Fast and Space-Efficient Adaptive Arithmetic Coding *

Boris Ryabko and Andrei Fionov

Abstract

We consider the problem of constructing an adaptive arithmetic code in the case when the source alphabet is large. A method is suggested whose coding time is less in order of magnitude than that for known methods. We also suggest an implementation of the method by using a data structure called "imaginary sliding window", which allows to significantly reduce the memory size of the encoder and decoder.

Index terms: Arithmetic coding, fast algorithms, source coding, randomization, data compression.

1 Introduction

Arithmetic coding is now one of the most popular methods of source coding. The basic idea of arithmetic coding was formulated by Elias in the early 1960s (see [1]). The first step toward practical implementation of this idea was made by Rissanen [2] and Pasco [3] in 1976. Soon after that in [4]–[6] the modern concept of arithmetic coding, which is referred to in the present paper, was described. The method was further developed in a number of works, see, e.g., [7].

The advantage of arithmetic coding over other coding techniques is that it allows to attain arbitrarily small coding redundancy per one source symbol at less computational effort than any other method (the redundancy is defined as the difference between the mean codeword length and the source entropy). Furthermore, arithmetic coding may easily be applied to sources with unknown statistics, when being combined with adaptive models in an adaptive coding scheme (see, e.g., [8]–[11]).

The main problem in linking arithmetic coding with adaptive models is the following. All known adaptive models provide some estimate of the actual probability $p(a_i)$ for each letter a_i over a source alpabet A. But arithmetic coding is based on cumulative probabilities $Q(a_i) = \sum_{j<i} p(a_j)$. Since probability estimates change after each coded symbol, cumulative probabilities must always be re-computed. This requires about $|A|/2$ operations per symbol, where $|A|$ denotes the size of the alphabet, which affects the speed of coding. The reduction of the coding speed becomes the more noticeable as the alphabet size increases. From this point of view, the alphabet of $2^8 = 256$ symbols, which is commonly used in compressing computer files, may already be treated as a large one. With adoption of the UNICODE the alphabet size will grow up to $2^{16} = 65536$ and

*Supported by the Russian Foundation of Basis Research under the Grant no. 99-01-00586

become sufficiently large to prevent any direct computation of cumulative probabilities. In this paper, we suggest an algorithm that computes cumulative probabilities in $O(\log |A|)$ time, which is exponentially less than for known methods.

To obtain a specified (arbitrarily small) model redundancy we use an adaptive scheme with sliding window. The idea of sliding window is well known in information theory. We give preference to such a scheme because it has the following advantages in comparison with other techniques:

- after having been filled with the actual source symbols, the window represents the source statistics quite precisely and guarantees the specified model redundancy (depending on the window size);
- the scheme easily adapts to changes in statistics;
- the number of digits to represent the probability estimates is constant and independent of the messege length, which enables constant arithmetic precision in computations;
- choosing a proper size of the window eliminates division operations in arithmetic encoding, thus making the encoder (and the decoder) operate more than two times faster.

The only disadvantage of the sliding window scheme, that has so far prevented its wide usage, is the necessity to store the entire window in the encoder (decoder) memory. In obtaining arbitrarily small redundancy, the memory allocated to the window becomes a dominating part in space complexity of the encoder and decoder.

To remedy this disadvantage we use a type of the scheme called "imaginary sliding window" (the concept of which was presented in [12]). This approach allows to preserve all the properties of sliding window without storing the window itself.

The space complexity (S) and time complexity (T) of the proposed method in case of a Markov source of order μ, seen as functions of two variables — the alphabet size $|A|$ and redundancy r, are upper bounded by the following estimates:

$$S < \text{const}\left(|A|^{\mu}|A|\log\frac{|A|}{r}\right)\ \text{bits},$$

$$T < \text{const}\left(\log\frac{|A|}{r}\left(\log|A| + \log\log\frac{|A|}{r}\log\log\log\frac{|A|}{r}\right)\right)\ \text{bit operations}$$

(here and below $\log x = \log_2 x$ and const denotes some (different) constants greater than 1).

Note that the estimates for memoryless sources are obtained if $\mu = 0$. The space complexity is the same as for known methods whereas the time complexity is less in order of magnitude (it assumes the first addend in parentheses to be $\log|A|$ instead of $|A|$ as for known methods).

The paper is organized as follows. In Section 2 we consider the problem of combining a conventional sliding window scheme with arithmetic coding. We describe a method for fast operation with cumulative probabilities and investigate the main properties of a resulting coding scheme. In Section 3 we present an imaginary sliding window scheme which dispenses with the need for storing the window. In Section 4 generalizations for Markov sources are given. The details of arithmetic coding and related issues, that are of interest for the main material but not new, are relegated to the appendix.

2 Fast Coding Using Sliding Window

Let there be given a memoryless source generating letters over the alphabet $A = \{a_1, a_2, \ldots, a_n\}$ with unknown probabilities $p(a_1), p(a_2), \ldots, p(a_n)$. Let the size of the alphabet be a power of two, $n = 2^m$ (it is not a restrictive assumption, because, if $n < 2^m$, the alphabet may be expanded with $2^m - n$ dummy symbols having zero probability with trivial modifications of the algorithms). Let the source generate a message $x_1 \ldots x_{l-1} x_l \ldots$, $x_i \in A$ for all i. The window is defined as a sequence of the last w symbols generated by the source, where w denotes the size of the window. At the moment l the window contains the symbols $x_{l-w} \ldots x_{l-2} x_{l-1}$. During encoding (or decoding), the window "slides" along the message: as a novel symbol is introduced in the window, the oldest one is removed. Each letter $a_i \in A$ is assigned a counter c_i that contains a number of occurences of a_i in the current window plus 1. The addent 1 ensures that no letter of the alphabet has zero probability, i.e. every letter may occur next in the source message despite of its being absent in the current window. In these settings the sum of all counters is equal to the window size plus the alphabet size, $\sum_{i=1}^{n} c_i = w + n$. It is convenient to choose the window size w such that

$$w + n = 2^t, \tag{1}$$

where t is a positive integer. Then the estimates of the symbol probabilities $\hat{p}(a_1)$, $\hat{p}(a_2)$, $\ldots$, $\hat{p}(a_n)$ may be obtained as $\hat{p}(a_i) = c_i/2^t$ for all i.

