An Adaptive Multialphabet Arithmetic Coding Based on Generalized Virtual Sliding Window

Belyaev, Evgeny; Forchhammer, Søren; Liu, Kai

Published in:
I E E E Signal Processing Letters

Link to article, DOI:
10.1109/LSP.2017.2705250

Publication date:
2017

Document Version
Peer reviewed version

An adaptive multi-alphabet arithmetic coding based on generalized virtual sliding window

Evgeny Belyaev, Søren Forchhammer and Kai Liu

***Abstract*—We propose a novel efficient multi-alphabet multiplication-free adaptive arithmetic coder. First, we generalize probability estimation via virtual sliding window for the multi-alphabet case and show that it does not require multiplications and provides a trade-off between the probability adaptation speed and the precision of the probability estimation. Second, we show how the generalized virtual sliding window can be used to eliminate multiplications and divisions. Finally, we demonstrate that the proposed arithmetic coder provides better compression performance than existing implementations based on state-of-the-art multiplication-free binary arithmetic coders.**

***Index Terms*—multi-alphabet arithmetic coding, probability estimation**

I. Introduction

Adaptive multi-alphabet arithmetic coding is a key component of many data compression algorithms. The well-known integer implementation of the arithmetic coding, proposed by Witten et al. [16], requires multiplications and divisions. This makes it difficult to use the coder, especially in hardware. Therefore, the vast majority of existing implementations represent non-binary data as a set of binary symbols (called binarization) and compress them using context modeling and a multiplication-free adaptive binary arithmetic coding [1], [2], [3], [4], [5], [6], [7]. However, this approach can provide less compression performance comparing to multi-alphabet arithmetic coding [1].

In this letter, we present a novel efficient multi-alphabet multiplication-free adaptive arithmetic coder. The main contributions of the letter are the following:

1) We generalize the probability estimation via virtual sliding window [11] into multi-alphabet case and show that it does not require multiplications and provides a trade-off between the probability adaptation speed and the precision of the probability estimation.
2) We show how the generalized virtual sliding window can be used to eliminate multiplications and divisions.
3) We demonstrate that the proposed arithmetic coder provides better compression performance comparing to implementations based on binary arithmetic coders from JPEG2000, H.264/AVC and H.265/HEVC standards.

Evgeny Belyaev and Søren Forchhammer are with the Department of Photonics Engineering, Technical University of Denmark, e-mails: {evbel,sofo}@fotonik.dtu.dk

Kai Liu is with the School of Computer Science and Technology, Xidian University, China, e-mail: kailiu@mail.xidian.edu.cn

This work was supported in part by the National Natural Science Foundation of China under Grants 61550110247, 91538101 and 61571345.

The rest of the letter is organized as follows. Section II reviews multi-alphabet arithmetic coding and its integer implementation. Section III introduces the proposed arithmetic coding implementation. Performance evaluation and conclusions are drawn in Sections IV and V.

II. Arithmetic Coding and Its Implementation

A. General description

Let us consider a discrete stationary memoryless source generating letters $\mathbf{x}^N = \{x_1, x_2, ..., x_N\}$ from alphabet $\mathbf{a} = \{a_1, ..., a_M\}$ with probabilities $p(a_1), ..., p(a_M)$. In arithmetic coding, a codeword for $\mathbf{x}^N$ is represented as the first $\lceil -\log_2 p(\mathbf{x}^N) + 1 \rceil$ bits in binary representation of $q(\mathbf{x}^N) + p(\mathbf{x}^N)/2$, where $p(\mathbf{x}^N)$ is the probability of $\mathbf{x}^N$ written as

$$p(\mathbf{x}^N) = \prod_{i=1}^{N} p(x_i), \tag{1}$$

and $q(\mathbf{x}^N)$ is cumulative probability of $\mathbf{x}^N$ written as

$$q(\mathbf{x}^N) = \sum_{\mathbf{y}^N \prec \mathbf{x}^N} p(\mathbf{y}^N), \tag{2}$$

