Diss. ETH No. 13099

On Coding by Probability Transformation

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by
JOSSY SAYIR
dipl. El.-Ing. ETH
born 8 June 1968
citizen of Zürich ZH

accepted on the recommendation of
Prof. Dr. James L. Massey, examiner
Prof. Dr. Frans M. J. Willems, co-examiner
Prof. Dr. Fritz Eggimann, co-examiner

1999

Acknowledgements

It is not simple to express the extent of my gratitude for Jim Massey. His original and playful attitude towards teaching, research and learning has been an inspiration and an example which I strive to reflect. I place the moments I spent with Jim among my dearest memories and wish that there will be more such moments after the completion of my doctorate. I also thank Lis Massey for her warm hospitality and continued concern for the well-being of Jim's students.

I profited from many enriching discussions with Frans Willems before and after he agreed to be the co-referee of my thesis and I would like to thank him on this account. Many thanks are also due to Prof. Eggimann for accepting the task of second co-referee in the last minute and fulfilling this task so gracefully.

Many people at ISI and ETH contributed to the outstanding environment I enjoyed during my period here. Among them, I want to mention Zsolt Kukorelly who shared my office in spite of my awkward reputation and has become a close friend, Gerhard Kramer with whom I had uncountable discussions and whose interest was a constant source of motivation, Beat Keusch and Richi de Moliner with whom I shared many pizzas and late night conversations. At various stages of my thesis, I was lucky and grateful to collaborate with Thomas Ernst and with Claudio Weidmann. I would like to extend my deep thanks to all the students who trusted me to supervise their research projects.

The members of the jazz band "Who Cares?", Richi Hunziker, Stefan Oberle, Reto Bättig and Igor Ugolini were instrumental in making my stay at ETH unforgettable. Special thanks to Markus Schenkel, my co-saxophonist and sailing partner, who stirred my enthusiasm for my work during many fruitful discussions.

Finally, I would like to express my warmest gratitude to my family and friends for supporting and encouraging me at all times. Especially, I want to mention my friend Laurent who delivered food to the ETH at

ungodly hours in times of emergency, and my dearest friend Keren, who constantly reminded me that there are far more important things in life than entropy and capacity by elegantly changing the subject each time I started to talk about my work. My last and greatest thank you goes to my parents Edna and Mahir for putting up with me when I was grumpy and for all the rest.

Abstract

An introduction to arithmetic source coding is given, showing how this technique transforms the output probability distribution of the source into an almost uniform probability distribution. A similar approach is investigated for block coding for noisy channels, leading to the definition of the (N, K) block coding capacity of a discrete memoryless channel. Arithmetic coding is modified for data transmission over noisy channels and a metric is developed for a sequential decoder to be used in conjunction with an arithmetic encoder.

Universal arithmetic source coding is investigated for a class of sources whose output distributions lie within a polytope of probability distributions. The properties of the optimal coding distribution over a polytope of distributions are derived. Gallager's redundancy-capacity theorem is presented. An iterated version of the Arimoto-Blahut algorithm is fomulated to compute the optimal coding distribution for a polytope with many vertices, or, alternatively, to compute the capacity of a discrete memoryless channel with a large input alphabet. The optimal coding distribution is computed for the polytope of all monotone non-increasing probability distributions with a given expected value.

Universal source coding with a source transformation is described. Two source transformers are investigated: the recency-rank calculator (also called the move-to-front list) and the competitive list transformer. It is shown that the steady-state output distribution of these transformers is monotone non-increasing when the transformers are applied to the output of a discrete memoryless source. A context-tree algorithm is formulated which uses competitive list transformers followed by a universal arithmetic source encoder. The performance of this algorithm is compared to the performance of other universal source coding algorithms.

Zusammenfassung

Nach einer Einführung in die arithmetische Quellenkodierung wird erläutert, dass diese Methode die Wahrscheinlichkeitsverteilung am Ausgang einer Quelle in eine angenäherte Gleichverteilung umwandelt. Eine ähnliche Überlegung für die Blockkodierung für verrauschte Kanäle führt zur Definition der (N, K) Blockkodierungskapazität eines diskreten gedächtnisfreien Kanals. Die Funktionsweise eines arithmetischen Kodierers wird angepasst an die Übertragung von Daten über verrauschte Kanäle. Eine Metrik wird erläutert für einen sequentiellen Dekodierer, der gemeinsam mit einem arithmetischen Kodierer benützt wird.

Universelle arithmetische Quellenkodierung wird für eine Klasse von Quellen erforscht, deren Ausgangsverteilungen in einem Polytop von Wahrscheinlichkeitsverteilungen liegt. Die Eigenschaften der optimalen Kodierverteilung für einen Polytop von Verteilungen werden erläutert. Das Redundanz-Kapazitätstheorem von Gallager wird erklärt. Eine iterierte Version des Arimoto-Blahut Algorithmus wird formuliert, um die optimale Kodierverteilung für einen Polytop mit vielen Ecken oder um die Kapazität eines diskreten gedächtnisfreien Kanals mit einem grossen Eingangsalphabet zu berechnen. Die optimale Kodierverteilung für den Polytop aller monoton abnehmenden Wahrscheinlichkeitsverteilungen mit einem vorgegebenen Erwartungswert wird hergeleitet.

Universelle Quellenkodierung mit einer Quellentransformation wird beschrieben. Zwei Quellentransformatoren werden erforscht: der Neuheitsrangberechner (auch "move-to-front" Liste genannt) und der Transformator mit einer kompetitiven Liste. Es wird gezeigt, dass die Ausgangsverteilungen dieser Transformatoren im stationären Zustand monoton abnehmen, wenn die Transformatoren auf den Ausgang einer diskreten gedächtnisfreien Quelle angewendet werden. Ein Kontext-Baum Algorithmus wird formuliert, der Transformatoren mit kompetitiven Listen gefolgt von einem universellen arithmetischen Quellenkodierer einsetzt. Dieser wird mit anderen universellen Quellenkodieralgorithmen verglichen.

Contents

Introduction

At the International Symposium held on the occasion of the fiftieth anniversary of Shannon's 1948 paper [1], the "fathers" of Information Theory turned a satisfied eye on half a century of sensational accomplishments and gave an optimistic outlook for the future of the field. Having survived decades of skepticism, first by mathematicians, then by engineers refusing to give up their unfounded beliefs, Information Theory has finally gained widespread acceptance. Its precepts have silently conquered many fields of engineering, spanning applications as varied as mobile telephony, the Internet and medical research. Nonetheless, the technical sessions at this and other symposia and conferences can produce a very different impression in the enthusiastic mind of a student of Information Theory: though it has survived many onslaughts from outside the field in the past, Information Theory is *about to be devoured by its own children.*

Source coding, more commonly known by its non-information-theoretic name of "data compression", has drifted far into the field of complexity theory. The latest successes in the field have little to do with the probabilistic model underlying Shannon's Theory[1]. Channel coding has undergone a meiotic division to become a branch of algebra on the one hand, and a part of modulation and communication engineering on the other hand. Cryptology, which owes part of its rebirth to Information Theory, has become a field of its own whose portion of public attention rivals the fame of Information Theory in its heyday. Meanwhile, Information Theory has retired to the sidelines, becoming a mere spectator while its brilliant descendants strive to fulfill its promises. Remembering the old division between the "coders" and the "bounders", today the "bounders" have dwindled and remained within the field, while the "coders" have prospered by digging roots into more fertile soil.

Far from decrying this fruitful division of labor, the present disserta-

[1] I was very much taken aback at a conference on data compression when I mentioned the name of Claude Shannon and received the answer "Claude who?"

tion will take a nostalgic stand by attempting to develop coding methods based on purely probabilistic and information-theoretic premises. While the promises of Information Theory rest upon probabilistic measures such as the entropy of a source and the capacity of a channel, coding algorithms designed to fulfill those promises are commonly tailored for non-probabilistic measures such as the compression rate achieved for a set of sequences and the distance properties of a channel code. In this dissertation, we will take the inverse approach of designing coding algorithms based on the entropy, the capacity and the probability distributions associated with them. We will view source and channel encoders as devices whose aim is to *transform* the probability distribution of a source into a probability distribution required by a channel or by a universal source encoder. In order to assess the success of this approach, we will evaluate the coding algorithms developed in terms of the non-information-theoretic measures of coding theory.

This dissertation contains three chapters. We give an outline of each chapter.

- After giving an introduction to arithmetic coding in Chapter 1, we explain why this well-known source coding algorithm falls into the framework of coding by probability transformation. Building on this insight, we show how an arithmetic encoder can be modified to function as an encoder for noisy channels. We derive a metric for a sequential decoder to be used in conjunction with an arithmetic channel encoder, and evaluate the performance of the encoder in a simulation involving a binary memoryless source, an arithmetic encoder, a binary memoryless channel and a sequential decoder.

- In Chapter 2, we consider the use of an arithmetic source encoder for universal coding. After providing some reflections on the meaning of the term "universal", we specify the problem of finding the optimal arithmetic encoder for a class of sources. We concentrate on the class of all N-ary stationary sources whose marginal probability distributions lie in a convex set known as a *polytope* and derive the properties of an optimal arithmetic encoder for any polytope of probability distributions. A polytope is of practical interest because it includes the set of all monotonely decreasing distributions as a special case. We then present Gallager's much simpler approach to finding the optimal encoder for a class of sources, which reduces the problem to the equivalent problem of computing the capacity of a channel. Based on Gallager's result and on the insight we gained in our derivation, we formulate an iterated variation of the Arimoto-Blahut algorithm,

which can be used to determine the optimal arithmetic encoder for a polytope, or, alternatively, to compute the capacity of a channel whose input alphabet is very large.

- In Chapter 3, the concept of source transformation for universal arithmetic coding is introduced. We present a context-tree source coding algorithm which uses source transformers and the optimal arithmetic encoders of Chapter 2. The performance of this algorithm is evaluated when the algorithm is used to compress a few standard collections of files.

The three chapters of this dissertation are mostly independent, but they share the view of source and channel encoders as devices performing a probability transformation. Each of the three chapters draws its inspiration from a different aspect of the work of Peter Elias, so this dissertation can be viewed as an attempt to pursue and extend some of Elias' ideas.

Chapter 1

Arithmetic Coding for Noisy Channels

It has been said that one cannot truly understand an idea in science until one has tracked down the idea to its origin, found out how the idea was generated and how it rose to maturity. In information theory, the origin of all things is Claude Shannon's 1948 paper "A Mathematical Theory of Communication" [1]. This is particularly true of arithmetic coding. The purpose of this chapter is first to trace the elements of this coding technique back to Shannon and beyond. Following this, we will modify arithmetic coding so it can be applied to channel coding.

We begin with a tutorial on arithmetic coding that takes us from the fundamentals to the practical aspects of implementing an encoder and a decoder. Following this, we will drop our engineering spectacles and take a historian's look at arithmetic coding. We will show how the algorithm was born by linking a method described in Shannon's original paper to an idea commonly attributed to Peter Elias, and we will venture a guess as to how Shannon came up with this particular method. We will then attempt to transfer Shannon's perspective on source coding to channel coding. This will lead us to arithmetic coding with gaps. Although this will seem like an eccentric coding method at first glance, we will demonstrate how every block code and convolutional encoder can be implemented as an arithmetic encoder. As an example, we will show the performance of a simple arithmetic encoder used in conjunction with a sequential decoder, which will require the development of a metric for arithmetic coding similar to Fano's metric for sequential decoding. We will conclude with some remarks on the implications and possible improvements of the method described.

1.1 An Introduction to Arithmetic Coding

An arithmetic encoder is a simple source-coding procedure whose description in any reasonable programming language will typically require only a page. The decoder is equally simple and almost identical to the encoder. Although the implementation of the arithmetic coding algorithm is simple, it relies on three of fundamental ideas which we will describe in this section. Following these explanations, we outline a simple program of the encoder in pseudo-code.

Codewords locate the cumulative probability of the input.

