Bounds on Redundancy in Constrained Delay Arithmetic Coding

Ofer Shayevitz[*], Eado Meron, Meir Feder and Ram Zamir
Dept. of Electrical Engineering Systems
Tel Aviv University, Tel Aviv 69978, Israel
Email: {ofersha, eado, meir, zamir}@eng.tau.ac.il

Abstract

We address the problem of a finite delay constraint in an arithmetic coding system. Due to the nature of the arithmetic coding process, source sequences causing *arbitrarily large* encoding or decoding delays exist. Therefore, to meet a finite delay constraint, it is necessary to intervene with the normal flow of the coding process, e.g., to insert fictitious symbols. This results in an inevitable coding rate redundancy. In this paper, we derive an upper bound on the achievable redundancy for a memoryless source. We show that this redundancy decays exponentially as a function of the delay constraint, and thus it is clearly superior to block to variable methods in that aspect. The redundancy-delay exponent is shown to be lower bounded by $\log(1/\alpha)$, where α is the probability of the most likely source symbol. Our results are easily applied to practical problems such as the compression of English text.

I Introduction

Arithmetic coding has been introduced by Elias [1], as simple means to sequentially encode a source at its entropy rate, while significantly reducing the extensive memory usage characterizing non-sequential schemes. The basic idea underlying this technique is the successive mapping of growing source sequences into shrinking intervals of size equal to the probability of the corresponding sequence, and then representing those intervals by a binary expansion. Other coding schemes reminiscent of Elias' arithmetic coding have been suggested since, aimed mostly to overcome the finite precision problem of the original scheme [2][3]. The work in [4][5][6] has rendered arithmetic coding not only aesthetic but also practical, solving numerous implementation complexity issues. In this paper, however, we focus solely upon the theoretical trade-off in constrained delay arithmetic coding, neglecting practical considerations such as the above.

Delay in the classical setting of arithmetic coding stems from the discrepancy between source intervals and binary intervals, which may prohibit the encoder from

[*]The work of O. Shayevitz was supported by the Adams Fellowship.

producing bits (encoding delay) or the decoder from reproducing source symbols (decoding delay). Since a significant advantage of arithmetic coding is its inherent sequentiality, a large delay may turn this advantage on its head. As it turns out, for most sources there exists infinite number of source sequences for which the delay is infinite, where each sequence usually occurs with probability zero. A well known example demonstrating this phenomena is that of a uniform source over a ternary alphabet $\{0, 1, 2\}$. The source sequence $111\dots$ is mapped into shrinking intervals that always contain the point $\frac{1}{2}$, and so not even a single bit can be encoded. This observation leads to the question of just how large is the expected delay of the arithmetic coding process for a given source, and if it is bounded at all.

In his lecture notes [7], Gallager has provided an upper bound for the expected delay in arithmetic coding for a memoryless source, which was later generalized to coding over cost channels [8]. In a recent work [9], Gallager's bound on the expected delay was uniformly improved, and the delay's tail probability was upper bounded. Moreover, the expected delay for sources with memory was analyzed, and a sufficient condition for boundedness was derived.

The problem of delay can be practically dealt with by insertion of a fictitious source symbol into the stream to "release" bits from the encoder or symbols from the decoder, whenever the delay exceeds some predetermined threshold. In this paper, we make use of the fictitious symbol strategy, and build on the result of [9] for the delay's tail probability to analyze the achievable redundancy for any given delay constraint.

II Preliminaries

Consider a discrete source over a finite alphabet $\mathcal{X} = \{0, 1, \dots, K-1\}$ with positive symbol probabilities $\{p_0, p_1, \dots, p_{K-1}\}$. A finite source sequence is denoted by $x_m^n = \{x_m, x_{m+1}, \dots, x_n\}$ with $x^n = x_1^n$, while an infinite one is denoted by x^∞. An arithmetic coder maps the sequences $x^n, x^{n+1}, \dots$ into a sequence of nested *source intervals* $\mathcal{I}(x^n) \supset \mathcal{I}(x^{n+1}) \supset \dots$ in the unit interval that converge to a point $y(x^\infty) = \cap_{n=1}^{\infty} \mathcal{I}(x^n)$. The mapping is defined as follows:

$$\begin{aligned} f_1(i) &= \sum_{j=0}^{i-1} p_j\,, \quad f(x^1) = f_1(x_1) \\ f(x^n) &= f(x^{n-1}) + f_1(x_n)\Pr(x^{n-1}) \\ \mathcal{I}(x^n) &= [f(x^n), f(x^n) + \Pr(x^n)) \end{aligned}$$