where $\mathbf{y}^N \prec \mathbf{x}^N$ means that sequence $\mathbf{y}^N$ is preceding $\mathbf{x}^N$ in lexicographic order. Both $p(\mathbf{x}^N)$ and $q(\mathbf{x}^N)$ can be calculated using the following iterative equations:

$$\begin{cases} p(\mathbf{x}^i) = p(\mathbf{x}^{i-1}) p(x_i), \\ q(\mathbf{x}^i) = q(\mathbf{x}^{i-1}) + p(\mathbf{x}^{i-1}) q(x_i), \end{cases} \tag{3}$$

where $p(\mathbf{x}^0) = 1$, $q(\mathbf{x}^0) = 0$ and $q(x_i)$ is the cumulative probability of letter x_i.

Let us define $\hat{f}(\mathbf{x}^N)$ as a value of the first $\lceil -\log_2 p(\mathbf{x}^N) + 1 \rceil$ bits in binary representation of $q(\mathbf{x}^N) + p(\mathbf{x}^N)/2$. Then the arithmetic decoder iteratively determines the decoded letter as $x_i = a_j$, where j is the minimum possible value satisfying

$$p(\mathbf{x}^{i-1}) q(a_j) + q(\mathbf{x}^{i-1}) \geq \hat{f}(\mathbf{x}^N). \tag{4}$$

B. Integer implementation of encoder

In case of adaptive coding, probabilities $p(a_1)$,...,$p(a_M)$ are unknown and should be estimated [9]. Let us consider the scaled counters algorithm [16], where the probability is estimated as $\hat{p}(a_k) = \frac{C_k}{\mathcal{C}}$, where C_k is counter of letter a_k and $\mathcal{C} = \sum_{m=1}^{M} C_m$. In order to avoid the counters overflowing, when $\mathcal{C}$ exceeds $\mathcal{C}_{max}$ the counters are reduced two times (scaled). Finally, registers $Q_1, ..., Q_M$ are used to calculate the cumulative probabilities (see Algorithm 1)[1].

[1]In this letter we use "$\leftarrow$" as the assignment operation, "$\ll$" and "$\gg$" as left and right arithmetic shift.

Algorithm 1 : Probability estimation procedure

$k \leftarrow j$, where j satisfies $a_j = x_i$.
$C_k \leftarrow C_k + 1$
$\mathcal{C} \leftarrow \mathcal{C} + 1$
if $\mathcal{C} > \mathcal{C}_{max}$ **then**
  $\mathcal{C} \leftarrow 0$
  for $m = 1, ..., M$ **do**
    $C_m \leftarrow \max\{1, \lfloor C_m/2 \rfloor\}$
    $\mathcal{C} \leftarrow \mathcal{C} + C_m$
  end for
end if
$Q_1 \leftarrow 0$
for $m = 2, ..., M$ **do**
  $Q_m \leftarrow Q_{m-1} + C_{m-1}$
end for

An integer implementation of an arithmetic encoder engine is based on two registers: L and R of size b bits. Register L corresponds to $q(\mathbf{x}^i)$ and register R corresponds to $p(\mathbf{x}^i)$. The precision required to represent registers L and R grows with increasing of N. In order to decrease the coding latency and avoid register underflow, the renormalization procedure [16] is used for each output symbol (see Algorithm 2). Here procedures *WriteZeros*(z) and *WriteOnes*(z) write z zeros or ones into the output bit stream, respectively.