Consider all blocks of L symbols from the alphabet of an information source. The *probability* of a block is the probability that the source emits this block. The *cumulative probability* of a block is the sum of the probabilities of all blocks lexicographically prior to that block. Let f be the cumulative probability of a given block and p be its probability. We call the semi-open interval $[f, f+p)$ the *source interval* of the given block. We want to assign a D-ary codeword to each block. A D-ary sequence $b_1, \ldots, b_w$ of length w will be thought of as a base D rational number $0.b_1 b_2 \ldots b_w = \sum_{i=1}^{w} b_i D^{-i}$. We call the semi-open interval $[0.b_1 \ldots b_w, 0.b_1 \ldots b_w + D^{-w})$ the *sequence interval* for the D-ary sequence $b_1 \ldots b_w$. The interval of a sequence that is the prefix of another sequence contains the interval of this other sequence. Therefore, a code is prefix-free just when the sequence intervals of its codewords are pairwise disjoint. This can be ensured by demanding that the sequence interval of each codeword lie inside the source interval of the block to which it is assigned. The following rule accomplishes this:

> For each block, choose the codeword interval to be (one of) the longest sequence interval(s) that fits inside the source interval for this block.

Example: Consider a binary memoryless source whose symbol probabilities are $p_0 = .25$ and $p_1 = .75$. We will design a 10-ary code, i.e. $D = 10$, for blocks of $L = 2$ source symbols. We set up a table of block probabilities and cumulative probabilities, choose codeword intervals and obtain codewords, as indicated in Table 1.1. The choice of the codeword interval for the second and for the third block is unique. However, there are six possible choices for the codeword interval for the first block, viz. $[.00, .01)$, $[.01, .02)$, $[.02, .03)$, $[.03, .04)$, $[.04, .05)$ and $[.05, .06)$, and five possible

Block	p	f	Source interval	Codeword interval	Codeword
00	.0625	0	[0, .0625)	[.00, .01)	00
01	.1875	.0625	[.0625, .25)	[.1, .2)	1
10	.1875	.25	[.25, .4375)	[.3, .4)	3
11	.5625	.4375	[.4375, 1)	[.5, .6)	5

Table 1.1: Encoding blocks of 2 symbols using a 10-ary code.

Block	p	f	Source interval	Codeword interval	Codeword
00	.0001	0	[0, .0001)	[.0000, .0001)	0000
01	.0011	.0001	[.0001, .01)	[.001, .010)	001
10	.0011	.01	[.01, .0111)	[.010, .011)	010
11	.1001	.0111	[.0111, 1.0)	[.1, 1.0)	1

Table 1.2: Encoding blocks of 2 symbols using a binary code.

choices for the codeword interval for the last block, viz. $[.5, .6)$, $[.6, .7)$, $[.7, .8)$, $[.8, .9)$ and $[.9, 1.0)$. A more practical example for the same source might use a binary code, i.e., $D = 2$. This requires us to rewrite our table in base 2, as indicated in Table 1.2. For this binary case, the choice of each of the four codeword intervals happens to be unique.

Since the length of a source interval is equal to the probability p of a block, a sequence interval of length $D^{-\lceil -\log_D p \rceil}$ will always be shorter than or equal to the length p of the source interval and any smaller power of D^{-1} will be greater than this length. However, since sequence intervals of length D^{-w} must begin at multiples of D^{-w}, an interval of this largest possible length will not always fit inside a source interval. This Diophantine problem is illustrated for the binary case in Figure 1.1. A sequence interval of length $2^{-\lceil -\log_2 p \rceil}$ fits inside the source interval on the left, but no sequence interval of this length fits inside the source interval on the right. There will always be at least $D - 1$ sequence intervals of length $D^{-\lceil -\log_D p \rceil - 1}$ inside any source interval of length p. Thus, we can be sure that

$$-\log_D p \leq w < -\log_D p + 2$$

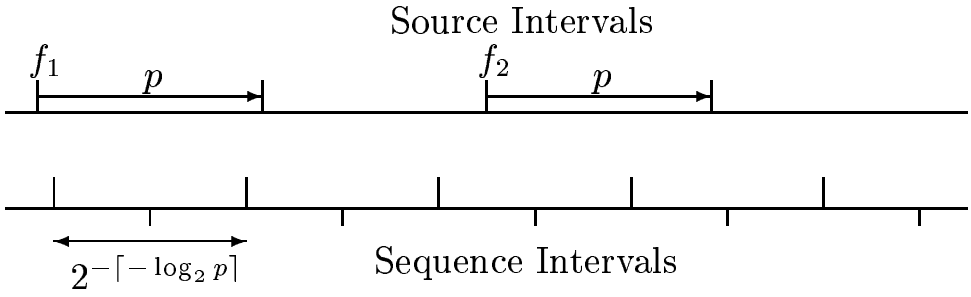


Figure 1.1: Choosing the longest sequence interval

where w is the length of the codeword assigned to the block. Averaging over all blocks, we obtain

$$\frac{H(X_1 \ldots X_L)}{\log_2 D} \leq E[W_B] < \frac{H(X_1 \ldots X_L)}{\log_2 D} + 2,$$

where $E[W_B]$ is the expected codeword length when encoding the source sequence $X_1 \ldots X_L$ and where $H(.)$ denotes the entropy of a random variable in bits. Using the notation

$$H_L(X) \stackrel{\text{def}}{=} \frac{1}{L} H(X_1 \ldots X_L),$$

we obtain

$$\frac{H_L(X)}{\log_2 D} \leq E[W] < \frac{H_L(X)}{\log_2 D} + \frac{2}{L} \tag{1.1}$$

for this coding method, where $E[W]$ is the average expected codeword length per source symbol.

Now that we have seen how blocks are encoded in arithmetic coding, we are ready to discuss the next fundamental constituent of this coding method.

The cumulative probability can be computed recursively.

Note that each codeword depends only on the cumulative probability of the source block to which it is assigned and on the probability of that block. Therefore, there is no need to compute the complete table of all block probabilities, as we did in the example, when we are interested in determining only the codeword for a particular source sequence. For any discrete stationary source, the required block probabilities can be

computed recursively from the symbol probabilities. The recursion rules for a discrete memoryless source are:

$$P(x_1 \ldots x_L x_{L+1}) = P(x_1 \ldots x_L)P(x_{L+1}) \quad (1.2)$$

$$F(x_1 \ldots x_L x_{L+1}) = F(x_1 \ldots x_L) + P(x_1 \ldots x_L)F(x_{L+1}), \quad (1.3)$$

where $P(.)$ denotes the probability of the indicated block and $F(.)$ denotes its cumulative probability. For discrete stationary sources with memory, the marginal probabilities and cumulative probabilities in (1.2) and (1.3) have to replaced by the appropriate conditional probabilities and cumulative probabilities. In both cases, the recursion starts with the probability and the cumulative probability of the empty sequence, whose values by way of convention are 1 and 0, respectively.

Example: Returning to the binary memoryless source with probabilities $p_0 = .25$ and $p_1 = .75$ of our previous example, we suppose that we wish to find the 10-ary codeword for the length $L = 5$ source block 10011. The

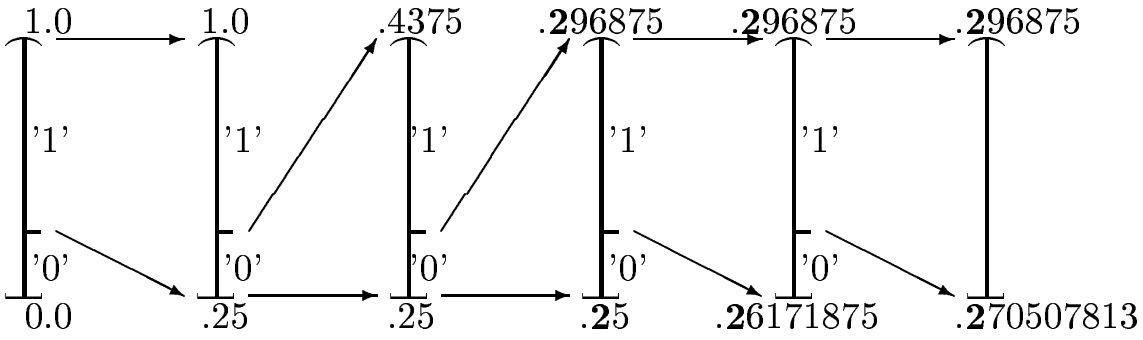


Figure 1.2: Computing a source interval

process of narrowing down the source intervals recursively by multiplying their length by the probability of the next source symbol is represented in Figure 1.2. The codeword obtained is '28'. Note that the first code digit '2' is already determined after the third step because the first digit of the decimal expansion of both endpoints of the codeword interval has there been determined to be a '2'. When the source interval is narrowed so that its interior contains no multiple of D^{-1}, the encoder can output the first code symbol and can then re-scale the source interval by dropping the first digit followed by a multiplication by D. In this example, the encoder could have proceeded after the third step with the interval $[.5, .96875)$ rather than $[.25, .296875)$. Thus, the encoding becomes a recursive process and we do not need to wait until the end of a source block before outputting digits of the codeword.

This process is even simpler in the practical binary case, where a code digit is determined whenever 1/2 is not an interior point of the source

interval. The rescaling operation when the source interval lies below 1/2 simply multiplies the interval endpoints by 2. Rescaling a source interval which lies above 1/2 is equivalent to multiplying the endpoints of the interval by two, then subtracting one. This allows us to implement the rescaling operation in any radix arithmetic that we choose. To illustrate this, we now determine the binary codeword for the same source block 10011, but using decimal arithmetic. The recursive process is represented in Table 1.3. We must add code digits to specify a longest sequence interval

Input	Interval	Operation	Output
	[0, 1)	get input	
1	[.25, 1)	get input	
0	[.25, .4375)	$\times 2$	0
	[.5, .875)	$\times 2 - 1$	1
	[0, .75)	get input	
0	[0, .1875)	$\times 2$	0
	[0, .375)	$\times 2$	0
	[0, .75)	get input	
1	[.1875, .75)	get input	
1	[.328125, .75)		

Table 1.3: Binary encoding of the source sequence 10011

inside the source interval remaining at the end of the recursive process. In this case, the code digits '10' specify the sequence interval [.5, .75) which lies inside [.328125, .75). The codeword obtained is thus '010010'.

The decoder performs almost the same operations as the encoder. At every step, it computes the source intervals for all possible single-digit extensions of the current source block using (1.2) and (1.3), then decides which symbol was actually encoded based on which of these source intervals contains the sequence interval of the received codeword.

We are ready to proceed to the third and last component of our practical implementation of an arithmetic encoder.

Implementation with finite-precision arithmetic.

Practical implementations of source coding techniques must use finite-precision arithmetic. Although the rescaling introduced above slows down the increase in required precision somewhat, the number of digits required to specify interval endpoints will still increase steadily in the coding method

just described. The algorithm can be approximated using integer arithmetic, where the interval endpoints are rounded inwards to the nearest integers. The endpoints must be rounded inwards to ensure that the intervals will never overlap, which in turn guarantees prefix-freeness as discussed earlier. The upper bound in (1.1) assumes infinite-precision arithmetic. Suppose however that the smallest interval that can be represented with finite precision is of size ϵ, $\epsilon > 0$. For example, if 32-bit integers are used to represent the interval endpoints, then $\epsilon = 2^{-32}$.