Notice that $|\mathcal{I}(x^n)| = \Pr(x^n)$ and that source intervals corresponding to different sequences of the same length are disjoint. Following that, a random source sequence X^n is mapped into a random interval $\mathcal{I}(X^n)$, which as n grows converges to a random variable $Y(X^\infty)$ that is uniformly distributed over the unit interval.

For any sequence of binary digits $b^k = \{b_1, b_2, \dots, b_k\}$ we define a corresponding *binary interval*

$$\mathcal{J}(b^k) = \big[0.b_1b_2, \dots b_k0,\ 0.b_1b_2, \dots b_k1\big) \tag{1}$$

and the midpoint of $\mathcal{J}(b^k)$ is denoted by $m(b^k)$.

The process of arithmetic coding is performed as follows. The encoder maps the input symbols x^n into a source interval according to (1), and outputs the bits representing the smallest binary interval $\mathcal{J}(b^k)$ containing the source interval $\mathcal{I}(x^n)$. This process is performed sequentially so the encoder produces further bits whenever it can. The decoder maps the received bits into a binary interval, and outputs source symbols that correspond to the minimal source interval that contains that binary interval. Again, this process is performed sequentially so the decoder produces further source symbols whenever it can.

We now introduce some notions used throughout the paper.

Definition 1. *Assume that n symbols were encoded, k bits were emitted by the encoder, and m symbols were reconstructed by the decoder (both k, m are random variables). The delay*[1] *$d(n)$ of the arithmetic coder is defined as $d(n) = n - m$.*

Let R be the asymptotic expected rate of a lossless source coding system for a DMS with entropy H. The *redundancy* of the coding system is the difference $R - H$.

Definition 2. *The redundancy-delay function $\mathcal{R}(d_c)$ of a DMS for a given delay constraint d_c, is defined as the the minimal redundancy attainable for any lossless coding system, under the constraint that the delay does not exceed $d_c + 1$.*

Definition 3. *The redundancy-delay exponent is defined as*

$$E_{rd} \triangleq \lim_{d_c \to \infty} -\frac{1}{d_c} \log \mathcal{R}(d_c) \tag{2}$$

whenever the limit exists.

As mentioned earlier, in order to satisfy a delay constraint, it is necessary to intervene in the normal flow of the coding process. One possible way to this end, is the insertions of fictitious source symbols along the transmission. This type of strategy is referred to as an *insertion algorithm.* We emphasize that the delay defined above does not take into account the inserted fictitious symbols. This is rationalized if one remembers that in real world applications, delay is measured in time units. Assuming that the source produces symbols in some constant rate synchronized to an external clock, and that the internal system clock is much faster then the external clock, then it is clear that inserting fictitious symbols is costless, and also apparent why measuring the delay in symbols is the natural thing to do (rather than measuring the delay in encoded bits).

III The Delay Tail's Probability

In this section, we review a recent work that considered the delay of an arithmetic coding system for a memoryless source, as a function of the probability of the most likely source symbol

$$\alpha \triangleq \max p_k$$

[1]This definition of the delay is look-back in nature, while in [9] the definition was look-ahead. Nevertheless, these definitions are essentially the same and the results in [9] hold.

Theorem 1 (from [9]). *Assume a sequence of n source symbols x^n has been encoded, and let D be the number of extra symbols that need to be encoded to allow x^n to be fully decoded. Then*[2,3]

$$\Pr(D > d) \leq 4\alpha^d \left(\log e - \log\log e + d \log(1/\alpha)\right) \leq 4\alpha^d \left(1 + d \log(1/\alpha)\right) \tag{3}$$