Algorithm 2 : Encoder renormalization procedure

while $R < 2^{b-2}$ **do**
  if $L \geq 2^{b-1}$ **then**
    WriteOnes(1)
    WriteZeros($bits\_to\_follow$), $bits\_to\_follow \leftarrow 0$
    $L \leftarrow L - 2^{b-1}$
  else if $L < 2^{b-2}$ **then**
    WriteZeros(1)
    WriteOnes($bits\_to\_follow$), $bits\_to\_follow \leftarrow 0$
  else
    $bits\_to\_follow \leftarrow bits\_to\_follow + 1$
    $L \leftarrow L - 2^{b-2}$
  end if
  $L \leftarrow L \ll 1$, $R \leftarrow R \ll 1$
end while

Finally, an integer implementation of arithmetic encoding is given by Algorithm 3. Notice, a file header containing a number of encoded symbols is used for correct bit stream termination at the decoder side.

Algorithm 3 : Symbol x_i encoding procedure

$k \leftarrow j$, where j satisfies $a_j = x_i$.
$L \leftarrow L + \left\lfloor \frac{RQ_k}{\mathcal{C}} \right\rfloor$
$R \leftarrow \left\lfloor \frac{RC_k}{\mathcal{C}} \right\rfloor$
call Probability estimation procedure via **Algorithm 1**
call Encoder renormalization procedure via **Algorithm 2**

C. Integer implementation of decoder

An integer implementation of an arithmetic decoder engine utilizes additional register F corresponding to $f(\mathbf{x}^N)$. Register F consists of b bits which are enough to determine the decoded letter x_i in (4), and updated in the renormalization procedure given by Algorithm 4. Here procedure $ReadBit()$ reads one bit from the bit stream.

Algorithm 4 : Decoder renormalization procedure

while $R < 2^{b-2}$ **do**
  if $L \geq 2^{b-1}$ **then**
    $L \leftarrow L - 2^{b-1}$
    $F \leftarrow F - 2^{b-1}$
  else if $L \geq 2^{b-2}$ **then**
    $L \leftarrow L - 2^{b-2}$
    $F \leftarrow F - 2^{b-2}$
  end if
  $L \leftarrow L \ll 1$, $R \leftarrow R \ll 1$
  $F \leftarrow (F \ll 1) + ReadBit()$
end while

An integer implementation of arithmetic decoding is given by Algorithm 5. Here the probability estimation procedure and update for registers L and R are the same as for the encoder. For more details towards to the implementation see [16].

Algorithm 5 : Symbol x_i decoding procedure

$\mathcal{Q} \leftarrow \left\lfloor \frac{(F - L + 1)\mathcal{C} - 1}{R} \right\rfloor$
for $m = 1, ..., M$ **do**
  if $Q_m > \mathcal{Q}$ **then break**
  $x_i \leftarrow a_m$
end for
$k \leftarrow j$, where j satisfies $a_j = x_i$.
$L \leftarrow L + \left\lfloor \frac{RQ_k}{\mathcal{C}} \right\rfloor$
$R \leftarrow \left\lfloor \frac{RC_k}{\mathcal{C}} \right\rfloor$
call Probability estimation procedure via **Algorithm 1**
call Decoder renormalization procedure via **Algorithm 4**

III. Proposed arithmetic coding

A. Probability estimation via virtual sliding window

In [11], [12] a probability estimation for a binary source based on *virtual sliding window* was proposed. In this algorithm the probability that symbol x_i is equal to one is estimated as

$$\hat{p}(1) = \frac{S}{2^{2W}}, \tag{5}$$

where S is the window state updated by the following rule:

$$S \leftarrow \begin{cases} S + \left\lfloor \dfrac{2^{2W} - S + 2^{W-1}}{2^W} \right\rfloor, \text{ if } x_i = 1. \\ S - \left\lfloor \dfrac{S + 2^{W-1}}{2^W} \right\rfloor, \text{ if } x_i = 0, \end{cases} \tag{6}$$

where 2^W is the window length, and 2^{2W} is the maximum possible value of state S. Notice that division by 2^W can be

implemented via right arithmetic shift, i.e., $\lfloor x/2^W \rfloor$ is equal to $x \gg W$.