Denote the novel symbol to be encoded by u and the oldest symbol stored in the window, to be removed, by v (at the moment l, $u = x_l$ and $v = x_{l-w}$). The adaptive encoding of the symbol is performed as follows:

- encode u according to the current estimated probability distribution ($\hat{p}(u) = c(u)/2^t$);
- decrement counter $c(v)$ corresponding to the letter v;
- remove v out of the window;
- increment counter $c(u)$ corresponding to the letter u;
- introduce u in the window.

Note that the total sum of counters remains constant.

The decoder, provided that it starts with the same counter contents as the encoder, decodes the symbol according to the current probability estimates and updates the counters in the same way as the encoder.

Encoding of a symbol given an estimated probability distribution may efficiently be carried out by means of arithmetic coding (see Appendix A for more details). This technique, however, requires that cumulative range $[Q_i,\ Q_{i+1})$ be specified for the symbol $u = a_i$, where Q_* are defined as follows:

$$Q_1 = 0, \quad Q_{i+1} = Q_i + c_i, \quad i = 1, 2, \ldots, n$$

or, equivalently,

$$Q_1 = 0, \quad Q_i = \sum_{j<i} c_j, \quad i = 2, 3, \ldots, n+1. \tag{2}$$

The direct calculation of Q_* using (2) requires $O(n)$ operations (summing t-bit numbers). Below we describe a method that allows to reduce complexity down to $O(\log n)$ operations.

Let us store not only the counters $c_1, c_2, \ldots$ (denote this set by C^1), but also the sums of the pairs $c_1 + c_2, c_3 + c_4, \ldots$ (C^2), the sums of quadruples $c_1 + c_2 + c_3 + c_4, c_5 + c_6 + c_7 + c_8, \ldots$ (C^3), and so on. For example, if $n = 8$ then we need to store the sets C^1, C^2, C^3 (or another equivalent data structure) as shown in Fig. 1. Each member

C^1:	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
	1	2	3	4	5	6	7	8

C^2:	$c_1 + c_2$	$c_3 + c_4$	$c_5 + c_6$	$c_7 + c_8$
	1	2	3	4

C^3:	$c_1 + c_2 + c_3 + c_4$	$c_5 + c_6 + c_7 + c_8$
	1	2

Figure 1: Structure of the sets C^* for $n = 8$

of the sets is a t-bit number and we need $(2n-2)t$ bits of memory to store all these numbers. By using the sets C^* the values of, e.g., Q_4 and Q_8 can be computed as

$$\begin{aligned} Q_4 &= c_1 + c_2 + c_3 &&= C_1^2 + C_3^1, \\ Q_8 &= c_1 + c_2 + \cdots + c_7 &&= C_1^3 + C_3^2 + C_7^1. \end{aligned}$$

It is easily seen that computation of any Q_i, $i = 2, 3, \ldots, 8$, requires at most 3 additions.

More formally, given the counters $c_1, c_2, \ldots, c_n$, $n = 2^m$, the sets C^i, $i = 1, 2, \ldots, m$, are to be constructed as follows:

$$\begin{aligned} &C_1^1 = c_1, \quad C_2^1 = c_2, \quad \ldots, \quad C_n^1 = c_n; \\ &C_1^2 = c_1 + c_2, \quad C_2^2 = c_3 + c_4, \quad \ldots, \quad C_{n/2}^2 = c_{n-1} + c_n; \\ &C_1^3 = c_1 + c_2 + c_3 + c_4, \quad C_2^3 = c_5 + c_6 + c_7 + c_8, \quad \ldots, \quad C_{n/4}^3 = c_{n-3} + c_{n-2} + c_{n-1} + c_n; \\ &\ldots \\ &C_1^m = c_1 + c_2 + \cdots + c_{n/2}, \quad C_2^m = c_{n/2+1} + c_{n/2+2} + \cdots + c_n. \end{aligned}$$

Notice that not all of the members of these sets are used in operations. Virtually, only the members with odd subscripts need to be stored. Nonetheless we refer to the above construction for ease of understanding.

Let us show how to compute Q_{k+1}, $k = 1, 2, \ldots, n$ (recall that $Q_1 = 0$). Let k have the binary expansion $k_m k_{m-1} \ldots k_1$, where each $k_i = 0$ or 1. Then

$$Q_{k+1} = k_m C^m_{\lfloor 2k/n \rfloor} + k_{m-1} C^{m-1}_{\lfloor 4k/n \rfloor} + k_{m-2} C^{m-2}_{\lfloor 8k/n \rfloor} + \cdots + k_1 C^1_k$$

(note that only those C^i_j for which $k_i = 1$ are included in the computation). This sum has at most $m = \log n$ t-bit addents, therefore the complexity of computing cumulative counts is $O(t \log n)$ bit operations.