If we assume that $0 < \epsilon < p_i/2$ for all i, we can write a new upper bound for the expected average codeword length per source digit when encoding a very long block from a discrete memoryless source with probabilities $p_1, \ldots, p_N$ as

$$\lim_{L\to\infty} \frac{E[W]}{L} < -\sum_{i=1}^{N} p_i \log_D(p_i - 2\epsilon),$$

because the rounding error which can occur on each endpoint of the source interval is at most ϵ and hence the actual multiplier for the interval length, which ideally would be p_i, is at least $p_i - 2\epsilon$. We note that

$$\begin{aligned} -\log_D(p_i - 2\epsilon) &= -\log_D \left[p_i \frac{p_i - 2\epsilon}{p_i} \right] \\ &= -\log_D p_i + \log_D \frac{p_i}{p_i - 2\epsilon} \\ &< -\log_D p_i + \left[\frac{p_i}{p_i - 2\epsilon} - 1 \right] \log_D e = \\ &= -\log_D p_i + \frac{1}{\ln D} \frac{2\epsilon}{p_i - 2\epsilon} \end{aligned}$$

where we have used the inequality $\log r \leq (r-1)\log e$ which holds for $r > 0$ with equality if and only if $r = 1$. Thus, we see that

$$\lim_{L\to\infty} \frac{E[W]}{L} < H(X) + \frac{1}{\ln D} \sum_{i=1}^{N} p_i \frac{2\epsilon}{p_i - 2\epsilon} \leq H(X) + \frac{1}{\ln D} \frac{2\epsilon}{p_{\min} - 2\epsilon}, \tag{1.4}$$

where $p_{\min} = \min_i p_i$. If $2\epsilon << p_{\min}$, the last term in (1.4) is negligible and the average expected length per source symbol approaches the source entropy. A similar upper bound holds for stationary non-memoryless sources, replacing probabilities by the appropriate conditional probabilities.

If $2\epsilon \geq p_{\min}$, the length of a codeword becomes infinite when the source sequence is such that rounding causes the source interval to vanish.

When the length of the source interval is zero, the corresponding codeword interval also has length zero which, by our definition of sequence intervals, implies that the length of the codeword is infinite. It follows that when $p_{\min} \geq 2\epsilon$, there can be no finite upper bound for the average codeword length.

Apart from rounding, there is an additional effect that may cause us to lose precision when using integer arithmetic. This effect will be called "straddling". It occurs when the source interval contains the same multiple of D^{-1} for several steps. During these steps, no rescaling is performed and no output is generated as the encoder waits to find out whether the source interval will eventually lie above or below this multiple of D^{-1}. This causes a loss of precision as the size of the source interval decreases steadily. To avoid this loss of precision, a new procedure can be introduced. Whenever the interior of the source interval contains a multiple of D^{-1} but no other multiple of D^{-2}, the encoder knows that it will need to perform an additional rescaling operation as soon as straddling is resolved. Instead of waiting for straddling to be resolved, the encoder can perform this additional rescaling operation immediately by simply dropping the digit corresponding to D^{-2} in the radix-D representations of the interval endpoints. When straddling is finally resolved, i.e., when the interior of the source interval has gravitated so as to lie completely above or completely below the multiple of D^{-1}, a '0' code digit must be inserted or a 'D-1' code digit must be inserted, respectively, after the digit just now determined.

Example: In a decimal encoder, suppose that the current source interval is $[.1932, .2046)$. This interval straddles .2 within $D^{-2} = .01$ on each side. Thus, the interval is rescaled to give $[.132, .246)$. Suppose that the next source digit causes the interval to be $[.132, .189)$. Straddling has been resolved and the encoder outputs the digit '1' and rescales the interval to $[.32, .89)$. Before continuing, the encoder must output a '9' because the interval $[.132, .189)$ lies below the point .2 that was straddled. This rescaling may be repeated several times before straddling is resolved and the appropriate digit will then be inserted as many times as the rescaling was performed.

It is important to notice that this procedure only modifies the order in which operations are performed by the encoder, but the encoder will eventually compute the same source interval with this new procedure as it would have computed without it. Therefore, no loss of precision results from this procedure[1] .

[1] In practice, the counter used to count the number of straddling operations performed can only count up to a finite number. The maximal value of this counter also

In the binary case, straddling occurs when the source interval includes 1/2. The rescaling is performed when straddling occurs and the interior of the source interval lies entirely between 1/4 and 3/4. The operation of dropping the digit corresponding to 2^{-2} from the endpoints of the interval is equivalent to multiplying these endpoints by 2 and subtracting 1/2.

Example: Consider a ternary memoryless source with alphabet $\{a, b, c\}$ and probabilities $p_a = .2$, $p_b = .5$ and $p_c = .3$. Using 4-bit integer arithmetic, we would like to determine the binary codeword for the source sequence a, b, b, a, c, b, c. To implement the encoder using 4-bit integers, i.e., the integers 0 to 15, we let the non-zero integers represent the fractional numbers 1/16 to 15/16. Both the real numbers 0.0 and 1.0 will be represented by the integer 0, which causes no ambiguity because 0.0 can occur only as the lower endpoint and 1.0 can occur only as the upper endpoint of an interval[2]. The points 1/4 = 4/16, 1/2 = 8/16 and 3/4 = 12/16 are represented by the integers 4, 8 and 12 respectively. The resulting encoding is shown in Table 1.4, where, for clarity, we have shown upper endpoints of 1.0 by '16' rather than by '0' as would actually be used in the implementation. To terminate the encoding of the source block, we need to fit a longest sequence interval within the resulting source interval [4, 16). This is achieved by adding a code digit '1', which restricts the sequence interval to [8, 16). The resulting codeword is 0001100011.

We now give the pseudo-code of an algorithm implementing a finite-precision binary arithmetic encoder for a discrete memoryless source:

```
1)    high := one ; low := zero ; S := 0

2)    input x ; range := high - low
      high := floor[ low + range * (F(x) + P(x)) ]
      low  := ceiling[ low + range * F(x) ]

3)    if ( (high > 3quarters AND low < half) OR
           (low < 1quarter AND high > half) ) goto (2)
```

---

determines the greatest number of code digits which can be generated in one coding operation. Thus, this value has implications on the size of the input and output buffer which are implemented in the encoder and on the maximal decoding delay of the decoder. When the straddling counter reaches its maximal value, the encoder intervenes by shifting one of the interval endpoints inwards to enforce a resolution. Since this is a very unlikely event, its influence on the expected codeword length can safely be neglected.

[2]It is also possible to represent the source intervals by storing their left endpoint and their integer width minus one.

Input	Interval	Straddle Counter	Operation	Output
	[0, 16)	0	get input	
a	[0, 3.2)	0	round	
	[0, 3)	0	$\times 2$	0
	[0, 6)	0	$\times 2$	0
	[0, 12)	0	get input	
b	[2.4, 8.4)	0	round	
	[3, 8)	0	$\times 2$	0
	[6, 16)	0	get input	
b	[8.4, 14.4)	0	round	
	[9, 14)	0	$\times 2 - 16$	1
	[2, 12)	0	get input	
a	[4.0, 9.0)	0	round	
	[4, 9)	0	$\times 2 - 8$	
	[0, 10)	1	get input	
c	[7.0, 10.0)	1	round	
	[7, 10)	1	$\times 2 - 8$	
	[6, 12)	2	$\times 2 - 8$	
	[4, 16)	3	get input	
b	[6.4, 12.4)	3	round	
	[7, 12)	3	$\times 2 - 8$	
	[6, 16)	3	get input	
c	[13.0, 16.0)	3	round	
	[13, 16)	3	$\times 2 - 16$	1, 0, 0, 0
	[10, 16)	0	$\times 2 - 16$	1
	[4, 16)	0		

Table 1.4: Finite-precision binary encoding of the source sequence a, b, b, a, c, b, c

```
4)    if (high <= half) high := 2*high
                        low := 2*low
                        output '0'
                        repeat S times [output '1']
                        S := 0 ; goto (3)

5)    if (low >= half)  high := 2*high - one
                        low := 2*low - one
                        output '1'
                        repeat S times [output '0']
                        S := 0 ; goto (3)

6)    high := 2*high - half ; low := 2*low - half
      S := S+1 ; goto (3)
```

In the program, the words 'zero', '1quarter', 'half', '3quarters' and 'one' stand for the integers representing 0.0, 1/4, 1/2, 3/4 and 1.0, respectively. This algorithm does not include the termination to find the last digits of the codeword: when step (2) is reached and no further input is available from the source, the algorithm stops and the codeword must be completed.

1.2 The History of Arithmetic Coding

We have seen that arithmetic coding relies on three ideas:

1. codewords serve to identify the cumulative probability of a source block,
2. the cumulative probability of a block can be computed recursively, and
3. the algorithm can be implemented approximately using finite-precision arithmetic.

We now seek the origin of each of these ideas.

Who invented the three ideas?

Of these three ideas, the first is the most subtle. We usually think of an encoder as a function that assigns codewords to source blocks. We draw trees to represent a code and judge the performance of an encoder by the expected path length in the tree. Instead, we are now told to use the D-ary representation of the cumulative probability of a block as a basis for

its codeword. This seems very different from the "mainstream" of coding and information theory. However, this idea comes right out of Shannon's 1948 paper [1], Section 8. Shannon proposes to use the binary expression for the cumulative probability truncated to $\lceil -\log p \rceil$ positions, as a codeword, where p is the probability of the block. In Shannon's method, the blocks must be ordered in order of decreasing probabilities rather than lexicographically to compute the cumulative probabilities. This guarantees that there will always be a codeword interval of size $2^{-\lceil -\log_2 p \rceil}$ available. Therefore, Shannon's method achieves an upper-bound on $E[W]$ of $H_L(X) + 1/L$, which is better than the bound (1.1) for arithmetic coding.

According to Shannon, the code obtained by his method is essentially equivalent to the code obtained by Fano's construction [11]. For this reason, both methods have become better known as "Shannon-Fano coding". In his comparison of the two methods, Shannon calls his method an "arithmetic process", which is the origin of the name we use today. However, Fano's construction requires a list of all the block probabilities in order to generate a codeword. Huffman's algorithm, which generates the optimal code for a given probability distribution, also requires a list of all the probabilities. Forming blocks of source symbols enables one to approach the entropy of a discrete stationary source as closely as desired using any of the methods named. However, there is a practical limitation to the block length for any of the constructive algorithms since the number of possible blocks increases exponentially in the block length. Shannon's original method also requires the computation of all the block probabilities so that they can be ordered for the cumulative distribution. The use of only one extra code digit per source block allows the blocks to be ordered lexicographically, thereby opening the way for a recursive computation of the cumulative probability. In this way, large block lengths become practical as we must compute only the cumulative probability of the actual source output block and need not worry about other potential output blocks. Arithmetic coding is used and implemented in practice for block sizes of several megabytes.

The idea to modify Shannon's original code in such a way as to compute the cumulative probability recursively is commonly attributed to Elias. Elias indeed mentioned the idea to Norman Abramson, who included it as a note in his book [26]. This is the earliest written reference to the approach. However, Elias denies having invented the method, although he was working on it at the time and was aware of its consequences. Gallager became aware of the idea at about the same time as Elias[3].

[3] I spoke with Peter Elias and Robert Gallager at M.I.T. on April 6, 1998, and discussed the origin of arithmetic coding with them.

In Gallager's words, "this idea was in the air at M.I.T. at the time and many people were working on it". Gallager believes that the idea was originally presented by Shannon as part of one of his lectures at M.I.T. Whether Shannon, Elias or Gallager invented arithmetic coding, one thing is clear: contrary to an impression shared by many today, this coding technique is not a new development. Indeed, it can be seen as a great classic of information theory since so many great names in the field have contributed to it. Perhaps calling it "Shannon-Fano-Elias-Gallager coding" or "Recursive Shannon-Fano Coding" would have done it a better service.

However, it was not until the early 1980s that arithmetic coding came into widespread use. The third idea we discussed, the realization using finite-precision arithmetic, was introduced in 1976 by Pasco [16] and by Rissanen [18]. This explains the relatively small impact of arithmetic coding prior to their work. With all their elegance, the first two ideas would not be of much use if the coding method they imply required infinite-precision arithmetic. Thus, in a certain way, arithmetic coding is indeed a relatively recent development.

Why did Shannon come up with the cumulative probability?

We have done our best to give a detailed description of arithmetic coding and its underlying principles. Still, though we have tried to explain the "How?" and the "How good?", we have not touched on the "Why?". In our view, no treatment of arithmetic coding would be complete without an answer to the question: "Why did Shannon think of using the cumulative probability as a codeword?". Though it is not possible to give a definitive answer, we will venture a guess[4].

