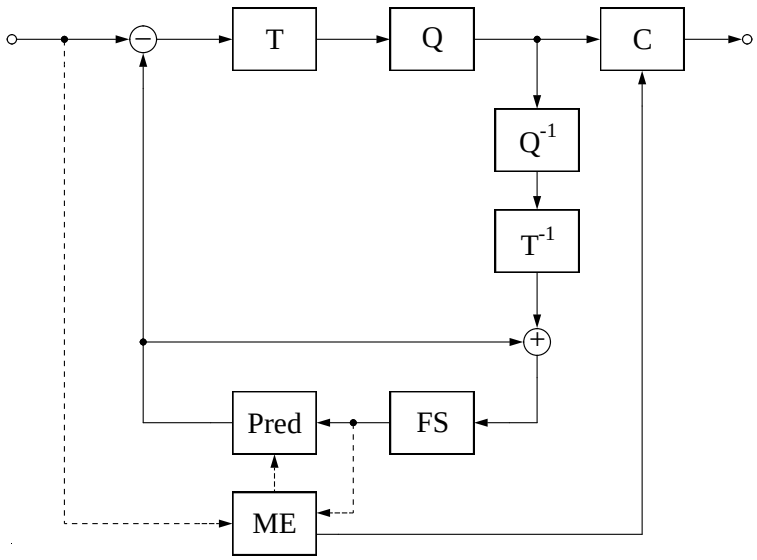


Fig. 1. Source coder according to ITU-T Rec. H.263, T: Transform (DCT), Q: Quantizer, FS: Framestore, Pred: Predictor, ME: Motion Estimator, C: Entropy Coder.

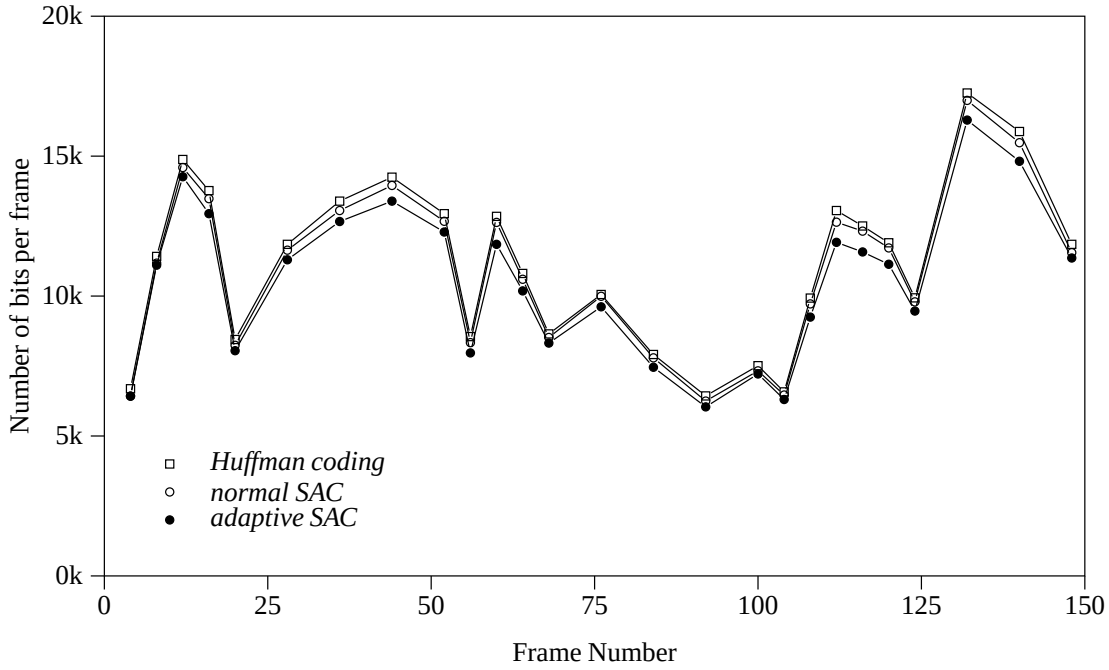


Fig. 2. Number of bits per frame versus frame number for *Salesman* sequence encoded at 48 kbit/s.

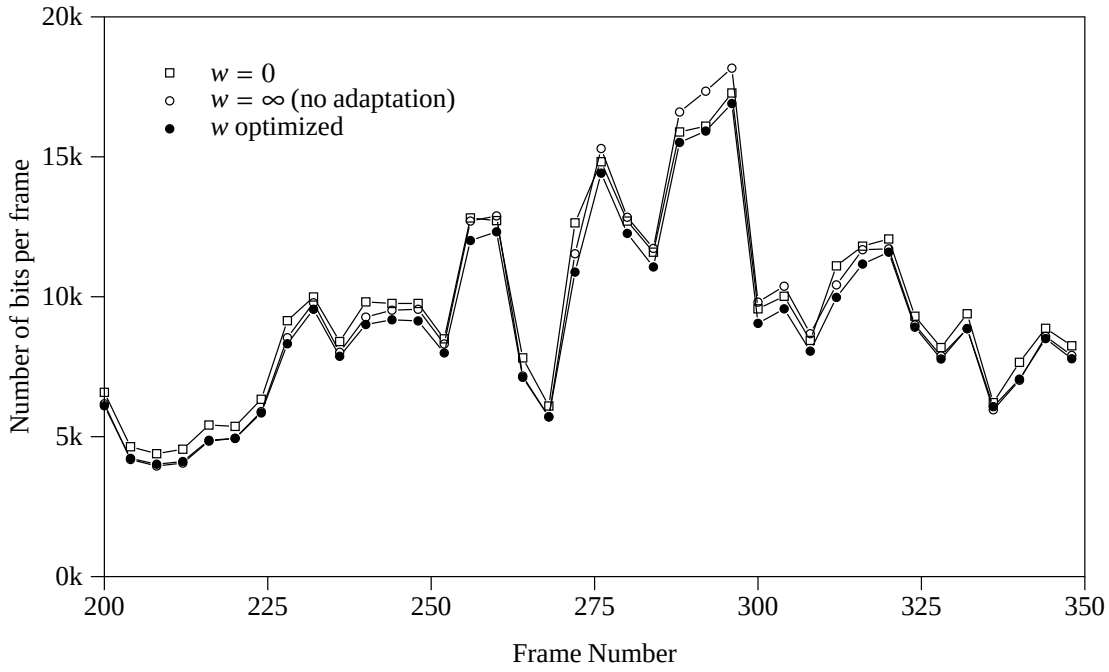


Fig. 3. Number of bits per frame versus frame number for *Foreman* sequence at 58 kbit/s.