We now outline the proof.[4] The sequence x^n has been encoded into the binary sequence b^k which represents the minimal binary interval $\mathcal{J}(b^k)$ satisfying $\mathcal{I}(x^n) \subseteq \mathcal{J}(b^k)$. The decoder has so far been able to decode only $m < n$ symbols, where m is maximal such that $\mathcal{J}(b^k) \subseteq \mathcal{I}(x^m)$. After d more source symbols are fed to the encoder, x^{n+d} is encoded into $b^{k'}$ where $k' \geq k$ is maximal such that $\mathcal{I}(x^{n+d}) \subseteq \mathcal{J}(b^{k'})$. Thus, the entire sequence x^n is decoded if and only if

$$\mathcal{I}(x^{n+d}) \subseteq \mathcal{J}(b^{k'}) \subseteq \mathcal{I}(x^n). \tag{4}$$

Now, consider the middle point $m(b^k)$, which is always contained inside $\mathcal{I}(x^n)$ as otherwise another bit could have been encoded. If $m(b^k)$ is contained in $\mathcal{I}(x^{n+d})$ (but not as an edge), then condition (4) cannot be satisfied, and the encoder cannot yield even one further bit. This observation can be generalized to a set $S(\mathcal{I}(x^n))$ of *forbidden points* which, if contained in $\mathcal{I}(x^{n+d})$, x^n cannot be completely *decoded*. For each of these points the encoder outputs a number of bits which may enable the decoder to produce source symbols, but not enough to fully decode x^n. The encoding and decoding delays are therefore treated here simultaneously, rather than separately as in [8].

We now introduce some notations useful in identifying the set $S(\mathcal{I}(x^n))$. Let $[a, b) \subseteq [0, 1)$ be some interval, and p some point in that interval. In the definitions that now follow we sometime omit the dependence on a, b for brevity. We say that p is *strictly contained* in $[a, b)$ if $p \in [a, b)$ but $p \neq a$. We define the *left-adjacent* of p w.r.t. $[a, b)$ to be

$$\ell(p) \triangleq \min\left\{x \in [a, p) \,:\, \exists k \in \mathbb{Z}^+,\, x = p - 2^{-k}\right\}$$

and the *t-left-adjacent* of p w.r.t. $[a, b)$ as

$$\ell^{(t)}(p) \triangleq \overbrace{(\ell \circ \ell \circ \cdots \circ \ell)}^{t}(p)\ , \quad \ell^{(0)}(p) \triangleq p$$

Notice that $\ell^{(t)}(p) \to a$ monotonically with t. We also define the *right-adjacent* of p w.r.t $[a, b)$ to be

$$r(p) \triangleq \max\left\{x \in (p, b) \,:\, \exists k \in \mathbb{Z}^+,\, x = p + 2^{-k}\right\}$$

and $r^{(t)}(p)$ as the *t-right-adjacent* of p w.r.t. $[a, b)$ similarly, where now $r^{(t)}(p) \to b$ monotonically.

[2]All logarithms in this paper are taken to the base of 2.

[3]This bound is tighter than that of [9], by correcting a minor glitch in the optimization step of the proof. However, we use the looser bound as it is more aesthetic.

[4]For a complete treatment accompanied by an illustrative figure, see [9].

Following the above definitions, the set S of all the forbidden points is given by

$$S(\mathcal{I}(x^n)) = \left\{x \in \mathcal{I}(x^n) \,:\, \exists t \in \mathbb{Z}^+ \cup \{0\}\,, x = \ell^{(t)}(m(\mathcal{I}(x^n))) \,\vee\, x = r^{(t)}(m(\mathcal{I}(x^n)))\right\}$$

namely, the set of all forbidden points is exactly the set of all left and right adjacents of the encoder interval's midpoint.

IV Main Result

In this section we use the delay's probability tail distribution mentioned in the previous section, to derive an upper bound for the redundancy-delay function, via a specific arithmetic coding scheme. We emphasize that unlike previous schemes [10], the presented scheme is error free, hence there is zero probability of buffer overflow. Moreover, our figure of merit is the delay in source symbols vs. the redundancy in bits per symbol. Although our results are general in nature, the typical application in mind is that of compressing a source over a large alphabet, much like say, English text. The following Theorem states our main result.