Shannon stood before a problem which can be stated as follows: find a variable-length, prefix-free set of codewords to represent the output blocks of a source so as to yield a small expected codeword length. Huffman's algorithm is the direct, optimal solution to this problem. Though Shannon guessed there would be such a constructive solution, he never found it himself. His approach was founded on the realization that the output of an ideal source encoder should be a sequence of independent and uniformly distributed random variables. At first glance, this seems to be only a

[4]To my knowledge, there is no previous written report of this guess. However, I do not claim credit for it: this explanation was given to me by Léon Bottou of AT&T-Labs in the late hours of the night after consuming a considerable quantity of alcoholic beverage during a conference in Snowbird, Utah in March, 1998. I am indebted to Léon for having provided an explanation of what I always considered to be the unsolvable mystery of arithmetic coding.

side-effect of compression. On the other hand, Shannon realized that, while he could not find an ideal compression scheme, he knew how to design a device that would transform any random variable into a uniformly distributed random variable.

For continuous random variables, the cumulative density function can be used to produce a uniform random variable. Let X be a continuous random variable whose cumulative density function is $f_X(.)$, and let $Y = f_X(X)$. We can write

$$\begin{aligned} f_Y(y) &= \text{Prob}[f_X(X) < y] \\ &= \text{Prob}[X < f_X^{-1}(y)] \\ &= f_X(f_X^{-1}(y)) = y, \end{aligned}$$

where the second inequality holds only if $f_X(.)$ is strictly monotone, which is the case when the probability density function $p_X(.)$ is non-zero over the range $\mathcal{D}_X$ of X. In this case, Y is uniformly distributed over the interval $[0, 1]$. This explains why the cumulative probability appears in our source coding problem. The basic idea of arithmetic coding has less to do with a coding technique than with "probability equalization"[5]. When $f_X(.)$ is not strictly monotone over $\mathcal{D}_X$, as is the case for discrete random variables, Y has an "approximately uniform distribution" over the set $f_X(\mathcal{D}_X)$.

The invention of Shannon's probability-based coding versus Huffman's optimal constructive algorithm is a perfect example of what could be called Shannon's method of "bypassing the difficulty" to solve engineering problems. Prompted by his mentor Fano, Huffman took a direct approach to the problem of coding a random variable and found the optimal solution by devising a new algorithmic technique. Shannon reduced the task to a problem of probabilities and discovered a solution which, while not optimal, can be applied recursively with the known advantages. There are many examples of such bypasses of difficulty in Shannon's work. One obvious bypass was his invention of typical sequences and random coding, which bypassed the still unsolved problem of giving a constructive proof of the noisy coding theorem. Another bypass, which comes closest to our example, is Bode and Shannon's trick for constructing the causal linear minimum-mean-squared-error filter versus Norbert Wiener's direct solution of the problem. In his lecture at the ETH Zürich, Massey illustrates the discovery of the Bode-Shannon trick with a small tale recounting how the idea may have occurred to the two scientists:

> Claude Shannon and his colleague Hendrik Bode were sitting in the cafeteria at Bell Labs, lamenting over the difficulty of un-

[5]This name was suggested by Claudio Weidmann of the EPF Lausanne.

> derstanding Wiener's proof, nicknamed the "yellow peril" due to its yellow cover[6]. Shannon remarked that there was a simple solution when the observation signal happened to be white noise. Whereupon Bode retorted that he knew how to transform any observed signal into white noise by applying a filter to it. Thus the Bode-Shannon trick was born, achieving the same effect as Wiener's complicated proof in a much simpler fashion.

Again, Shannon's contribution to this bypass was to demand a probabilistic transformation of the signal.

In a final attempt to illustrate Shannon's bypass technique, we tell a short tale of our own invention. Imagine two translators who are each given a text to translate into a foreign language of which they have only a moderate knowledge. Both translators have a fair notion of the grammar of this foreign language, but their vocabulary is restricted. The first translator grabs a dictionary and sets out to translate his text. Each time he stumbles across a word that he does not know the translation for, he looks it up in the dictionary and uses the optimal translation for the word. This first translator uses the "direct approach". He achieves the most accurate translation of the original. To a native speaker of the language, his translation will sound correct, but slightly unnatural. The second translator refuses the dictionary she is offered and sets out to translate the text using only the vocabulary she already knows. Each time she encounters a word that she does not know a translation for, she finds a description in her own words. Her translation will be less accurate and, to a pedantic native speaker, may sound a little too simple. But her translation catches the flavor of the foreign language. The first translator uses the methods of Huffman or Wiener. The second translator uses Shannon's bypass method. She does not find the optimal solution to each problem, but her solution catches the real nature of the problem.

Following this excursion into the history and motivation of arithmetic coding, we will try to remain true to the tradition of Shannon bypasses of difficulty as we attempt to use arithmetic coding for noisy channels in the following sections.[7]

---

[6] this was the time of World War II, when the term "yellow peril" was more commonly used to refer to the Japanese enemy forces.

[7] When I first heard the story of Huffman's invention of the Huffman algorithm, my reaction was to think "How could Shannon not have thought of this simple algorithm himself?". Huffman's algorithm has found its way into the basics of information theory to such a wide extent that it seems almost a trivial idea seen by students like myself. I was sure that Shannon must have been very angry at himself when Fano told him of Huffman's discovery. After writing this chapter, I realize that Shannon was probably

1.3 Arithmetic Coding with Gaps

The field of channel coding has inspired considerably more research than has source coding. Codes have been developed for every conceivable channel, relying on mathematical subtleties well beyond the worst nightmares of any source-coding researcher. For such a researcher to venture into the field of channel coding is like an English chef trying to open a cooking school in France.

Nevertheless, we will attempt to construct a Shannon bypass for channel coding based on the approach to source coding that we assume gave birth to arithmetic coding. This will lead us to a modified arithmetic encoder for channels. Although this arithmetic encoder will seem fundamentally different from traditional block codes or convolutional codes, will will show that both of these coding techniques can be viewed as special cases of our arithmetic encoder.

A Shannon bypass for channel coding

In order to find our Shannon bypass, we need to define the channel coding problem and describe what its direct solution would be. We consider the situation when the output sequence of a discrete stationary source (DSS) must be transmitted over a discrete memoryless channel (DMC) without feedback. An encoder may be interposed between the output of the source and the input of the channel, and a decoder will provide the maximum a-posteriori (MAP) source sequence based on the output of the channel. The situation is illustrated in Figure 1.3. The channel input and output random variables are denoted as X and Y, respectively. The discrete memoryless channel is fully described by its transition probability distributions $P_{Y|X}(.|x)$ for all x. The capacity of the discrete memoryless channel can be computed as

$$C = \max_{P_X} I(X;Y)$$

and depends only on those conditional distributions. The channel coding problem appears in response to the promise made in the channel coding theorem. We state the theorem when the source encoded is a binary symmetric source. In this case, every source sequence is equally likely and the maximum a-posteriori decoder is equivalent to a maximum likelihood decoder. If the actual source to be transmitted is a discrete stationary

not angry at all, because he knew that he would never have found Huffman's solution as it is just not the type of solution he would have bothered to look for. This of course is pure speculation.

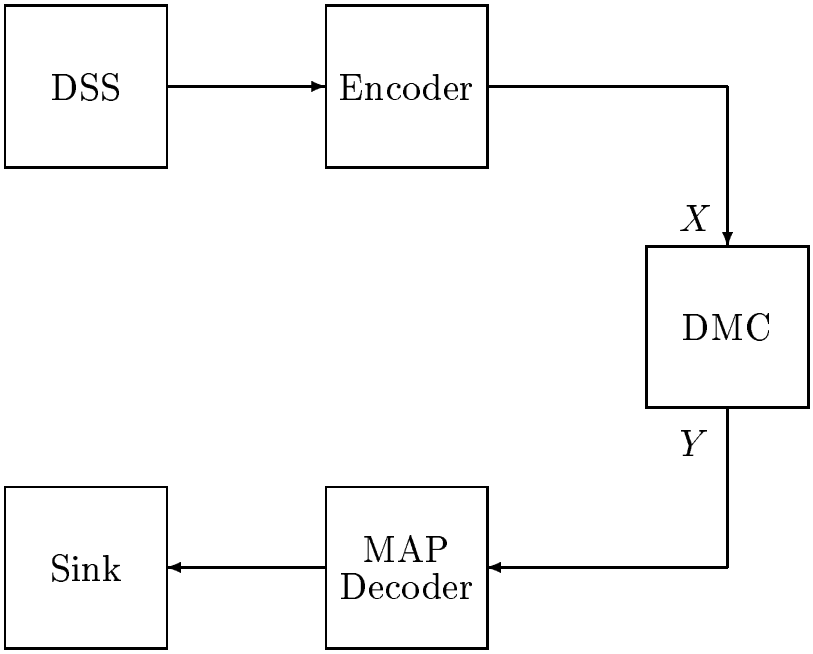


Figure 1.3: Situation Considered for Channel Coding

source, then source coding can be used to transform its output into essentially the output of a binary symmetric source.

Theorem 1.1 (The Channel Coding Theorem) *Given a discrete memoryless channel with capacity C, for every $R < C$ and every $\varepsilon > 0$, there exists an encoder of rate at least R and sufficiently large output block length N for the output of a binary symmetric source such that the maximum likelihood decoder will achieve a block error rate P_B satisfying*

$$P_B < \varepsilon.$$

The channel coding theorem gives rise to the following channel coding problem.

> Given ε and a discrete memoryless channel with capacity C, *find* an encoder whose rate R satisfies $R < C$ for the output of a binary symmetric source such that the block error probability of the maximum likelihood decoder is smaller than ε.

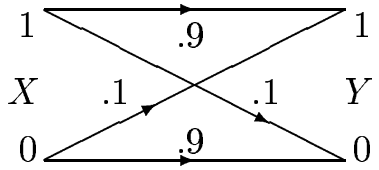
The following more specific statement of this channel coding problem for block coding could serve as a blueprint for its direct solution:

> Given a discrete memoryless channel with capacity C and given K and N such that $K/N = R < C$, find a mapping of K bits into N code digits such that the block error probability of the maximum likelihood decoder is small.

The solution of this problem is an encoder which fulfills the promise of the channel coding theorem. The "optimal" solution to this problem would be an encoder whose intricate structure ensures that the block error probability is minimized. However, no practical method to construct optimal codes is known. Moreover, an optimal code would not be of practical interest unless it also happened to be easy to encode and to decode.

We have stated the channel coding problem in a way similar to our statement of the source coding problem in the previous section. The Shannon bypass for source coding was based on the realization that the output of a perfect source encoder would be a sequence of independent and uniformly distributed random variables. A device that transforms the output of any source into such a sequence could also be expected to encode sources efficiently. Thinking along the same lines, we notice that, apart from the task of providing proper error protection, a channel encoder must also transform the output of the source into a sequence whose probability distribution corresponds to the capacity-achieving distribution of the channel. Pursuing this analogy, we could consider a Shannon bypass in which we design a channel encoder as a device to transform the probability distribution of the source into the capacity-achieving distribution, and hope that the error protection properties would follow automatically.

Example: The capacity-achieving distribution of the following binary symmetric channel with crossover probability $\varepsilon = .1$



is the uniform distribution $P_X(0) = P_X(1) = 1/2$, as it is for every binary symmetric channel. If the source to be encoded is a binary symmetric source, there is no need for an encoder since the source distribution is already the same as the capacity-achieving distribution.

Obviously, there is an important element missing in our Shannon bypass, since the bit error rate in our example will be 0.1 if we use no encoder. It is not enough to transform the probability distribution of the source into the capacity-achieving distribution. We have to modify our Shannon bypass to include the requirement to "dilute" the source output in order to achieve an information rate below the capacity of the channel.