When incrementing or decrementing counters in the adaptive scheme, all the members of the sets which depend on those counters should also be incremented or decremented. More exactly, incrementing (decrementing) c_k must be replaced by incrementing (decrementing) $C^1_k, C^2_{\lceil k/2 \rceil}, C^3_{\lceil k/4 \rceil}, \ldots, C^m_{\lceil k/(n/2) \rceil}$, which requires also

$O(t \log n)$ bit operations. To increment, e.g., c_3 (Fig. 1) we must increment not only C_3^1, but also C_2^2 and C_1^3.

Finally, the last operation which relies on cumulative counts is finding an interval $I(z) = [Q_i,\, Q_{i+1})$ that contains a given number z (and the letter $u = a_i$ to whom this interval corresponds). This problem arises in arithmetic decoding. Let us show how one can solve this problem using the sets C^* in $O(t \log n)$ time.

First compare z with C_1^m. If $z < C_1^m$ then the target interval lies within $[0,\, C_1^m)$, otherwise it belongs to $[C_1^m,\, 2^t)$. Go on comparing z with C_1^{m-1} in the former case, and with $(C_1^m + C_3^{m-1})$ in the latter, which determines one of the intervals $[0,\, C_1^{m-1})$, $[C_1^{m-1},\, C_1^m)$, $[C_1^m,\, C_1^m + C_3^{m-1})$, or $[C_1^m + C_3^{m-1},\, 2^t)$. This procedure may be described as the following iterative process:

$k := m,\ j := 1,\ Q := 0;$
while $k \neq 0$ **do**
  $j := 2j - 1;$
  if $z \geq Q + C_j^k$ **then** $Q := Q + C_j^k,\ j := j + 1;$
  $k := k - 1.$

The process stops when $k = 0$ and we have $I(z) = [Q,\, Q + C_j^1)$, $u = a_j$. Since the number of iterations in this algorithm is equal to $m = \log n$, the complexity $O(t \log n)$ is obvious.

Now let us concentrate on determining the value of t. From (1) $t = \log(w + n)$, where the alphabet size n is specified a priori and the window size w may be chosen to ensure the desired redundancy. The relationship between the window size and redundancy will be given by the theorem below, but before formulating the theorem some remarks should be made on the notion of redundancy. We consider the maximum (supremum) per symbol redundancy over the set of all memoryless sources, generating letters over the alphabet A, and for brevity, we call it redundancy r, as before.

Theorem 1 *Let there be given a memoryless source generating letters over the alphabet $A = \{a_1, a_2, \ldots, a_n\}$. Let the adaptive scheme based on the sliding window of w symbols, $w + n = 2^t$, and arithmetic coding be applied for encoding symbols generated by the source. Then the redundancy r satisfies the inequality*

$$r < \log e \frac{n-1}{w+1} + \frac{n+1}{w+n}. \tag{3}$$

Proof: It will be convenient to consider the redundancy r to consist of two parts: the one caused by replacing unknown symbol probabilities by their estimates, denoted by r_m (model redundancy), and the other caused by the encoder, denoted by r_c (coding redundancy),

$$r \leq r_m + r_c.$$

Consider these two parts separately.

The model redundancy may be defined by the equation

$$r_m = \sum_{v \in A^w} p(v) \sum_{a \in A} p(a) \log \frac{1}{\hat{p}_v(a)} - \sum_{a \in A} p(a) \log \frac{1}{p(a)}. \tag{4}$$

Here v denotes a window context that determines a probability estimate $\hat{p}_v(a)$ for the symbol a to be encoded, $\hat{p}_v(a) = c_v(a)/2^t$, $p(v)$ denotes the probability of a particular context v. The first term in (4) may be treated as the model entropy, while the second term is the source entropy. The following sequence of equalities and inequalities gives the estimate for r_m.

$$\begin{aligned}
r_m &= \sum_{v\in A^w} p(v)\sum_{a\in A} p(a)\log\frac{2^t}{c_v(a)} - \sum_{a\in A} p(a)\log\frac{1}{p(a)} \\
&= \sum_{v\in A^w} p(v)\sum_{a\in A} p(a)\log\frac{2^t p(a)}{c_v(a)} \\
&= \log e \sum_{v\in A^w} p(v)\sum_{a\in A} p(a)\ln\left(\frac{2^t p(a)}{c_v(a)} - 1 + 1\right) \\
&\leq \log e \sum_{v\in A^w} p(v)\sum_{a\in A} p(a)\left(\frac{2^t p(a)}{c_v(a)} - 1\right) \\
&= \log e \left(\sum_{a\in A}\sum_{i=0}^{w}\binom{w}{i} p(a)^{i+1}(1-p(a))^{w-i}\frac{2^t p(a)}{i+1}\right) - \log e \\
&= \log e \left(\sum_{a\in A} p(a)\sum_{i=0}^{w}\frac{2^t}{w+1}\binom{w+1}{i+1} p(a)^{i+1}(1-p(a))^{(w+1)-(i+1)}\right) - \log e \\
&< \log e \left(\frac{2^t}{w+1}\sum_{a\in A} p(a)\sum_{i=0}^{w+1}\binom{w+1}{i+1} p(a)^{i+1}(1-p(a))^{(w+1)-(i+1)}\right) - \log e \\
&= \log e\frac{2^t}{w+1} - \log e = \log e\frac{2^t-(w+1)}{w+1} \\
&= \log e\frac{n-1}{w+1}.
\end{aligned}$$

Here we used the inequality $\ln(x+1) < x$ and the properties of Bernoulli distribution.