Theorem 2. *Consider a DMS over a finite alphabet $\mathcal{X}$. Then there exists an arithmetic coder in conjunction with a sequential symbol insertion algorithm, such that the decoding delay does not exceed $d_c + 1$ symbols and the following redundancy is achieved, serving as an upper bound on the redundancy-delay function:*

$$\mathcal{R}(d_c) \le 4\alpha^{d_c - c(\alpha)}\Big(1 + (d_c - c(\alpha))\log\left(1/\alpha\right)\Big)^2 \tag{5}$$

where

$$c(\alpha) = \begin{cases} 0 & \alpha < \frac{1}{16} \\ 2\left\lfloor \frac{4}{\log\left(1/\alpha\right)} \right\rfloor - 1 & o.w. \end{cases}$$

.

Corollary 1. *For any DMS, the asymptotical behavior of the redundancy-delay function is exponentially decaying with the delay constraint (5). The redundancy-delay exponent (2) is lower bounded by* [5]

$$E_{rd} \ge \log\left(1/\alpha\right) \tag{6}$$

where α is the probability of the most likely source symbol.

Let us first outline the idea behind the proof. Assume that the encoder discovers that the current decoder delay is $d_c + 1$, and is therefore required to intervene and insert a fictitious symbol to reduce the decoder's delay. We show that it is actually possible to nullify this delay with a single fictitious symbol. We note that if the fictitious symbol does not contain any of the forbidden points discussed in Section III, then this goal is achieved. To this end, we show that there exists an interval mapping for the arithmetic coder which always makes this possible. This key point is proved in the following Lemma.

[5] We can actually show that $E_{rd} \ge \max_{0<\mu<H(P)-\log\frac{1}{\alpha}} \min\left(H(P) - \mu\,,\, \min_{Q\in A_\mu} D(Q\|P) + \log\frac{1}{\alpha}\right)$, where P is the source's distribution, and $A_\mu = \{Q : D(Q\|P) + H(Q) < H(P) - \mu\}$, the proof is omitted due to lack of space.

Lemma 1. *Assume that the current encoder interval is $\mathcal{I}(x^n)$, and let*

$$\varphi_n(\lambda) = (1-\lambda)a_n + \lambda b_n\,, \quad \lambda \in [0,1)\,, \quad \mathcal{I}(x^n) = [a_n, b_n)$$

Then at least one of the following subintervals includes no forbidden points:

$$\mathcal{LF}_n \triangleq (\varphi_n(3/8)\ , \varphi_n(1/2))\ , \quad \mathcal{RF}_n \triangleq (\varphi_n(1/2)\ , \varphi_n(5/8))$$

Proof. Without loss of generality, assume that the midpoint of the smallest binary interval containing $\mathcal{I}(x^n)$, which was denoted by $m = m(b^k)$, satisfies $m \le \varphi_n(1/2)$. We consider two different cases, corresponding to $m \le \varphi_n(3/8)$ and $m > \varphi_n(3/8)$.

In the first case, it is easily verified that the right adjacent of m satisfies $r(m) > \varphi_n(1/2)$, as if this is not so then

$$m + 2(r(m) - m) \in \mathcal{I}(x^n)$$

which contradicts the maximality in the definition of the right adjacent. Therefore in this case $\mathcal{LF}_n$ contains no forbidden points.

For the second case considered where $m \in \mathcal{LF}_n$, take note that since $m < \varphi_n(1/2)$

$$r(m) - m \ge \frac{1}{4}|\mathcal{I}(x^n)|$$

and therefore

$$r(m) \ge m + \frac{1}{4}|\mathcal{I}(x^n)| \ge \varphi_n(5/8)$$

Thus $\mathcal{RF}_n$ contains no forbidden points, proving the Lemma.

□

In the sequel we show that the existence of a *forbidden free* subinterval of $\mathcal{I}(x^n)$, the size of which is lower bounded by $\frac{1}{8}|\mathcal{I}(x^n)|$, which has only two optional positions within $\mathcal{I}(x^n)$ (either $\mathcal{LF}_n$ or $\mathcal{RF}_n$), enables us to guarantee the existence of a *flushing symbol*, i.e., a symbol which if inserted assures that all of its preceding symbols are fully decoded[6]. We are now ready to prove Theorem 2.