Example: The capacity of the binary symmetric channel with crossover probability $\varepsilon = 0.1$ is $C_\varepsilon = 0.531$. Therefore, a code of rate $R = 1/2$ satisfies the requirement of the coding theorem that $R < C$. To encode

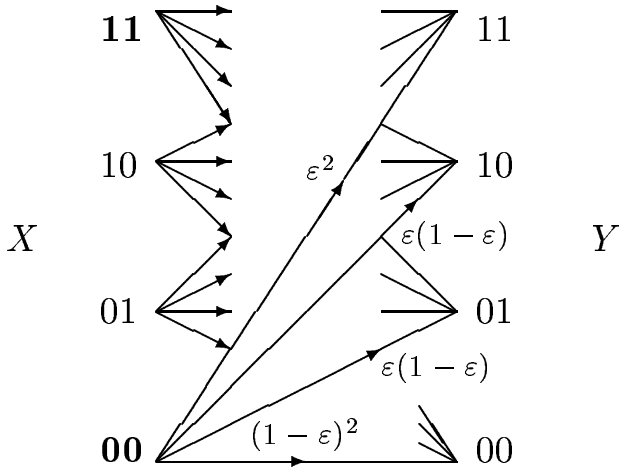
the output of a binary symmetric source, a (2,1) repeat code

$$\begin{aligned} 0 &\longrightarrow 00 \\ 1 &\longrightarrow 11 \end{aligned}$$

fulfills our design criteria: its rate is below capacity and the probability distribution for a single code digit is the uniform distribution.

Again, we find that our Shannon bypass is incomplete. The rate requirement and the probability transformation are both accomplished, but the code described in the example has no error-correction capability and the bit error rate remains 0.1. The problem is that, while the distribution for each code digit is indeed uniform, the code digits are not independent. Our block encoder has produced a code sequence whose letters are independent only when blocks of two code digits are considered as one letter. This suggests that we consider an "extension channel" which describes two consecutive uses of the binary symmetric channel.

Example: The extension channel resulting from two consecutive uses (i.e., the second-order extension channel) of the binary symmetric channel with crossover probability $\varepsilon = 0.1$ is the following discrete memoryless channel:



Its capacity is $C = 2C_\varepsilon$ and its capacity-achieving distribution is the uniform distribution over the four possible two-digit input blocks.

In terms of the extension channel, we can state our Shannon bypass as follows:

> Construct an encoder which encodes K source bits into N code digits such that the probability distribution of the code sequence corresponds to the capacity-achieving distribution of its N-th order extension channel.

This problem has no solution for most channels and the reason is obvious from the nature of a block code. For example, when encoding the output of a binary symmetric source for a binary symmetric channel using a block code, the encoder is a function from $\{0,1\}^K$ to $\{0,1\}^N$. Thus, the encoder cannot possibly achieve a uniform distribution over the input space of the channel. Therefore, we must be satisfied with a weaker statement of the Shannon bypass we are aiming for:

> Construct an encoder which encodes K source bits into N code digits such that the resulting probability distribution of the code sequence considered as an input distribution for the N-th order extension channel yields a mutual information between the channel input and the output that is maximum over all input probability distributions with 2^K non-zero terms.

This statement of the Shannon bypass suggests a definition which relates the capacity of a discrete memoryless channel to the dimensions of the block code used.

Definition 1.1 *The* M-input-constrained capacity C_M *of a discrete memoryless channel described by its transition probabilities* $P_{Y|X}(.|x)$ *for all* x *is defined as*

$$C_M \stackrel{\text{def}}{=} \max_{\substack{P_X \\ \#(suppP_X) \,=\, M}} I(X;Y), \tag{1.5}$$

where $\#(A)$ *denotes the cardinality of the set* A *and* $suppP_X$ *denotes the support*[8] *of* P_X*. The* (N,K) block-coding capacity $C_{(N,K)}$ *of a discrete memoryless channel is defined as the* 2^K*-input-constrained capacity* C_{2^K} *of its* N*-th order extension channel.*

The capacity C of a discrete memoryless channel gives an upper bound on the rate at which reliable communication is possible but gives no indication as to how an encoder is to be designed for the channel. The block coding capacity $C_{(N,K)}$ gives no such upper bound, because it is specific to the rate K/N. However, the block coding capacity is the true capacity of the channel induced by the block encoder, which is the N-th order extension channel where only 2^K input values are being used. In addition, the corresponding capacity-achieving distribution yields a list of the codewords which must be used by the block encoder and a list of codeword probabilities. In this respect, the task of the encoder reduces to

[8] For a real-valued function $f(.)$, the *support* of f is defined as the subset of its domain where f takes on non-zero values.

N	K	$C_{(N,K)}/N$	Codewords	Block Error Probability
2	1	.371	00 11	$P_B = \varepsilon$ $= .1$
3	1	.287	000 111	$P_B = 3\varepsilon^2 - 2\varepsilon^3$ $= .028$
3	2	.465	000 011 110 101	$P_B = 2\varepsilon - \varepsilon^2$ $= .19$
4	2	.389	0000 0011 1101 1110	$P_B = 1 - (1 + 2\varepsilon)(1 - \varepsilon)^3$ $= .1252$
5	2	.339	00000 00111 11001 11110	$P_B = 1 - (1 - \varepsilon)^3 \cdot$ $\cdot(1 + 3\varepsilon - 2\varepsilon^2)$ $= .0669$

Table 1.5: The (N, K) block coding capacities of a few extension channels of the binary symmetric channel and associated codes.

the task of transforming the probability distribution of the source into the codeword probabilities associated with the (N, K) block coding capacity of the channel. If the block coding capacity is achieved by more than one distribution, then either set of codewords and corresponding probabilities can be used.

Example: The (N, K) block coding capacities of a few extension channels of the binary symmetric channel with crossover probability $\varepsilon = .1$ and the associated codes are represented in Table 1.5. In all the cases represented, the distribution associated with the (N, K) block coding capacity is the uniform distribution over the codewords listed. Furthermore, this distribution is not unique and the uniform distribution over any essentially equivalent code (codes which can be obtained by permutation and complementation of the codes listed) also achieves the block-coding capacity.

It is computationally infeasible to compute the (N, K) block coding capacity of a discrete memoryless channel based on the direct approach used in the example for much larger values of N and K. In order to construct a practical channel coding method, we will take a step back in

the construction of the Shannon bypass which preceded Definition 1.1.

Hereafter in this chapter, we will restrict our attention to channels whose capacity-achieving distribution is the uniform distribution. As we have presented it, arithmetic coding is a method which transforms the output of any discrete stationary source into a sequence of essentially independent and uniformly distributed random variables. Therefore, the rate of the output sequence of an arithmetic encoder is close to 1. If the output sequence of an arithmetic encoder is to be transmitted over a noisy channel, we must enable the encoder to achieve a rate which is acceptable to the channel, i.e., a rate which is sufficiently below the capacity of the channel. Our hope is that, after modifying our arithmetic encoder to achieve the desired rate, the encoder will still provide an output distribution which is sufficiently close to the capacity-achieving distribution of the channel to ensure a small block error probability.

Shrinking the source intervals.

In an arithmetic source encoder, the length of the codeword obtained after processing L source digits and computing the corresponding source interval is the negative logarithm of the length of the largest sequence interval which fits inside the source interval. In order to modify arithmetic coding for noisy channels, we need to increase the length of each codeword so as to achieve the code rate required. This is equivalent to reducing the size of the codeword intervals. There are two ways that this could be realized:

1. After the source interval is determined in the usual manner, we can require the encoder to select a smaller sequence interval inside the source interval than it normally would. Using this approach, the first digits of the codeword for a given source sequence are the digits of the codeword which would have been generated by an arithmetic *source* encoder for the same sequence. Following these digits, the channel encoder adds a number of code digits by shrinking the codeword interval. This method can be seen to realize a type of systematic encoding.

2. Alternatively, we could shrink, or rescale the single-letter source intervals used at every recursion of the arithmetic encoder to compute the source interval corresponding to the source sequence. In this approach, we abide by the convention of choosing the largest sequence interval which fits inside the resulting source interval, but the size of that source interval is reduced because the single-letter source intervals that were used to compute it have been rescaled.

In practice, we will use a combination of the two approaches. We rely mostly on the second approach in the design of our channel encoder. We use the first approach to complete the codeword when a codeword length is specified but the second method has yielded an insufficient number of code digits. We now concentrate on analyzing the second approach stated.

The lengths of the single-letter source intervals in an arithmetic source encoder are equal to the probabilities of the source if the source is a discrete memoryless source, or to its conditional probabilities if the source is a discrete stationary source. In both cases, the picture of the single-letter source intervals covering the unit interval [0, 1) is a graphic representation of a probability distribution. If we now shrink the single-letter source intervals as described, the resulting interval lengths do not form a probability distribution since their sum is not equal to one: gaps appear between the rescaled source intervals. Figure 1.4 shows an example of

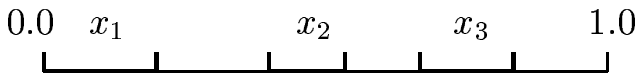


Figure 1.4: The rescaled source intervals for a ternary source

the rescaled source intervals and the corresponding gaps for a ternary memoryless source. The presence of gaps accounts for the name "arithmetic coding with gaps" which we use to describe this method[9].

To remedy the situation where the lengths of the single-letter source intervals do not form a probability distribution, we could think of the gaps as being the source intervals corresponding to dummy symbols which never appear in the source output. The lengths of the rescaled source intervals and the lengths of the gaps now form the probability distribution $P_{\tilde{X}}(.)$ which is used by the arithmetic encoder to encode a source whose true probability distribution is $P_X(.)$. Therefore, we will speak of "coding probabilities" to describe the probabilities $P_{\tilde{X}}(x)$ which correspond to rescaled source intervals and to gaps, while keeping in mind that the coding probabilities of the gaps will never be used in the actual encoding.

Example: For a binary memoryless source whose alphabet consists of the letters $\{x_1, x_2\}$ with probabilities $P_X(x_1) = 0.6$ and $P_X(x_2) = 0.4$, we write a table for the true probability distribution $P_X(.)$ and for the coding probability distribution $P_{\tilde{X}}(.)$ of the source symbols and of the dummy symbols when the source intervals are rescaled by a factor of $\gamma = 1/2$:

[9]The idea to introduce wide gaps between source intervals occurred to me while eating a piece of Emmentaler cheese, a swiss cheese in which large holes are formed naturally during the fermentation process.

x	$P_X(x)$	$P_{\tilde{X}}(x)$
x_1	0.6	0.3
d_1	0	0.2
x_2	0.4	0.2
d_2	0	0.3

For a discrete memoryless source, if every interval is rescaled by a factor of γ, then the information rate of the code, i.e., the entropy of a source letter divided by the average number of code digits per source letter, is

$$\begin{aligned} R &= \frac{H(X)}{-\sum_i p_i \log_D(\gamma p_i)} \\ &= \frac{H(X)}{H(X) - \log_2 \gamma} \log_2 D \text{ [bits/use].} \end{aligned} \tag{1.6}$$

When coding the output of a binary symmetric source, this equation reduces to

$$R = \frac{1}{1 - \log_D \gamma} \text{ [bits/use].} \tag{1.7}$$

An interesting property of this coding method follows immediately from (1.6):

> This coding method can achieve any rational or irrational rate R directly, without puncturing, by appropriate choice of the rescaling parameter γ.

Example: The arithmetic source encoder for a binary symmetric source with output alphabet 0, 1 is the identity encoder, which uses the source sequence as its code sequence. The parameters of the identity encoder are the source probabilities $P_X(0) = P_X(1) = 1/2$ and the cumulative probabilities $F_X(0) = 0$ and $F_X(1) = 1/2$. Without changing the cumulative probabilities, we replace the source probabilities by the coding probabilities $P_{\tilde{X}}(0) = P_{\tilde{X}}(1) = 2^{-\pi}$. The scaling factor is $\gamma = 2^{1-\pi}$ and we compute the rate of the encoder based on (1.7) to be $R = 1/\pi$.

In the example and in the derivation preceding it, we have silently assumed that the cumulative single-letter probabilities used by the source encoder would remain unchanged for the channel encoder. However, now that the single-letter source intervals have been rescaled, we are free to choose the cumulative probabilities by placing the rescaled intervals on the unit interval while making sure that they do not overlap. How do we

place the intervals to optimize the performance of the channel encoder? This is one question we will not find a satisfactory answer for, although it is certainly a central problem. Later in this chapter, we will investigate the effect of placing the intervals in the simulation of a simple encoder, but the last word is far from spoken on this matter.