In this letter we propose to *generalize* this probability estimation approach to a multi-alphabet case in the following way. Let us utilize M virtual sliding windows, where window $m \in \{1, .., M\}$ estimates the probabilities of two possibilities: next symbol $x_i = a_m$ or next symbol $x_i \neq a_m$. Then the probability that $x_i = a_m$ is estimated as

$$\hat{p}(a_m) = \frac{S_m}{M2^{2W}}, \tag{7}$$

where states $S_1, ..., S_M$ are updated in two steps. At the first step, all states of windows which are not related with the current symbol x_i are updated as

$$S_m \leftarrow S_m - \left\lfloor \frac{S_m + 2^{W-1}}{2^W} \right\rfloor, \text{ for } \forall m : a_m \neq x_i, \tag{8}$$

and at the second step the state related with the current symbol x_i is updated as

$$S_m \leftarrow M2^{2W} - \sum_{k=1, k\neq m}^{M} S_k, \text{ for } m : a_m = x_i. \tag{9}$$

Utilizing (8) and (9) the proposed probability estimation procedure is given by Algorithm 6. Here the initial states correspond to equal probabilities for all symbols, i.e., $S_1 = ... = S_M = 2^{2W}$.

Algorithm 6 : Probability estimation procedure

1: $k \leftarrow j$, where j satisfies $a_j = x_i$.
2: **for** $m = 1, ..., M$ **do**
3: $\quad S_m \leftarrow S_m - (S_m + 2^{W-1}) \gg W$
4: **end for**
5: $S_k \leftarrow M2^{2W} - \sum_{m=1}^{M} S_m + S_k$
6: $Q_1 \leftarrow 0$
7: **for** $m = 2, ..., M$ **do**
8: $\quad Q_m \leftarrow Q_{m-1} + S_{m-1}$
9: **end for**

Algorithm 6 does not use multiplications, divisions or conditional operations. A trade-off between the probability adaptation speed and the precision of the probability estimation can be easily achieved by selecting a specific W.

B. Multiplication-free encoder and decoder

After renormalization in Algorithm 2 and Algorithm 4, register R satisfies the following inequality [13]:

$$\frac{1}{2}2^{b-1} \leq R < 2^{b-1}. \tag{10}$$

From (10) it follows that a multiplication $R \times \hat{p}(a_m)$ can be approximated in the following way:

$$R \times \hat{p}(a_m) \approx \alpha 2^{b-1} \times \hat{p}(a_m) = \alpha 2^{b-1} \times \frac{S_m}{M2^{2W}}, \tag{11}$$

where $\alpha \in [(1/2), ...1)$. Let us quantize the interval $[\frac{1}{2}2^{b-1}, 2^{b-1})$ into 2^K sub-intervals. Then the sub-interval index is determined as

$$\Delta = \left\lfloor \frac{R - 2^{b-2}}{2^{b-2-K}} \right\rfloor, \tag{12}$$

and multiplication (11) is approximated as

$$R \times \hat{p}(a_m) \approx \left(2^{b-2} + \Delta 2^{b-2-K}\right) \times \frac{S_m}{M2^{2W}}. \tag{13}$$

Let us assume that the number of letters in alphabet **a** is a power of two, i.e., $M = 2^d$. Then if the registers size is

$$b = 2W + d + 2, \tag{14}$$

then (13) is rewritten as

$$R \times \hat{p}(a_m) \approx S_m + \left\lfloor \frac{\Delta \times S_m}{2^K} \right\rfloor. \tag{15}$$

From (14) follows that for 32-bit arithmetic and alphabet size of $M = 256$ the window length 2^W should not exceed 2^{11}. Repeating the reasoning described above, a multiplication $R \times \hat{q}(a_m)$ is approximated as

$$R \times \hat{q}(a_m) \approx Q_m + \left\lfloor \frac{\Delta \times Q_m}{2^K} \right\rfloor. \tag{16}$$

Thus, the proposed implementation of arithmetic encoding is given by Algorithm 7.