Proof. Define an extended alphabet of size $|\mathcal{X}| + 2$, by appending two fictitious symbols $\varepsilon_1, \varepsilon_2$ with equal probability ε to the original alphabet, and shrinking the original symbol probabilities by a factor of $1 - 2\varepsilon$. Assuming that $\alpha < \frac{1}{16}$, we show in Lemma 2 on the appendix that there exists an arithmetic coder matched to the new probabilities, such that the fictitious symbols $\varepsilon_1, \varepsilon_2$ are mapped into intervals contained in $\mathcal{LF}_n, \mathcal{RF}_n$ respectively. If the condition on α is not satisfied, then we can always aggregate a few symbols into a super-symbol, so that the maximal product probability satisfies the required condition (the effect of this aggregation on the delay is treated later on).

Our insertion algorithm is the following. The encoder keeps track of the decoding delay (measured in source symbols) by emulating the decoder. Whenever this delay

[6]Note that the flushing symbol is not unique and is prefix dependent

reaches d_c+1, the fictitious symbol that lies within a forbidden free interval is inserted, hence nullifying the decoding delay. This way, the decoding delay never exceeds d_c+1 and no errors are incurred.

Let us now bound the redundancy (in bits per symbol) due to the insertion algorithm above. There are two different sources of redundancy. The first is due to the shrinking of the original source symbols probabilities, which amounts to $\log \frac{1}{1-2\varepsilon}$ for each symbol that is coded.

The second source of redundancy is due to the coding of the inserted fictitious symbol. The probability of insertion is upper bounded by the probability that the delay exceeds d_c, which is given in Theorem 1, by substituting α with $(1-2\varepsilon)\alpha$ and dividing by $1-2\varepsilon$ to compensate for the unmatched coder[7] . Whenever an insertion occurs, it results in a cost of $\log \frac{1}{\varepsilon}$. Therefore, the expected rate loss due to the above is

$$4(1-2\varepsilon)^{d_c-1}\alpha^{d_c}\left(1+d_c\log\frac{1}{(1-2\varepsilon)\alpha}\right)\log\frac{1}{\varepsilon} \le 4\alpha^{d_c}\left(1+d_c\log(1/\alpha)\right)\log\frac{1}{\varepsilon} \tag{7}$$

Summing both terms, the total redundancy is bounded by

$$\begin{aligned}\mathcal{R}(d_c) &\le \log\frac{1}{1-2\varepsilon} + 4\alpha^{d_c}\left(1+d_c\log(1/\alpha)\right)\log\frac{1}{\varepsilon}\\ &\le 4\varepsilon + 4\alpha^{d_c}\left(1+d_c\log(1/\alpha)\right)\log\frac{1}{\varepsilon} \triangleq \widehat{R}(d_c)\end{aligned} \tag{8}$$

Minimizing, we take the derivative of the bound $\widehat{R}(d_c)$ w.r.t. ε

$$\frac{\partial \widehat{R}(d_c)}{\partial\varepsilon} = 4 - \frac{4\log e}{\varepsilon}\,\alpha^{d_c}\left(1+d_c\log(1/\alpha)\right) = 0 \tag{9}$$

Solving for ε we get

$$\varepsilon = \alpha^{d_c}\left(1+d_c\log(1/\alpha)\right)\log e \tag{10}$$

and substituting into the bound we have

$$\mathcal{R}(d_c) \le 4\alpha^{d_c}\left(1+d_c\log(1/\alpha)\right)^2 \tag{11}$$

Finally, we address the case where $\alpha > \frac{1}{16}$. As mentioned before, we aggregate a minimal number of source symbols w into a super-symbol, such that $\alpha^w < \frac{1}{16}$. This means, for one, that $w < \left\lfloor \frac{4}{\log 1/\alpha} \right\rfloor$. We now carry out the above procedure for the product alphabet. However, since decoding is performed for w symbols at a time, we set our decoding threshold to be $\widetilde{d}_c = \left\lfloor \frac{d_c+1}{w} - 1 \right\rfloor$. Substituting the above into (11) we get

$$\mathcal{R}(d_c) \le 4\left(\alpha^w\right)^{\widetilde{d}_c}\left(1+\widetilde{d}_c\log(1/\alpha^w)\right)^2 \le 4\alpha^{d_c-c(\alpha)}\left(1+(d_c-c(\alpha))\log(1/\alpha)\right)^2 \tag{12}$$

where $c(\alpha) = 2\left\lfloor \frac{4}{\log(1/\alpha)} \right\rfloor - 1$

$\square$

[7]Note that subintervals that correspond to fictitious symbols have zero probability. This in turn increases the probability of the complementary subintervals by a factor of $\frac{1}{1-2\varepsilon}$.