As it is stated by Corollary 3 to Theorem 5, relegated to the appendix, the redundancy of arithmetic coding, provided that the minimum symbol probability is $1/2^t$ and τ-bit registers are used in computations, $\tau \geq t+2$, satisfies the inequality

$$r_c < (n+1)(t+\log e)2^{-(\tau-2)}.$$

We may choose any $\tau = O(t)$, $t\to\infty$, without deteriorating the asymptotic complexity of arithmetic coding. Let

$$\tau \geq t + \log(t+\log e) + 2. \qquad (5)$$

Then

$$r_c < (n+1)(t+\log e)2^{-(t+\log(t+\log e))} = \frac{(n+1)(t+\log e)}{(t+\log e)2^t} = \frac{n+1}{w+n}.$$

□

The next our task is to investigate the computational complexity of the proposed adaptive scheme. We shall measure the space complexity in bits and the time complexity in bit operations and consider both complexities as functions of two arguments: n (the alphabet size) and r (redundancy).

The memory size of the encoder and decoder is caused by the necessity to store the window, which requires $w\log n$ bits, the sets C^* maintaining the cumulative counts, which requires at least $t(n-1)$ bits, and internal registers for arithmetic coding, which requires $O(\tau)$ bits.

The coding time per symbol is determined by the computations over the sets C^*, which requires $O(t \log n)$ bit operations, and the computation of ranges in arithmetic coding, which uses τ-bit multiplication and division (in decoding), which, in turn, requires $O(\tau \log \tau \log\log \tau)$ bit operations (for good algorithms).

According to Theorem 1, to decrease redundancy one should increase the size of the window. Furthermore, when the alphabet size gets bigger, the increase of the window size should be greater to attain the specified redundancy. A question of interest is how will this affect the computational complexity of the scheme. The next theorem answers this question.

Theorem 2 *Let there be given a memoryless source generating letters over an alphabet of size n. Let the adaptive scheme based on sliding window and arithmetic coding be applied to the source providing a specified redundancy r. Then the memory size of the encoder (decoder) S and encoding (decoding) time T are given by the estimates*

$$S < \text{const}\left(\frac{n}{r}\log n + n \log \frac{n}{r}\right), \tag{6}$$

$$T < \text{const}\left(\log \frac{n}{r}\left(\log n + \log\log \frac{n}{r} + \log\log\log \frac{n}{r}\right)\right). \tag{7}$$

Proof: From (5) $\tau = O(t)$, $t \to \infty$. From (1) $t = \log(w+n)$. From Theorem 1 it can be easily deduced that $w < \text{const}(n/r)$, $\log(w+n) < \text{const}\log(n/r)$. Then $w \log n < \text{const}(n/r)\log n$, $t(n-1) < \text{const}\, n \log(n/r)$, $\tau < \text{const}\log(n/r)$. Summing up dominating complexities of all operations gives the statement of the theorem. □

Corollary 1 *If n is increasing and r is to remain constant then*

$$S = O(n \log n),$$
$$T = O(\log^2 n).$$

The time estimate is better than that for known methods (where $T = O(n \log n)$).

Corollary 2 *If n is fixed and $r \to 0$ then*

$$S = O(1/r),$$
$$T = O(\log(1/r)\log\log(1/r)\log\log\log(1/r)).$$

In the next section we show how to improve the memory size estimate to $O(\log(1/r))$ implementing the concept of imaginary sliding window.

3 Imaginary Sliding Window

Here we give the description of imaginary sliding window (ISW) scheme in the form suitable for the coding technique considered in the previous sections.

Let there still be given a memoryless source defined in Section 2. We also use the same notions of the window, counters and probability estimation. Denote, additionally, by ν_i the number of occurences of the letter $a_i \in A$ in the current window, $\nu_i = c_i - 1$ for all i. Define ξ to be a random variable taking on values $1, 2, \ldots, n$ with probabilities

$$\Pr\{\xi = i\} = \frac{\nu_i}{w}, \quad i = 1, 2, \ldots, n \tag{8}$$

(the problem of generating such a variable will be considered further below).

The encoding of the novel symbol u using the ISW scheme is performed as follows:

- encode u according to the current estimated probability distribution ($\hat{p}(u) = c(u)/2^t$);
- generate a random variable ξ;
- decrement counter c_ξ, i.e., the counter corresponding to the ξth letter of the alphabet A;
- increment counter $c(u)$ corresponding to the letter u.

A distinctive feature of this scheme is that a *randomly chosen* counter is decremented rather than a counter corresponding to the oldest symbol in the window. In this scheme, the context of the window is not used, therefore storing the window is not needed, which saves $w \log n$ bits of memory. But a question arises whether this scheme is able to represent the source statistics with the precision sufficient to guarantee a specified redundancy? So the next our task is to show that it is, more exactly, we show that the estimated distribution provided by ISW converges with exponential speed to the distribution provided by real sliding window.

Denote by Λ the set of all vectors $\lambda = (\lambda_1, \ldots, \lambda_n)$ such that all λ_i are nonnegative integers and $\sum_{i=1}^n \lambda_i = w$. It is clear that in case of real sliding window $\nu = (\nu_1, \ldots, \nu_n)$ is a random vector that obeys the multinomial distribution

$$\Pr\{\nu_1 = \lambda_1, \nu_2 = \lambda_2, \ldots, \nu_n = \lambda_n\} = \binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \prod_{i=1}^n (p(a_i))^{\lambda_i}, \tag{9}$$

where

$$\binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} = \frac{w!}{\lambda_1! \lambda_2! \ldots \lambda_n!}$$

(see, e.g., [13]).