Algorithm 7 : Symbol x_i encoding procedure

1: $k \leftarrow j$, where j satisfies $a_j = x_i$.
2: $\Delta \leftarrow (R - 2^{b-2}) \gg (b - 2 - K)$
3: $L \leftarrow L + Q_k + (\Delta \times Q_k) \gg K$
4: $R \leftarrow S_k + (\Delta \times S_k) \gg K$
5: *call* Probability estimation procedure via **Algorithm 6**
6: *call* Encoder renormalization procedure via **Algorithm 2**

The corresponding arithmetic decoding is given by Algorithm 8.

Algorithm 8 : Symbol x_i decoding procedure

1: $\Delta \leftarrow (R - 2^{b-2}) \gg (b - 2 - K)$
2: **for** $m = 1, ..., M$ **do**
3: $\quad \mathcal{Q} \leftarrow Q_m + (\Delta \times Q_m) \gg K$
4: $\quad$ **if** $\mathcal{Q} > F - L$ **then break**
5: $\quad x_i \leftarrow a_m$
6: **end for**
7: $k \leftarrow j$, where j satisfies $a_j = x_i$.
8: $L \leftarrow L + Q_k + (\Delta \times Q_k) \gg K$
9: $R \leftarrow S_k + (\Delta \times S_k) \gg K$
10: *call* Probability estimation procedure via **Algorithm 6**
11: *call* Decoder renormalization procedure via **Algorithm 4**

One can see that the proposed implementation does not require divisions. Moreover, if K is small, then multiplications $\Delta \times S_k$ and $\Delta \times Q_k$ can be replaced by conditional, addition and shift operations. For example, if $K = 2$ then Δ can be 0, 1, 2 or 3 and $\Delta \times S_k$ is equal to 0, S_k, $S_k \ll 1$ or $S_k \ll 1 + S_k$, respectively.

Thus, the proposed approach provides multiplication-free implementation with a trade-off between implementation cost and compression efficiency by selecting a specific K. On one hand this is attractive for hardware implementations. On the other hand, for a given K, software implementations could apply the multiplication directly keeping bit stream compatibility with multiplication-free implementations.

TABLE I
COMPRESSION RATE FOR DIFFERENT AC IMPLEMENTATIONS FOR LARGE CALGARY CORPUS [14] DATA SET, BITS PER SYMBOL

File name	Witten and Cleary [16]	Unary binarization and M-coder	Unary binarization and MQ-coder	Tree binarization and M-coder	Tree binarization and MQ-coder	Proposed 1, $K = 8$	Proposed 2, $K = 8$	Proposed 2, $K = 5$	Proposed 2, $K = 3$	Proposed 2, $K = 1$
BIB	**5.24**	6.98	5.59	5.49	5.39	**5.24**	5.27	5.28	5.33	5.53
BOOK1	4.55	6.83	4.81	4.76	4.69	**4.54**	4.57	4.58	4.63	4.84
BOOK2	4.78	6.88	4.96	4.90	4.81	**4.71**	4.72	4.73	4.78	4.96
GEO	5.67	6.89	6.00	5.78	**5.65**	5.68	5.82	5.83	5.88	6.04
NEWS	5.19	6.86	5.32	5.24	5.15	**5.12**	**5.12**	5.13	5.18	5.37
OBJ1	5.98	6.90	5.59	5.51	**5.43**	5.59	5.59	5.60	5.65	5.85
OBJ2	6.08	7.47	6.09	5.95	**5.84**	5.86	5.86	5.87	5.92	6.11
PAPER1	4.99	6.94	5.17	5.07	5.00	**4.90**	**4.90**	4.91	4.96	5.15
PAPER2	4.63	6.93	4.92	4.83	4.76	**4.62**	4.63	4.65	4.69	4.88
PAPER3	**4.71**	6.98	4.99	4.91	4.84	**4.71**	**4.71**	4.72	4.77	4.95
PAPER4	4.82	7.02	5.08	4.94	4.93	**4.75**	**4.75**	4.76	4.81	5.00
PAPER5	5.06	7.01	5.26	5.14	5.13	**4.94**	4.95	4.96	5.01	5.21
PAPER6	5.01	6.79	5.05	4.99	4.94	**4.83**	**4.83**	4.84	4.89	5.09
PIC	1.17	1.44	**1.00**	1.03	1.10	1.08	1.08	1.09	1.15	1.46
PROGC	5.24	6.73	5.36	5.29	5.24	**5.14**	**5.14**	5.15	5.20	5.40
PROGL	4.76	6.36	4.75	4.70	4.66	**4.63**	4.64	4.65	4.70	4.90
PROGP	4.90	6.41	4.96	4.88	4.86	**4.76**	**4.76**	4.77	4.82	5.01
TRANS	5.50	6.54	5.44	5.38	**5.30**	**5.30**	5.31	5.32	5.37	5.57
Average	0.0%	+33.9%	+1.8%	+0.3%	-0.7%	-2.3%	-2.0%	-1.7%	-0.5%	+4.5%