We have obtained this method by constructing a Shannon bypass considering the channel coding problem from a purely probabilistic point of view and neglecting the aspect of error protection. However, we will try to give an intuitive explanation of how this method handles errors. This will allow us to obtain a notion of how these codes can be decoded.

The key to understanding how errors are handled by this encoder is to consider what happens when two identical encoders are fed with the same semi-infinite input sequence differing in only one symbol that occurs at the k-th step. As they reach the k-th step, both encoders will have computed the same source interval. In other words, the encoders will be in the same *state*. At the k-th step, the two encoders will select different sub-intervals of this source interval. If the encoders were using infinite-precision arithmetic and random irrational numbers had been chosen for all the cumulative probabilities, the probability that the two encoders would ever converge to the same state in subsequent rescaling operations is zero. Thus, this method is fundamentally a tree coding algorithm. It bears similarity to the error-free codes described by Elias in [2] in that the encoder must be an infinite-state machine. If the source sequence is never terminated to form a block, the inherent K and N of the encoder as stipulated by the coding theorem both become infinite. Nothing prevents such a code from achieving a zero probability of error, albeit with infinite decoding delay and infinite decoding complexity.

In practice, we must realize our encoder as a finite-state machine, even if the number of states is very large when double-precision integers are used. To understand how the precision of the arithmetic affects the Hamming distance between code sequences, consider the situation of our twin encoders. Let $[f_1, f_1 + p_1)$ be the source interval processed by our first encoder and $[f_2, f_2 + p_2)$ be the source interval of our second encoder following the k-th step. Now suppose that the semi-infinite input sequence of our first encoder following the k-th step will be such that its source interval will always include the point $f_1 + \theta p_1$ for some $\theta \in [0, 1)$. Such a θ always exists. Since the two encoders process the same information sequence following step k, the source interval for our second encoder will always include the point $f_2 + \theta p_2$. The semi-infinite code sequences generated by our twin encoders are the radix-D representations of the numbers $f_1 + \theta p_1$ and $f_2 + \theta p_2$, where D is the size of the code alphabet. The

numbers f_1, p_1, f_2 and p_2 must be expressed with finite precision. Therefore, their radix-D representations have a finite number of non-zero digits. However, θ is determined with infinite precision because of the recursive nature of the process. Thus, the radix-D representation of θ may have an infinity of non-zero digits. We leave it as an open question whether with finite precision θ can be an irrational number or whether it must be a rational number. Now if the paths of the twin encoders were to converge eventually, their semi-infinite code sequences would coincide from that moment onwards. Therefore, the radix-D representation of the number

$$\delta = (f_1 + \theta p_1) - (f_2 + \theta p_2) = (f_1 - f_2) + \theta(p_1 - p_2)$$

would have a finite number of non-zero digits. We see that this is necessarily the case when $p_1 = p_2$. The radix-D representation of the number $f_1 - f_2$ has at most as many non-zero digits as those of the numbers f_1 and f_2, which are given with finite precision. Thus, when the coding probabilities used are all equal, the Hamming distance between code sequences is upper bounded by the maximum number of non-zero digits of the radix-D representations of the cumulative probabilities used. However, when the probabilities are all different, the radix-D representation of δ may have as many non-zero digits as that of θ, which is determined with infinite precision. A very small difference between the coding probabilities is enough to engender this effect. It will be a good strategy in practice to introduce small differences between the coding probabilities even when the corresponding probabilities of the source encoded are equal, if the Hamming distance between code sequences is to be maximized.

We have seen that this coding algorithm is a kind of tree coding algorithm. Therefore, it seems natural to decode the output of the channel using a sequential decoder. In fact, the arithmetic decoder used for source coding can be viewed as a stack algorithm (see [32]) where the stack size is 1, as the decoder can always opt for the correct path at each step in the noiseless case. The next section will be devoted to deriving a metric for sequential decoding of an arithmetic channel encoder. This metric will be slightly different from the metric used for convolutional codes to reflect the fact that, because of the straddling effect discussed earlier, the arithmetic encoder may produce any number of code digits at each step.

Unfortunately, a sequential decoder is known to be computationally practical only for rates on the order of the cutoff rate of the channel or less (see [13]). We have attempted to modify the arithmetic decoder used in source coding to decode the output of a noisy channel, but this has led nowhere so far. A Viterbi decoder could be adapted to decode for the arithmetic encoder but would generally be impractical because of the

large number of states. We could reduce the number of states by reducing the precision of the arithmetic, but of course the performance of the codes would then degrade. Is it an eternal dilemma of channel coding that whenever a good code is really needed, it will be almost impossible to decode it, so that a weaker code must be chosen?

One important property of this encoder, both in its ideal infinite-state version and in its real finite-precision implementation, is its *causality*. Any error protection for a given information symbol can arise only from code digits that were generated after encoding the symbol. In other words, if we consider the block code obtained by terminating the encoding after N code digits have been generated, this code will have a very small minimum distance in general. If straddling occurs in the last information digits, the minimum distance with truncation may even be zero. Therefore, it will be necessary to encode a tail of known digits following the information digits. This is similar to the procedure used to terminate a convolutional code. We will return to the problem of terminating a codeword after discussing sequential decoding for arithmetic coding.

Before we introduce our metric for sequential decoding, we consider the relation between the codes generated by arithmetic coding and the codes used in block coding and convolutional coding.

Where it is shown that arithmetic coding includes block coding and convolutional coding as special cases.

The coding method presented here differs fundamentally from other coding methods such as block coding and convolutional coding. A block encoder performs a mapping of source sequences of length K into code sequences of length N. In our coding method, we use an arithmetic procedure to generate code digits progressively as the source sequence is being processed. As mentioned, the arithmetic encoder can be terminated after encoding some fixed number of source symbols. Due to the straddling effect, the termination may produce code sequences of varying lengths. We will use a trick, to be described later, to force the arithmetic encoder to produce a code sequence of a given length N. This will make the arithmetic encoder into a block encoder. The questions arise: what types of block codes can be generated by an arithmetic encoder? Can an arithmetic encoder generate "good" block codes?

The answer to these questions is not as difficult as might be expected. Consider the binary arithmetic encoder which uses the following two source intervals:

The coding probabilities are $P_{\tilde{X}}(0) = P_{\tilde{X}}(1) = 1/4$ and the cumulative probabilities are $F_X(0) = 0$ and $F_X(1) = .75$. The encoder starts with the sequence interval $[0, 1)$ for the empty sequence. If a '0' is encoded, it selects the source interval $[0, .25)$, rescales it twice, emits the code digits '00', and returns to the interval it started with $[0, 1)$. Similarly, if a '1' is encoded, the encoder emits the code digits '11' and returns to its original state. Since the encoder always returns to its original state, it will implement the simple mapping

$$\begin{aligned} 0 &\longrightarrow 00 \\ 1 &\longrightarrow 11 \end{aligned}$$

which is exactly the $(2, 1)$ binary repeat code. Any (N, K) block code can be implemented by selecting D^K source intervals of length D^{-N} such that the encoder will produce the corresponding blocks of code digits and return to the source interval $[0, 1)$ after each encoding operation.

Example: The binary block code consisting of the codewords {000, 011, 101, 110} can be implemented by an arithmetic encoder using four source intervals specified by the following table of cumulative probabilities and coding probabilities:

x	$F_{\tilde{X}}(x)$	$P_{\tilde{X}}(x)$
x_1	0.0	1/8
x_2	.375	1/8
x_3	.625	1/8
x_4	.75	1/8

By determining the indexing of the source symbols in our symbolic representation $\{x_1, x_2, x_3, x_4\}$ of the source alphabet, we can determine the assignment of source symbols to block codewords of the encoder.

There is no particular interest in implementing a block code in this fashion. However, this answers our question as to whether an arithmetic encoder can generate "good" block codes. The answer is that an arithmetic encoder can generate *any* block code. The question remains whether there are arithmetic encoders which are not equivalent to block codes and which will outperform block codes.

A convolutional encoder can be viewed as an encoder formed by a collection of block encoders in which the current block encoder is selected

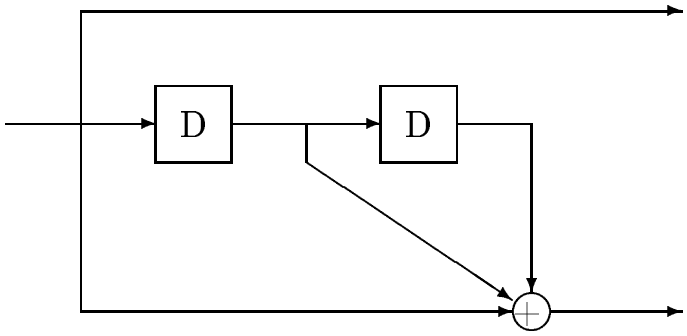


Figure 1.5: A Simple Convolutional Encoder

according to the value of a certain number of previous information digits. It is thus possible to implement a convolutional encoder as an arithmetic encoder in which the rescaled source intervals are assigned on the basis of a certain number of previous information digits. Such an arithmetic encoder uses the same approach as an arithmetic *source* encoder which is tailored for the output of a discrete stationary source, which bases its source intervals on the conditional probabilities of the source. When an arithmetic *channel* encoder places source intervals on the basis of previous information digits, the lengths of its source intervals are viewed as *conditional coding probabilities.* For example, the systematic convolutional encoder shown in Figure 1.5, which can be viewed as a collection of four (2,1) binary block codes, can be implemented using the conditional coding probabilities

rescaled source interval for		given
$X_k = 0$	$X_k = 1$	$X_{k-1}X_{k-2}$
$[0, .25)$	$[.75, 1)$	00
$[.25, .5)$	$[.5, .75)$	01
$[.25, .5)$	$[.5, .75)$	10
$[0, .25)$	$[.75, 1)$	11

To implement any (N, K) convolutional encoder of memory m using an arithmetic encoder, we must determine D^K conditional source intervals of length D^{-N} (where the size D of the coding alphabet must be a prime number in the case of a convolutional encoder) for each of the D^m possible states of the register, such that the equivalent conditional block code for each state corresponds to the mapping realized by the convolutional encoder in that state. Again, there is no practical interest in implementing a convolutional encoder in this fashion.

The arithmetic encoder for noisy channels presented in this chapter can be viewed as a general coding method that includes both block coding and convolutional coding as special cases.

1.4 A Metric for Sequential Decoding

A sequential decoder operates by exploring the tree whose vertices correspond to possible encoder states and whose branches are labeled with the encoded digits resulting from the transition between the states they connect. At every decoding step, the sequential decoder chooses one path to extend in the partial tree. This choice is based on a metric computed for each leaf of the partial tree. The essential characteristic of this metric is that it must enable one to compare encoder paths of unequal lengths. Fano postulated such a metric on intuitive grounds. In [15], Massey derived a metric based on the a-posteriori probability of the partial information sequence corresponding to a state given the complete received block. Massey showed that this metric is equal to Fano's metric for the maximum likelihood case, i.e., when the information digits are independent and uniformly distributed.

The purpose of this section is to derive a metric following Massey's approach which is suited for the arithmetic channel encoder we described. Without loss of generality, we restrict our attention to the case of a binary code alphabet. As in Massey's derivation, the expression for the a-posteriori probability of reaching a given encoder state given the received sequence will be derived. Massey's derivation relies on a simplifying assumption which does not apply exactly in the case of a convolutional encoder. Nonetheless, the metric he obtains, i.e., the Fano metric, works well in practice for decoding convolutional codes. We will make a similar assumption for arithmetic coding.