In case of imaginary sliding window, $\nu = (c_1 - 1, \ldots, c_n - 1)$ is also a random vector whose distribution is a question to be answered. We shall indicate by superscript l the state of vector ν after the lth encoded symbol, i.e., in the process of coding vector ν assumes the states $\nu^1, \nu^2, \ldots, \nu^l, \ldots$ The next theorem decides on the distribution for ν and shows that imaginary sliding window is a sufficiently precise model of real sliding window.

Theorem 3 *Let there be given a memoryless source generating letters over the alphabet $A = \{a_1, \ldots, a_n\}$ with probabilities $p(a_1), \ldots, p(a_n)$; the imaginary sliding window scheme is used with the window size w. Then*

$$\lim_{l \to \infty} \Pr\{\nu_1^l = \lambda_1, \nu_2^l = \lambda_2, \ldots, \nu_n^l = \lambda_n\} = \binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \prod_{i=1}^n (p(a_i))^{\lambda_i}, \tag{10}$$

the limit in (10) exists for any initial distribution $\nu^0 = (\nu_1^0, \ldots, \nu_n^0)$, *and there exists a constant* $C < 1$, *independent of* $(\lambda_1, \ldots, \lambda_n)$ *and* $(\nu_1^0, \ldots, \nu_n^0)$, *such that*

$$\left| \Pr\{\nu_1^l = \lambda_1, \nu_2^l = \lambda_2, \ldots, \nu_n^l = \lambda_n\} - \binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \prod_{i=1}^{n} (p(a_i))^{\lambda_i} \right| < C^l. \tag{11}$$

Proof: Define a Markov chain M with Λ states, each state being correspondent to a vector from Λ (informally, each vector $\nu = (\nu_1, \ldots, \nu_n)$ of imaginary sliding window has a corresponding state in M). A transition matrix for M is defined as follows:

$$P_{\lambda^1 \to \lambda^2} = \begin{cases} \frac{\lambda_j^1}{w} p(a_i), & \text{if } \lambda_i^2 = \lambda_i^1 + 1,\ \lambda_j^2 = \lambda_j^1 - 1,\ i \neq j,\ \lambda_k^2 = \lambda_k^1,\ k \neq i,\ k \neq j; \\ \sum\limits_{k=1}^{n} \frac{\lambda_k^1}{w} p(a_k), & \text{if } \lambda^1 = \lambda^2; \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

This probability matrix corresponds to transition probabilities of imaginary sliding window. The first line in (12) corresponds to the case when the jth coordinate of the vector ν^l is decremented by 1, and the ith coordinate is incremented by 1, which corresponds to introducing the letter a_i in the window, which, in turn, occurs with probability $p(a_i)$. The second line in (12) corresponds to the case when the vector ν^l remains intact. This happens if a coordinate ν_k^l is first decremented (with probability ν_k^l/w) and then incremented (with probability $p(a_k)$).

The chain M has a finite number of states and is plainly seen to be non-periodical (see definition in, e.g., [13]). Consequently, there exist limiting probabilities for M, i.e., such π_λ that for any $\mu, \lambda \in \Lambda$ there exists the limit

$$\lim_{l \to \infty} P_{\mu \to \lambda}(l) = \pi_\lambda \tag{13}$$

independent of μ (here $P_{\mu \to \lambda}(l)$ denotes the probability of transition from μ to λ in l steps, $l \geq 1$).

It is known (see [13]) that limiting probabilities satisfy the system of equations

$$\begin{cases} \pi_\lambda = \sum\limits_{\mu \in \Lambda} \pi_\mu P_{\mu \to \lambda}, \quad \lambda \in \Lambda, \\ \sum\limits_{\lambda \in \Lambda} \pi_\lambda = 1. \end{cases} \tag{14}$$

Next we show that for any $\lambda = (\lambda_1, \ldots, \lambda_n) \in \Lambda$

$$\pi_\lambda = \binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \prod_{i=1}^{n} (p(a_i))^{\lambda_i}, \tag{15}$$

To do this, substitute (15) and (12) in (14). As a result,

$$\binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \prod_{i=1}^{n} (p(a_i))^{\lambda_i}$$
$$= \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \binom{w}{\lambda_1, \ldots, \lambda_i - 1, \ldots, \lambda_j + 1 \ldots \lambda_n}$$

$$\times \left(\prod_{i=1}^{n}(p(a_i))^{\lambda_i}\right) \frac{p(a_j)}{p(a_i)} \left(\frac{\lambda_j+1}{w} p(a_i)\right)$$
$$+\sum_{k=1}^{n} \binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \prod_{i=1}^{n}(p(a_i))^{\lambda_i} \left(\frac{\lambda_k}{w} p(a_k)\right). \tag{16}$$

Substitute the equality

$$\binom{w}{\lambda_1, \ldots, \lambda_i - 1, \ldots, \lambda_j + 1 \ldots \lambda_n} = \binom{w}{\lambda_1, \lambda_2, \ldots, \lambda_n} \frac{\lambda_i}{\lambda_j + 1}$$

in (16) and after simple transpositions obtain

$$\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \left(\frac{\lambda_i}{\lambda_j+1} \frac{p(a_j)}{p(a_i)}\right) \left(\frac{\lambda_j+1}{w} p(a_i)\right) + \sum_{k=1}^{n} \frac{\lambda_k}{w} p(a_k) = 1.$$

Consequently, (15) is the solution of (14). Taking into account the definition (13), we obtain (10).