TABLE II
COMPRESSION RATE FOR DIFFERENT AC IMPLEMENTATIONS FOR THE LARGE CORPUS [15] DATA SET, BITS PER SYMBOL

File name	Witten and Cleary [16]	Unary binarization and M-coder	Unary binarization and MQ-coder	Tree binarization and M-coder	Tree binarization and MQ-coder	Proposed 1, $K = 8$	Proposed 2, $K = 8$	Proposed 2, $K = 5$	Proposed 2, $K = 3$	Proposed 2, $K = 1$
bible.txt	4.35	5.87	6.69	4.54	4.50	**4.33**	4.36	4.37	4.42	4.61
world192	5.00	6.31	6.73	5.14	5.07	**4.96**	4.97	4.98	5.03	5.24
E.coli	2.03	3.73	5.00	2.07	2.20	**2.00**	**2.00**	2.01	2.06	2.24
Average	0.0%	+48.2%	+78.2%	+3.0%	+4.4%	-1.0%	-0.7%	-0.3%	+1.2%	+7.0%

TABLE III
AN AVERAGE NUMBER OF HARDWARE-CRITICAL OPERATIONS PER SYMBOL NEEDED FOR ENCODING AND DECODING

Operation	Witten and Cleary [16]	Unary binarization and M/MQ-coder	Tree binarization and M/MQ-coder	Proposed 2
Multip.	5	0	0	0
Division	5	0	0	0
Renorm.	2	186	16	2

IV. PERFORMANCE EVALUATION

For comparisons, we used multi-alphabet adaptive arithmetic coding implementation described in Section II. This implementation is a slightly modified version of arithmetic coding from [16]. We also compare the proposed coder with four multi-alphabet adaptive arithmetic coding based on adaptive binary arithmetic coding. As an adaptive arithmetic coding engine we used two state-of-the-art multiplication-free adaptive binary arithmetic coders: MQ-coder from JPEG2000 image coding standard [8] and M-coder [10] from H.264/AVC [6] and H.265/HEVC [7] video coding standards. For representing non-binary data into binary format we applied unary binarization as in standards [6], [7] and tree binarization introduced in [1]. The proposed coder was realized in two versions. The first version, denoted as Proposed 1, is the case, when the best window length was selected from set $2^W = \{2^6, 2^7, ..., 2^{11}\}$ for each file, while Proposed 2 utilizes $W = 9$ for all files.

The compression performance has been evaluated utilizing two data sets: Large Calgary Corpus [14] (see Table I) and The Large Corpus [15] (see Table II). The first data set consist of files with different statistical properties and lengths, while the second one is used for testing of compression for files with large size. Here we use the coder from [16] as reference coder and compare it with the multiplication-free implementations[2]. The presented results shows that:

1) Proposed 1 and Proposed 2 with $K = 8$ provide the best average results among considered coders. Herewith, Proposed 1 outperforms Proposed 2 with the price of multiple encoding with different window lengths, i.e., it can be used to achieve the maximum compression ratio at high performance platforms.
2) Decreasing the precision of the multiplication approximation reduces the compression performance. However, the precision with $K = 3$ is enough to provide better results than both M-coder and MQ-coder for The Large Corpus (+3.0% and +4.4% versus +1.2%) and better or comparable results for Large Calgary Corpus (+0.3% and -0.7% versus -0.5%).