Simplifying Assumptions and Notation

Consider a sequential decoder which is about to examine a node which we refer to as the *current* node in the partial encoder tree. Massey's simplifying assumption can be stated roughly as follows: the code digits on the branches in the part of the code sequence beyond the current node in the encoder tree are independent and identically distributed (i.i.d.), which distribution is the distribution that achieves the capacity of the channel[10]. Massey considers an idealized situation where this assumption holds exactly, but the metric that he derives achieves excellent results in practice when used to decode a convolutional code. Therefore, we will adopt a similar assumption for our arithmetic encoder. However, we cannot directly apply this assumption to our problem, as the following example will illustrate.

[10] Massey forced me to write that he forgot to mention this latter part in his paper [15].

Example: We use the following rescaled source intervals to encode the output of a binary symmetric source:

Suppose that the encoder is about to compute the metric for an encoder state for which the current source interval is $[.49, .89)$ and straddling has occurred 4 times prior to reaching the current state. In this situation, it would be foolish to assume that the future of the code sequence is independent of the current encoder state. Given that the current source interval is "almost contained" in the interval $[.5, 1.0)$, we can reliably predict that the next 5 digits in the code sequence will be 10000. If no straddling had occurred, we could reliably predict that the next digit in the code sequence will be a 1.

Thus, we will always have to treat a portion of the future code sequence separately. This portion consists of at least one future code digit, plus as many code digits as will result from unresolved straddling operations. We call this portion of the code sequence the "postponed code digits" of the current node[11]. For the partial code sequence following the postponed code digits, we will revise Massey's assumption as follows: the code digits in the part of the code sequence which follows the postponed digits of the current node are independent and identically distributed, which distribution is the capacity-achieving distribution of the channel. For an arithmetic encoder, there is a strong case for making the assumption concerning the distribution of the code digits, since the encoder was designed mainly to approach the capacity-achieving distribution of the channel.

For the postponed digits, there are two possible code sequences depending on how straddling will be resolved in later encoding operations. We can compute the probability of those two code sequences in the previous example.

Example: (continued) We follow up the subtree of possible encoder states until straddling is resolved in each leaf. This process is represented in Figure 1.6, where the encoder operates in floating-point and without rescaling (see Figure 1.2), showing only the values of the postponed code digits of the current node on the branches where they are determined. The probability that the postponed code digits will be '01111' is equal to the probability of observing the sequence 'aaa' at the output of the binary symmetric source,

[11] Name offered to me by Jim Massey on the occasion of his 65th birthday.

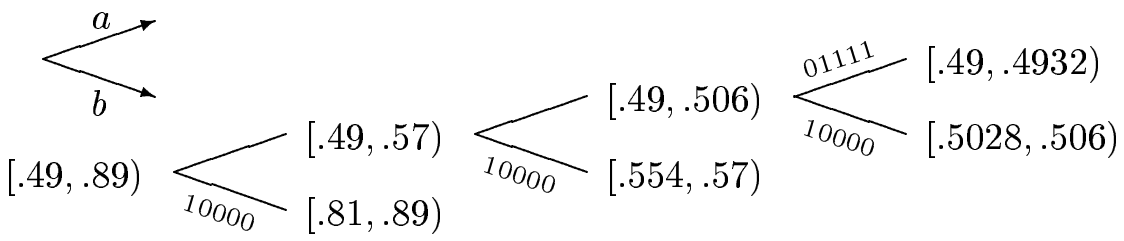


Figure 1.6: Determining the probabilities of the postponed digits

i.e., $P_0 = 2^{-3} = .125$. The probability of generating the postponed code digits '10000' is $1 - P_0 = .875$.

Unfortunately, it would generally be impractical to require a sequential decoding algorithm to look sufficiently far forward in the tree of encoder states to determine the probabilities of the postponed digits of the current node. Yet the metric that we will develop requires the use of these probabilities. Therefore, we adopt a new simplifying assumption concerning the probability of the postponed code digits and their dependence on the current encoder state: for a given encoder state, let $[l, h)$ be the source interval computed. The probability P_0 that straddling be resolved below $1/2$, i.e., that the postponed digits be equal to a 0 followed by 1's, is assumed to be equal to the fraction of the subinterval that lies below the critical point $1/2$, i.e.,

$$P_0 = \frac{\frac{1}{2} - l}{h - l}.$$

The probability that the postponed digits be equal to a 1 followed by 0's is necessarily then assumed to be $1 - P_0$.

In the previous example, this simplifying assumption yields $P_0 = (1/2 - .49)/(.89 - .49) = .025$. We computed the true P_0 and found it to be .125. This assumption will not give a close approximation of the true P_0 in general. In addition, the approximation will depend on how the shrunk source intervals have been placed in the unit interval. However, the fact that we only consider two possible outcomes for the postponed digits already makes for a considerable improvement compared to Fano's metric if we were to use it directly for our arithmetic encoder. This improvement has been confirmed by extensive simulation.

Before proceeding to derive the metric, we describe the setting and introduce some notation. We use a binary arithmetic encoder with gaps on the output of a discrete stationary source (DSS). The output of our encoder is transmitted through a binary memoryless channel (BMC) . The received sequence at the output of this channel is fed into the sequential

decoder. This situation is illustrated in Figure 1.7. When computing a

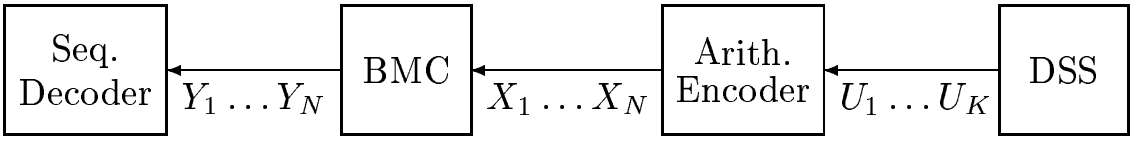


Figure 1.7: Situation for the variable-length decoding problem

metric in our sequential decoder, we are considering a node in the encoder tree where k information digits have been processed by the decoder. The following notation will be used for this situation:

$U_1 \ldots U_k$ denotes the k information digits that have been processed by the decoder at the node currently reached in the encoder tree.

$X_1 \ldots X_N$ denotes the complete output block of the encoder after all K information digits have been processed, of which the k information digits $U_1 \ldots U_k$ are but a prefix. This output block can be decomposed into the following three parts:

- $X_1 \ldots X_n$ denotes that part of the output block which is determined by the partial information sequence $U_1 \ldots U_k$.
- $X_{n+1} \ldots X_{n+S}$ denotes that part of the output block which is influenced by the current state of the encoder at time k. This consists of the next code digit plus the number $S-1$ of further postponed digits caused when the coding interval has straddled the middle of the unit interval.
- $X_{n+S+1} \ldots X_N$ denotes the tail of the output block which, following Massey, we assume to be independent of the information digits $U_1 \ldots U_k$.

$Y_1 \ldots Y_N$ denotes the complete received block at the output of the binary memoryless channel.

The Metric

The metric we will derive is defined as the logarithm of the a-posteriori probability of the k information bits given the complete received block, i.e.,

$$\mathcal{L} = \log P(u_1 \ldots u_k | y_1 \ldots y_N).$$

As in Fano's metric, we use the logarithm with the aim of obtaining an additive metric which can be computed recursively for each node based

on the metric of its parent node. The metric that we are about to derive will not achieve this property in the strict sense, but part of the metric can be computed recursively and another part will have to be computed individually for each node.

The a-posteriori probability of the information sequence to the current node can be written as

$$P(u_1 \dots u_k | y_1 \dots y_N) = \frac{P(u_1 \dots u_k, y_1 \dots y_N)}{P(y_1 \dots y_N)}. \tag{1.8}$$

We can expand the numerator in (1.8) as follows

$$P(u_1 \dots u_k, y_1 \dots y_N) = P(u_1 \dots u_k) P(y_1 \dots y_N | u_1 \dots u_k).$$

Using the notation introduced above, we can write the conditional probability in this expression as

$$\begin{aligned} P(y_1 \dots y_N | u_1 \dots u_k) &= P(y_1 \dots y_n | u_1 \dots u_k) \cdot \\ &\quad P(y_{n+1} \dots y_{n+S} | u_1 \dots u_k, y_1 \dots y_n) \cdot \\ &\quad P(y_{n+S+1} \dots y_N | u_1 \dots u_k, y_1 \dots y_{n+S}) \\ &= P(y_1 \dots y_n | x_1 \dots x_n) \cdot \\ &\quad P(y_{n+1} \dots y_{n+S} | u_1 \dots u_k) \cdot \\ &\quad P(y_{n+S+1} \dots y_N | u_1 \dots u_k) \\ &= P(y_1 \dots y_n | x_1 \dots x_n) \cdot P_S \cdot P(y_{n+S+1} \dots y_N) \end{aligned}$$

where we have used Massey's assumption and the fact that $U_1 \dots U_k$, $X_1 \dots X_n$ and $Y_1 \dots Y_n$ form a Markov chain. We have introduced the notation P_S for the probability of the "postponed digits" given the past information digits. We now derive P_S using P_0 as given by our simplifying assumption.

$$\begin{aligned} P_S &= P(y_{n+1} \dots y_{n+S} | u_1 \dots u_k) \\ &= \sum_{x_{n+1} \dots x_{n+S}} P(y_{n+1} \dots y_{n+S}, x_{n+1} \dots x_{n+S} | u_1 \dots u_k) \\ &= \sum_{x_{n+1} \dots x_{n+S}} P(y_{n+1} \dots y_{n+S} | x_{n+1} \dots x_{n+S}, u_1 \dots u_k) \cdot \\ &\qquad \cdot P(x_{n+1} \dots x_{n+S} | u_1 \dots u_k) \\ &= \sum_{x_{n+1} \dots x_{n+S}} P(y_{n+1} \dots y_{n+S} | x_{n+1} \dots x_{n+S}) \cdot \\ &\qquad \cdot P(x_{n+1} \dots x_{n+S} | u_1 \dots u_k) \end{aligned}$$

The second conditional probability in the sum is non-zero only when $x_{n+1} \dots x_{n+S}$ is equal to "011 ... 1" or to "100 ... 0", in which case it takes the values P_0 and $1 - P_0$ according to our assumption. Thus, we can continue

$$\begin{aligned} P_S &= P_{Y_{n+1}^{n+S}|X_{n+1}^{n+S}}(y_{n+1} \dots y_{n+S}|011 \dots 1) \cdot P_0 + \\ &\quad P_{Y_{n+1}^{n+S}|X_{n+1}^{n+S}}(y_{n+1} \dots y_{n+S}|100 \dots 0) \cdot (1 - P_0) \\ &= P_0 \cdot P_{Y|X}(y_{n+1}|0) \prod_{i=n+2}^{n+S} P_{Y|X}(y_i|1) + \\ &\quad (1 - P_0) \cdot P_{Y|X}(y_{n+1}|1) \prod_{i=n+2}^{n+S} P_{Y|X}(y_i|0) \end{aligned} \tag{1.9}$$

Finally, using (1.8) and the results of our derivation, we obtain

$$P(u_1 \dots u_k|y_1 \dots y_N) = \frac{P_S \prod_{i=1}^{n} P(y_i|x_i) P(u_1 \dots u_k)}{\prod_{i=1}^{n+S} P(y_i)}$$

Taking the logarithm of this expression, we obtain our metric

$$\mathcal{L} = \sum_{i=1}^{n} \log \frac{P(y_i|x_i)}{P(y_i)} + \log P(u_1 \dots u_k) + \log P_S - \sum_{i=n+1}^{n+S+1} \log P(y_i). \tag{1.10}$$

When the source considered is a discrete memoryless source, the metric becomes

$$\mathcal{L} = \sum_{i=1}^{n} \log \frac{P(y_i|x_i)}{P(y_i)} + \sum_{i=1}^{k} \log P(u_i) + \log P_S - \sum_{i=n+1}^{n+S+1} \log P(y_i). \tag{1.11}$$

As mentioned earlier, this metric is not strictly additive, in the sense that it cannot be computed for each node based on the metric of its parent node. However, the first two terms of (1.10) are identical with Massey's metric, while the following two terms correspond to the modification required to handle the postponed digits. Massey's metric can be computed recursively, but the additional terms corresponding to the postponed digits must be calculated individually for each node. Note that the probability P_S of the postponed digits must be taken into account even for nodes in which no straddling has occurred because the next output digit of the encoder depends on the current encoder state in the same way as the straddling digits do.