Notice that the scheme described above also allows the encoder to change the delay constraint *on the fly*, by inserting a suitable fictitious symbol in accordance to the modified constraint. Once the decoder is made aware of this change, both encoder and decoder need to simultaneously adjust the probability of the fictitious symbols.

V Discussion

In order to put our work in perspective, we briefly survey related results regarding different notions of delay constrained coding. In [10], an analysis of a variable length encoding scheme for a fixed rate DMS was presented, which minimizes the probability of buffer overflow, as a function of the desired redundancy. This result is very appealing, as the tradeoff is characterized in terms of the R*é*nyi entropy of the source. There are two significant differences between [10] and the one presented herein. First, the delay in [10] is measured in encoded bits, as opposed to source symbols. Second, and more importantly, in [10] there is a possibility of buffer overflow, which results in errors. It should be noted that our scheme has the extra merit of a finite buffer size (which is $d_c \cdot \log(1/p_{min})$).

Another important work that also considers the effects of delay is [11], although this aspect was not the main contribution of that work. The delay defined in [11] is somewhat different than our definition, and grows like $O(\log n)$, where n is the length of the encoded sequence, while achieving a vanishing redundancy. We emphasize that the setting in [11] is very different as it is both universal and context oriented, thus suitable for families of sources with memory.

One might wonder as to how the naive approach of *terminating* the arithmetic coding process compares to our approach. The usual convention of terminating an arithmetic coding scheme is based upon a predetermined sequence length, so that a prefix condition is satisfied and blocks can be concatenated [12]. This means that termination is obtained by emitting bits that force the binary interval to be contained in the last encoder source interval. This is essentially a block to variable coding scheme, and if one imposes a delay constraint d_c, the redundancy-delay function is given by $\mathcal{R}(d_c) = O(\frac{1}{d_c})$, which is clearly inferior to our results.

If, on the other hand, one wishes to terminate the scheme whenever the delay constraint is breached, rather than at a predetermined time, then termination is obtained only by forcing the decoder source interval to be contained inside the last encoder source interval, and it is not obvious what are the appropriate bits that need to be emitted to that end, and more importantly, what is the resulting redundancy. Actually, looking at the bits emitted by our insertion algorithm when the delay is nullified, one gets a possible solution to this problem.

To illustrates the usefulness of our result, we consider the classical example of the compression of English text [13][12]. Assuming a zero-order letter model, the empirical entropy is $H = 4.03$ bits per letter, and the most frequent symbol is the letter E, that appears with a relative frequency of 0.13. We plug in $\alpha = 0.13$ into the results of [9] and those presented herein. We find that the expected delay of an unconstrained arithmetic coding system is upper bounded by $\mathbb{E}(D) \leq 3.56$. In

Figure 1, we plot the bound on the redundancy-delay function for the English text, derived in Section IV. As can be seen, the redundancy for a delay constraint of $d_c = 7$, which is less than twice the bound on the unconstrained expected delay, is $\mathcal{R}(7) = 0.0067$. This loss is negligible w.r.t. the source's entropy, and to guarantee the same performance in a block to variable coding scheme, one would need to require a block size of ~ 150. This in turn implies a delay constraint of the same value, demonstrating the superiority of our algorithm.