The second statement of the theorem may be easily obtained from the general proposition on exponential speed convergence to a limiting distribution (see [13]) (this requires that a Markov chain should satisfy certain conditions which is easily checked in our case). □

From the first proposition of the theorem it follows that imaginary sliding window behaves asymptotically just like as real sliding window or, more exactly, the distribution of $(\nu_1^l, \ldots, \nu_n^l)$ approaches, as l increases, the distribution (9) of the numbers of occurences of the letters $a_1, \ldots, a_n$ in the real window. From the second proposition it follows that in case of change in the source statistics at the moment l, the distribution of $(\nu_1^{l+\delta}, \ldots, \nu_n^{l+\delta})$ converges to the new distribution as δ increases ($(\nu_1^l, \ldots, \nu_n^l)$ being thus considered as an initial distribution). Therefore, applied to memoryless sources, imaginary sliding window has the same properties as real sliding window, namely, it allows to estimate the source statistics quite precisely and adapts fast if the statistics change.

Let us now consider the problem of generating random variable ξ. Assume that the only source of randomness is a symmetric binary source generating a sequence $z_1 z_2 \ldots z_k \ldots$, where each $z_k \in \{0, 1\}$, $\Pr\{z_k = 0\} = \Pr\{z_k = 1\} = 1/2$, and all symbols are independent. We do not consider here the complexity of obtaining random bits. Our aim is to convert a random bit sequence into a random variable ξ with probability distribution (8). Denote by z a random number built from t random bits, i.e., the binary expansion of z is $z_1 z_2 \ldots z_t$. It is plain that z may take on any value from $\{0, 1, \ldots, 2^t - 1\}$ with probability $1/2^t$. It is also plain that, for any $w < 2^t$, the process of generating z until $z < w$ is obtained, produces a random variable z^* that takes on any value from $\{0, 1, \ldots, w-1\}$ with probability $1/w$.

Define

$$\Theta_1 = 0, \quad \Theta_i = \sum_{j<i} \nu_j, \quad i = 2, 3, \ldots, n+1, \quad (\Theta_{n+1} = w).$$

Consider a random variable ξ over $\{0, 1, \ldots, w-1\}$ such that $\xi = i$ iff $\Theta_i \le z^* < \Theta_{i+1}$. It can be easily found that

$$\Pr\{\xi = i\} = \Pr\{\Theta_i \le z^* < \Theta_{i+1}\} = \frac{\Theta_{i+1} - \Theta_i}{w} = \frac{\nu_i}{w}.$$

This is our intended distribution (8).

To implement this procedure notice that finding Θ_i and Θ_{i+1} given z^* is the same process as finding an interval and a correspondent symbol in arithmetic decoding (which has been described in Section 2). Hence, this procedure may be efficiently performed by using the data structure C^* defined in Section 2. The only difference is that the values ν_i are less by one than counters c_i involved in the construction of C^*. This may be easily checked by substracting $s = 2^{k-1}$ from C_j^k. Write down the algorithm:

(a)
repeat generating random number z **until** $z < 2^t - n$;
(b)
$k := m,\ i := 1,\ \Theta := 0,\ s := 2^{m-1}$;
while $k \neq 0$ **do**
  $i := 2i - 1$;
  if $z \geq \Theta + C_i^k - s$ **then** $\Theta := \Theta + C_i^k - s,\ i := i + 1$;
  $s := s/2,\ k := k - 1$.

When the process stops we have $\xi = i$.

Denote by N_a the average number of iterations in (a). Since each iteration is taken with probability $n/2^t$,

$$N_a = 1 + \frac{n}{2^t}\left(1 + \frac{n}{2^t}(1 + \cdots)\right) = 1 + \frac{n}{2^t} + \left(\frac{n}{2^t}\right)^2 + \cdots = \frac{1}{1 - n/2^t}.$$

Each of $m = \log n$ iterations in (b) requires $O(t)$ bit operations. Hence the average complexity of the algorithm of generating ξ

$$T_\xi = O\left(\frac{1}{1 - n/2^t} + t \log n\right) = O(t \log n), \quad t \to \infty,\ n \to \infty$$

and does not increase the overall asymptotic complexity of the method (here we assume that in attaining a specified redundancy 2^t grows faster than n).

All the results obtained in this section may be summed up in the following

Theorem 4 *Let there be given a memoryless source generating letters over the alphabet $A = \{a_1, a_2, \ldots, a_n\}$. Let the adaptive scheme based on ISW with t-bit frequency counters and arithmetic coding be applied for encoding symbols generated by the source. Then*

i) *the redundancy r is asymptotocally upper bounded by the inequality*

$$r < \log e \frac{n-1}{2^t - n + 1} + \frac{n+1}{2^t}; \tag{17}$$

ii) *the memory size of the encoder (decoder) S and encoding (decoding) time T are given by the estimates*

$$S < \text{const}\left(n \log \frac{n}{r}\right), \tag{18}$$

$$T < \text{const}\left(\log \frac{n}{r}\left(\log n + \log\log \frac{n}{r} + \log\log\log \frac{n}{r}\right)\right). \tag{19}$$

So the time complexity of adaptive coding using imaginary sliding window is essentially the same as in case of real sliding window, while the space complexity is less in order of magnitude because the window is not to be stored (we eliminate the term $(n/r)\log n$ in (6)).

Let us give some remarks, without any strict considerations, about the ways of obtaining random bits $z_1 z_2 \ldots z_k \ldots$. To maintain the imaginary sliding window these bits are to be known to both the encoder and decoder. A usual way to solve the problem is to use synchronized generators of pseudorandom numbers. We suggest an additional way, namely, using the bits of the code sequence (maybe, with some simple randomizing permutations). The ground for this proposal is that the compressed data are almost random, more exactly, the code sequence approaches a sequence of uniformly distributed code letters as the coding redundancy decreases. The encoder and decoder may use a part of already encoded message for producing random numbers. It is important that the code sequence is known to both the encoder and decoder and they need to store only a small current part of it. This way of obtaining random numbers may occur to be easier than using separate generators.