Table III shows an average number of hardware-critical operations, such as multiplications and divisions, needed for encoding and decoding obtained for The Large Corpus data set. We also included renormalizations, since the number of loops within it is not determined and depends on register R. One can see that Proposed 2 requires less renormalizations than other multiplication-free coders.

V. CONCLUSION

In this letter we presented an adaptive multiplication-free arithmetic coder which is preferable when comparing to the existing multiplication-free implementations based on adaptive binary arithmetic coding. The future research will be related to hardware implementation of the proposed arithmetic coding.

[2]In Tables I and II, an average gain with sign '-' means that a considered coder provides better results than the reference one

REFERENCES

[1] C. Lee and H. Park, "Multisymbol data compression using a binary arithmetic coder," *Electronic letters*, vol. 38, no. 3, pp. 124–125, 2002.

[2] S. Mahapatra, K. Singh, "An FPGA-Based Implementation of Multi-Alphabet Arithmetic Coding," *IEEE Transactions on Circuits and Systems – I*, vol.54, no.8, pp.1678–1686, 2007.

[3] A. Biasizzo, F. Novak, P. Korosec, "A Multi-Alphabet Arithmetic Coding Hardware Implementation for Small FPGA Devices," *Journal of Electrical Engineering*, vol. 64, no. 1, pp.44–49, 2013.

[4] Yong Fang and Liang Chen, "Improved binary DAC codec with spectrum for equiprobable sources," *IEEE Transactions on Communications*, vol.62, no.1, pp.256-268, 2014.

[5] D. Zhou, J. Zhou, W. Fei, and S. Goto, "Ultra-high-throughput VLSI architecture of H.265/HEVC CABAC encoder for UHDTV applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.25, no.3, pp.497-507, 2015.

[6] Advanced Video Coding for Generic Audiovisual Services, *document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC)*, Feb. 2012.

[7] High Efficiency Video Coding, *document ITU-T Rec. H.265 and ISO/IEC 23008-2*, Oct. 2014 .

[8] ITU-T and ISO/IEC JTC 1, "JPEG 2000 image coding system: Core coding system, ITU-T Recommendation T. 800 and ISO/IEC 15444-1," JPEG 2000 Part 1, 2000.

[9] E. Krichevski and V. Trofimov, "The performance of universal encoding, *IEEE Transactions on Information Theory*, vol.27, pp.199-207, 1981.

[10] D. Marpe, H. Schwarz and T. Wiegand, "Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.13, no.7, pp.620–636, 2003.

[11] E. Belyaev, M. Gilmutdinov, A. Turlikov, "Binary arithmetic coding system with adaptive probability estimation by Virtual Sliding Window," *Proceedings of the 10th IEEE International Symposium on Consumer Electronics*, St. Petersburg, Russia, pp.194–198, 2006.

[12] E. Belyaev, A. Veselov, A. Turlikov and Kai Liu, "Complexity Analysis of Adaptive Binary Arithmetic Coding Software Implementations," *The 11th International Conference on Next Generation Wired/Wireless Advanced Networking*, St. Petersburg, Russia, 2011.

[13] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon and R. B. Arps, "Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM Journal of Research and Development*, vol.32, no. 6, pp. 717–726, 1988.

[14] Large Calgary Corpus, http://www.data-compression.info/Corpora/CalgaryCorpus/.

[15] The Large Corpus, http://corpus.canterbury.ac.nz/descriptions/.

[16] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, pp. 520–540, 1987.