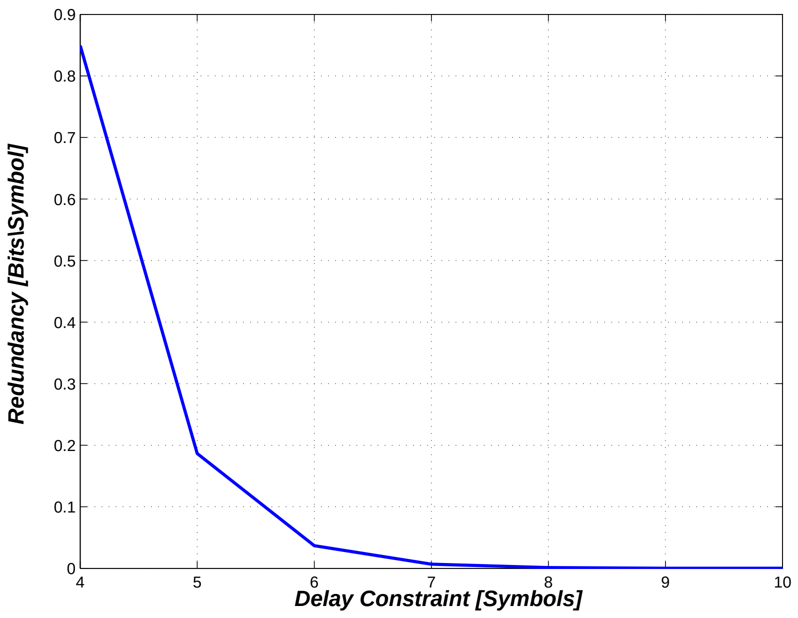

Figure 1: Upper bound on $\mathcal{R}(d_c)$ for zeroth order model of English text

We thank the anonymous reviewers for their helpful comments.

A Appendix

Lemma 2. *Consider a DMS over an alphabet $\mathcal{X}$, with maximal symbol probability $\alpha < \frac{1}{16}$. Then it is possible to generate a mapping for an arithmetic coder matched to that source, so that any two given symbols $i, j \in \mathcal{X}$ will always be mapped into subintervals of $\mathcal{LF}_n, \mathcal{RF}_n$ respectively.*

Proof. First, note that

$$|\mathcal{LF}_1| = |\mathcal{RF}_1| = \frac{1}{8}$$

We map the source symbols adjacently over the unit interval starting from zero and going towards one. The order of mapping can be arbitrary, except for that of i, j. Denote by $\mathcal{S}_k$ the interval to which the kth symbol is mapped, and let k_0 be minimal such that

$$\mathcal{S}_{k_0} \cap \mathcal{LF}_1 \neq \phi$$

Obviously we have that

$$\left|\mathcal{S}_{k_0} \cap \mathcal{LF}_1\right| < \frac{1}{16} \quad \Rightarrow \quad \left|\mathcal{LF}_1 \setminus \mathcal{S}_{k_0}\right| > \frac{1}{16}$$

and therefore we can now map the symbol i and it is guaranteed to be fully contained within $\mathcal{LF}_1$. The same argument applies to the symbol j and the interval $\mathcal{RF}_1$. Due to the nested nature of arithmetic coding, this property remains valid throughout the encoding process, and hence the i, j symbols are always mapped into subintervals of $\mathcal{LF}_n, \mathcal{RF}_n$ respectively. $\square$

References

[1] F. Jelinek, *Probabilistic Information Theory*, McGraw-Hill, New York, 1968.

[2] J. Rissanen, "Generalized kraft inequality and arithmetic coding," *IBM Journal of Research and Development*, vol. 20, pp. 198 – 203, 1976.

[3] R. Pasco, *Source Coding Algorithm for Fast Data Compression*, Ph.d dissertation, Dep. Elec. Eng., Stanford Univ., Stanford, CA, 1976.

[4] G. G. Langdon Jr., "An introduction to arithmetic coding.," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

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

[6] A. Moffat, R.M. Neal, and I.H. Witten, "Arithmetic coding revisited," *ACM Trans. Inf. Syst.*, vol. 16, no. 3, pp. 256–294, 1998.

[7] R.G. Gallager, *Lecture Notes (unpublished)*, 1991.

[8] S.A. Savari and R.G. Gallager, "Arithmetic coding for finite-state noiseless channels," *IEEE Trans. Info. Theory*, vol. 40, pp. 100 – 107, 1994.

[9] O. Shayevitz, R. Zamir, and M. Feder, "Bounded expected delay in arithmetic coding," *in Proceedings of ISIT 2006*, 2006.

[10] F. Jelinek, "Buffer overflow in variable length coding of fixed rate sources," *IEEE Trans. Info. Theory*, vol. IT-14, pp. 490 – 501, May 1968.

[11] M.J. Weinberger, A. Lempel, and J. Ziv, "A sequential algorithm for the universal coding of finite memory sources," *IEEE Trans. Info. Theory*, vol. 38, no. 3, pp. 1002 – 1014, May 1992.

[12] T.M. Cover and J.A Thomas, *Elements of Information theory*, John Wiley & Sons, Inc., 1991.

[13] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, Jul and Oct 1948.