4 Generalizations

All the results obtained for memoryless sources are extended to a more general case. Consider a Markov source of order μ, i.e., each source symbol depends on μ preceding symbols, $\mu \geq 1$. Let the source alphabet be again $A = \{a_1, a_2, \ldots, a_n\}$. For coding such a source the well-known in information theory technique of decomposing a Markov source into $|A|^\mu$ memoryless sources may be used. Store $|A|^\mu = n^\mu$ imaginary sliding windows, each window corresponding to one word in A^μ. Besides, store a small real window containing the last μ coded symbols. Let the source generate a message $x_1 x_2 \ldots x_l \ldots$ Then, before coding x_l, the word $x_{l-\mu} \ldots x_{l-1}$ is stored in the real window. This word belongs to A^μ and there exists an imaginary sliding window correspondent to the word. The novel symbol x_l may be coded according to the information contained in this imaginary window, after which the window itself is transformed in the same way as in memoryless case (i.e., a randomly chosen counter is decremented, and a counter corresponding to x_l is incremented).

Let us give an example. Let $A = \{a, b\}$, $\mu = 2$, and $x_1 x_2 x_3 x_4 \ldots = aaba\ldots$ is the message to be encoded. In the encoder memory 4 imaginary windows are stored correspondent to the combinations aa, ab, ba, and bb, each window containing 2 frequency counters $c(a)$ and $c(b)$. The letter x_3 is encoded based on information contained in the window correspondent to aa, after which the counters $c(a)$ and $c(b)$ of this window change according to the ISW algorithm; x_4 is encoded based on information contained in the window correspondent to ab, and so on. That is each source letter in encoded (and decoded) based on the window that represents the statistics of a corresponding memoryless source.

Since the estimate for redundancy (17) is a supremum over the set of all memoryless sources, encoding of a Markov source by means of the above scheme ensures this estimate for per symbol redundancy to be valid. The time complexity remains the same as (19). The memory size, however, increases n^μ times, because of necessity to store n^μ imaginary windows, so the space complexity becomes

$$S < \text{const}\left(n^{\mu} n \log \frac{n}{r}\right).$$

Appendix

A Arithmetic Coding Redundancy

The aim of this section is to give an estimate of arithmetic coding redundancy in the form that would allow to decide on computational complexity of the method. As far as we know, such an estimate was not ever made. First of all, we adduce the algorithm of arithmetic encoding suitable for our adaptive scheme. This algorithm is close to one described in [7].

Assume that we have three τ-bit registers l, h, and d, to contain the lower and higher bounds and the size of the interval, respectively, $\tau \geq t+2$, $1/2^t$ being the minimal symbol probability, and a counter s of the size σ bits to contain the number of "squeezes" of the interval. Denote by msb$(\cdot)$ the most significant bit of a register, denote by pmsb$(\cdot)$ the bit preceding to the most significant one. Arithmetic coding is applied to blocks of symbols (or the whole messages) $u_1 u_2 \ldots u_L$, where L denotes the length of a block (or a message). With these assumptions the algorithm of arithmetic encoding may be described as follows.

Initial setup

$l := 0,\ h := 2^{\tau} - 1,\ s := 0.$

Encoding of the symbol $u = a_i$ *given the cumulative range* $[Q_i,\ Q_{i+1})$

$d := h - l + 1,$
$h := l + \lfloor dQ_{i+1}/2^t \rfloor - 1,$
$l := l + \lfloor dQ_i/2^t \rfloor;$
while msb(l) = msb(h) **do**
  transmit msb(l),
  s times transmit $1 -$ msb(l), $s := 0,$
  $l := 2l,\ h := 2h + 1 \pmod{2^{\tau}};$
while pmsb$(l) = 1$ **and** pmsb$(h) = 0$ **do**
  $s := s + 1,$
  $l := 2l,\ h := 2h + 1 \pmod{2^{\tau}};$
msb$(l) := 0$ msb$(h) := 1.$

Terminating a block (or a message)

$s := s + 1;$
if pmsb$(l) = 0$ **then** $b := 0$ **else** $b := 1;$
transmit b,
s times transmit $1 - b$.

The following lemmas determine some important features of the algorithm and the theorem gives an estimate of redundancy.

Lemma 1 *The length L of a block to be encoded satisfies the inequality*

$$L \leq \frac{2^{\sigma} - 2}{t - 1}.$$

Proof: In encoding of any symbol whose probability is not less than $1/2^t$, the value of s may be incremented maximum by $t-1$. In encoding of L symbols s cannot be overflowed if

$$L(t-1)+1 \leq 2^\sigma - 1,$$

hence the statement of the lemma. □

Lemma 2 *After encoding of any symbol the size d of the interval satisfies the inequality*

$$d \geq 2^{\tau-2}+2.$$

Proof: By definition, d is the size of the interval $[l,\, h]$, $d = h - l + 1$. Considering the combinations of two high-order bits in l and h after encoding of a symbol, obtain three possible kinds of intervals: $[00*,\, 11*]$, $[00*,\, 10*]$, and $[01*,\, 11*]$, where $*$ denotes $\tau-2$ arbitrary bits. It can be seen that in the first case the interval length is greater than that in the second and third cases which are equivalent. For the second case

$$l \leq 2^{\tau-2}-1, \;\; h \geq 2\cdot 2^{\tau-2}.$$

Hence

$$d \geq 2\cdot 2^{\tau-2}+1-(2^{\tau-2}-1) = 2^{\tau-2}+2.$$

□

Theorem 5 *Per symbol redundancy r_c of arithmetic coding satisfies the inequality*

$$r_c < n(t+\log e)2^{-(\tau-2)} + 2/L, \tag{20}$$

where n is the alphabet size, t is the number of bits to represent probabilities, τ is the size of the registers l and h (interval bounds), $\tau \geq t+2$, L is the size of an encoded block or message.

Proof: The coding redundancy r_c consisits of two parts, denoted by r_τ and r_L, corresponding to the two addends in (20): r_τ arises due to truncation of the interval bounds to τ bits, and r_L is caused by the block length constraint.

Let the symbol u to be encoded encounter with probability $p(u)$, $p(u) \geq 1/2^t$. Because of truncation, the probability $p(u)$ is replaced by the approximation $\hat{p}(u)$ and

$$\begin{aligned} r_\tau &= \sum_{u\in A} p(u)\log\frac{1}{\hat{p}(u)} - \sum_{u\in A} p(u)\log\frac{1}{p(u)} \\ &= \sum_{u\in A}\left(p(u)\log p(u) - p(u)\log\hat{p}(u)\right). \end{aligned}$$

For $\hat{p}(u)$ we have

$$p(u) = \hat{p}(u) \pm q(u), \;\; q(u) < 1/d.$$

Suppose (pessimistically) that $q(u)$ is taken with plus for every letter. By Lemma 2, $d \geq 2^{\tau-2}+2$, $d > 2^{\tau-2}$. Taking into account these estimates, find an upper bound for $p(u)\log p(u) - p(u)\log\hat{p}(u)$ (omit (u) for brevity).

$$
\begin{aligned}
p \log p - p \log \hat{p} &= p \log p - \hat{p} \log \hat{p} - q \log \hat{p} \\
&< \; p \log p - (p - \frac{1}{d}) \log (p - \frac{1}{d}) - \frac{1}{d} \log \frac{\lfloor 2^{-t} d \rfloor}{d} \\
&< \; \frac{d-1}{d} \log \frac{d-1}{d} + \frac{1}{d} \log \frac{d}{\lfloor 2^{-t} d \rfloor} \\
&\leq \; \log e \frac{d-1}{d} \left(\frac{d}{d-1} - 1 \right) + \frac{1}{d} \log \frac{d}{\lfloor 2^{-t} d \rfloor} \\
&= \; \frac{1}{d} \left(\log \frac{d}{\lfloor 2^{-t} d \rfloor} + \log e \right) \\
&< \; \frac{1}{2^{\tau - 2}} \left(\log \frac{2^{\tau - 2}}{2^{\tau - 2} 2^{-t}} + \log e \right) \\
&= \; \frac{t + \log e}{2^{\tau - 2}}.
\end{aligned} \tag{21}
$$

Summing (21) over $u \in A$ gives

$$r_\tau < n(t + \log e) 2^{-(\tau - 2)}.$$

Terminating a block gives at most two extra bits to the code sequence (see the algorithm). Hence,

$$r_L < 2/L.$$

Combining r_τ and r_L completes the proof. □

For the purpose of the main part of the paper, we obtain a more convenient result by considering the following

Corollary 3 *If $\sigma = \tau$ and*

$$L = 2^\tau / t \tag{22}$$

then

$$r_c < (n + 1)(t + \log e) 2^{-(\tau - 2)}.$$

Proof: First notice that for $\tau \geq t + 2$

$$\frac{2^\tau - 2}{t - 1} > \frac{2^\tau}{t}$$

and (22) satisfies Lemma 1. Substitution of (22) in (20) gives the statement of the corollary. □

References

[1] F. Jelinek, *Probabilistic Information Theory.* New York: McGraw-Hill, 1968, pp. 476–489.

[2] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM J. Res. Dev.,* vol. 20, pp. 198–203, May 1976.

[3] R. Pasco, "Source coding algorithm for fast data compression," Ph. D. thesis, Dept. Elect. Eng., Stanford Univ., Stanford, CA, 1976.

[4] F. Rubin, "Arithmetic stream coding using fixed precision registers," *IEEE Trans. Inform. Theory,* vol. IT-25, no. 6, pp. 672–675, Nov. 1979.

[5] J. J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Dev.,* vol. 23, no. 2, pp. 149–162, Mar. 1979.

[6] M. Guazzo, "A general minimum-redundancy source-coding algorithm," *IEEE Trans. Inform. Theory,* vol. IT-26, no. 1, pp. 15–25, Jan. 1980.

[7] I. H. Witten, R. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Comm. ACM,* vol. 30, no. 6, pp. 520–540, June 1987.

[8] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.,* vol. COM-32, no. 4, pp. 396–402, Apr. 1984.

[9] A. Moffat, "A note on the PPM data compression algorithm," Res. Rep. 88/7, Dep. Comput. Sci., Univ. of Melbourne, Australia, 1988.

[10] M. J. Weinberger, N. Merhav, and M. Feder, "Optimal sequential probability assignment for individual sequences," *IEEE Trans. Inform. Theory,* vol. 40, no. 2, pp. 384–396, Mar. 1994.

[11] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: Basic properties," *IEEE Trans. Inform. Theory,* vol. 41, no. 3, pp.653–664, May 1995.

[12] B. Ya. Ryabko, "The imaginary sliding window," in *IEEE Int. Symp. on Information Theory* (Ulm, Germany, June 29 – July 4), 1997, p. 63.

[13] W. Feller, *An Introduction to Probability Theory and Its Applications.* New York: Wiley & Sons, 1970.