Our metric simplifies for the case where the source is a binary symmetric source and the channel considered is a binary symmetric channel with a crossover probability of ε. According to Massey's forgotten assumption, the received digits at the channel output will be independent and uniformly distributed. Massey's metric, which in this case is equal to Fano's metric, becomes

$$\mathcal{L}_{\text{Fano}} = w(e)\log\varepsilon + (n - w(e))\log(1-\varepsilon) + (n-k)\log 2, \tag{1.12}$$

where $w(e)$ denotes the Hamming weight of the error sequence. The additional term for the "postponed digits" is

$$\mathcal{L}_S = \log P_S + (S+1)\log 2. \tag{1.13}$$

1.5 Implementation and Results

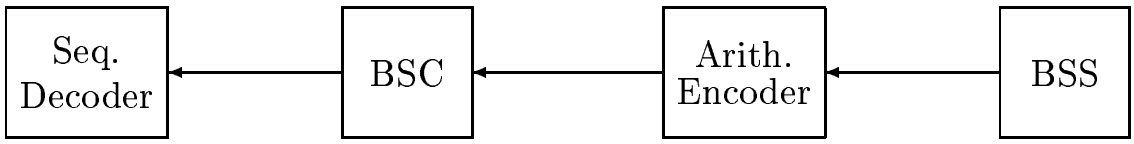


Figure 1.8: Simulation setup

In this section, we investigate the performance of an arithmetic encoder in the simulation of a simple communication system. The arithmetic encoder is applied to the output of a binary symmetric source (BMS). The output of the encoder is transmitted over a binary symmetric channel (BSC) whose output is forwarded to a sequential decoder. This situation is illustrated in Figure 1.8. The sequential decoder uses the stack algorithm described in [32]. Erasures are declared if a specified number of decoding steps have been performed without yielding a decoded block, where a decoding step is defined as one operation of extending the top path in the stack.

The aim of these simulations is not to measure the overall performance of arithmetic coding for noisy channels or to determine the choice of intervals for the best arithmetic encoder. Rather, we aim to obtain a first impression of the capabilities of this coding method, to evaluate the effect of the various parameters involved, and to perform a rough comparison with a few convolutional encoders, which are its closest relatives in this particular setup since they can also be decoded using a sequential decoder.

In all but one of the simulations, the shrunk source intervals of the arithmetic encoder are placed on the edges of the unit interval as indicated in the following picture:

The rate of the encoder is specified by (1.7) which can be written as

$$R = \frac{1}{1 - \log_2 2p} \text{ [bits/use]}. \tag{1.14}$$

It would seem natural to represent the performance of an arithmetic encoder by varying the rate and giving the measured probabilities of error in function of the rate for a given channel. However, since arithmetic coding is the only coding method known to us where the rate can be varied continuously, the performance of encoders is usually represented by fixing the rate and varying the crossover probability of the binary symmetric channel. We will stick to this convention.

It is a common practice in coding theory to represent the performance of a coding system by plotting the bit error rate versus the ratio E_b/N_0 of the energy per information bit to the noise spectral density. To convert the crossover probability of the binary symmetric channel to E_b/N_0 for a given effective code rate R_{eff}, we assume an additive white Gaussian noise channel with binary antipodal signaling. As will be explained shortly, the effective code rate of an arithmetic encoder is slightly lower than the code rate expressed in (1.14) because of the tail bits appended at the end of an information block. To avoid any confusion as to the units used, we include the conversion formulas that we used in an appendix to this chapter. Among others, those formulas show the meaning of the expression "one dB above the cutoff rate R_0", which may sound confusing since it is not the cutoff rate but E_b/N_0 which is expressed in dB.

We now explain the procedure used to add tail digits to the end of an information block and show simulation results which quantify the effect of the tail. The effect of the placing of the source intervals will be investigated in a simulation where the intervals are placed symmetrically at random for a given channel and a given rate. Finally, we compare the performance of an arithmetic encoder with the performance of three convolutional encoders.

Tail Digits and Termination

Like a convolutional encoder, an arithmetic encoder is a causal device, meaning that the code digits emitted by the encoder at any given time cannot depend on information digits which have not been processed yet. Therefore, if the code sequence is terminated immediately after the last information digits of a block have been processed, those last digits will

contribute very little to the code sequence and the probability of decoding them incorrectly will be very high. In other words, the Hamming distance between code sequences corresponding to information blocks which differ only in their last digits is small.

In order to overcome this shortcoming of causal encoders, tail digits which are known to the encoder and the decoder are appended at the end of an information block. For a convolutional encoder, it is common to append a number of zero digits equal to the memory m of the encoder so as to ensure that the encoder will have been driven back to its zero state after the last digit has been encoded. A sequential decoder considers a tree of possible encoder paths rather than a trellis, so it is not necessary to append m tail digits. Any number of tail digits up to m could be encoded. If the same tail sequence is encoded in every path of the encoder, it is not useful to encode more than m tail digits, as the resulting code sequences would not differ in the portion emitted after the first m tail digits have been processed. For the convolutional encoders presented shortly, we have used the conventional approach of encoding m tail zeros.

For an arithmetic encoder, the situation is slightly more complex due to two effects which we now describe briefly:

1. The encoder does not automatically generate a code sequence of the same length for each encoded block because of the straddling effect discussed earlier. Therefore, adding a fixed number of tail digits will not necessarily provide the same protection for the last digits of every information block encoded.

2. There is no obvious maximum for the number of tail digits when the same tail sequence is encoded in every encoder path. For example, if the source intervals are placed at the edge of the unit interval and a tail of zeros is appended to the information sequence, encoding the tail will effectively focus the source interval on the value of the lower endpoint it had attained after processing the information sequence. In this case, the number of significant digits in the finite-precision radix-2 representation of the lower endpoint of the source interval provides an upper bound for the number of useful tail zeros that can be appended. It is not clear what this upper bound would be if the source intervals were to be placed differently or if the tail does not consist only of zero digits.

In order to gain more flexibility in the length of the tail appended, we use a trick which circumvents the second effect stated. Incidentally, the same trick could also be used to allow longer tails for a convolutional encoder.

> We use a pseudo-random generator to generate a tail sequence for every information block, where the initial seed of the pseudo-random generator depends on the last digits of the information sequence. In each path of the code tree, the decoder can generate a sequence of tail digits which are identical to the tail digits the encoder would have generated in this path. The pseudo-random generator used is very sensitive to small variations of its seed value. Thus, the tail sequences encoded in neighboring paths in the code tree are expected to be independent.

Using this trick, we are free to use tail sequences of any length. To circumvent the first effect stated above, we use the following procedure:

> The length K of an input block and the length N of an output block are fixed, giving an effective code rate $R_{\text{eff}} = K/N$. The encoder will encode the information block followed by tail digits generated by the pseudo-random generator until it has generated a code sequence of length at least N. The first N digits of this code sequence are transmitted.

By using this procedure, we guarantee that the arithmetic encoder will always be able to generate a code block of length N for an information block of length K, except in the unlikely case when the encoder follows a path where straddling is never resolved. In practice, this case is excluded by requiring that the arithmetic encoder be given a maximal number of allowed straddling operations, after which it intervenes by shifting one endpoint of the source interval inwards to force a resolution.

In the procedure described, the effective length of the tail is determined by choosing the natural rate of the arithmetic encoder according to (1.14). Therefore, when the length of the tail is varied in our simulations, the effective code rate $R_{\text{eff}} = K/N$ remains constant. The effective length t of the tail is given as

$$t = N - \frac{K}{R} \tag{1.15}$$

where R is the natural rate computed in (1.14).

The metric used by the sequential decoder in the portion of the encoder tree corresponding to the tail digits must be modified to reflect the fact that the digits encoded are not the output of a binary symmetric source as is the case for the information digits. Since the digits encoded by the encoder are known to the decoder, the new metric is obtained by setting $P(u_i) = 1$ in (1.11) if u_i is the correct tail digit at position i and $P(u_i) = 0$ if it is not, in which case the metric is equal to $-\infty$. For

Input block size K	128
Output block size N	256
Effective rate R_{eff}	1/2
E_b/N_0	5.59 (1 dB above R_0)
Crossover probability ε	.0285
Stack size	64'000
Maximum steps before erasure	32'000

Table 1.6: Parameters for the simulation results in Figures 1.9, 1.10 and 1.11

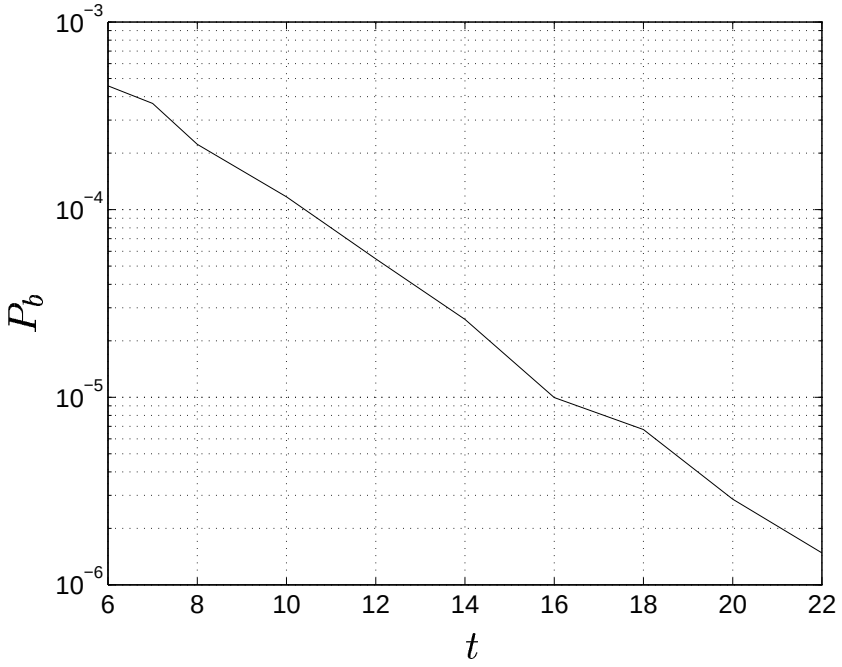


Figure 1.9: Bit error rate P_b in function of the tail size t

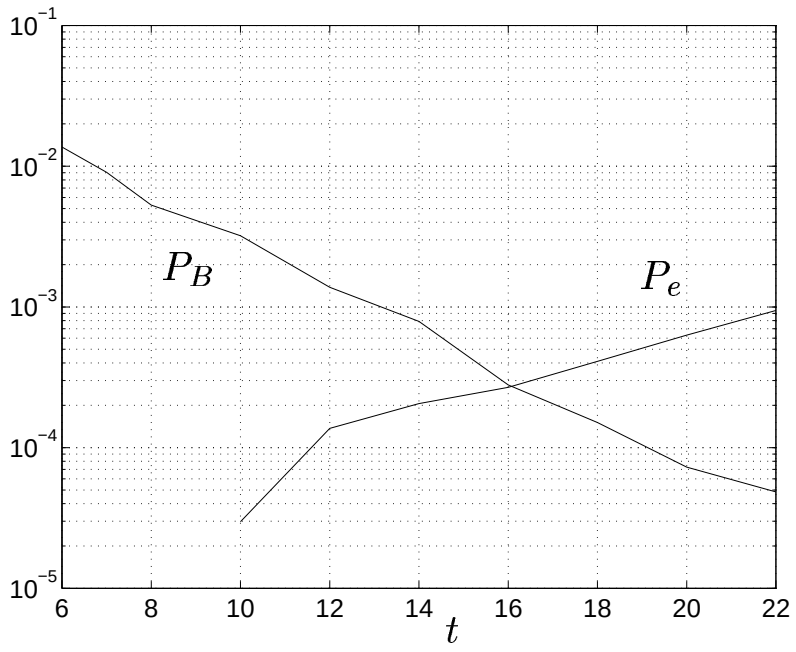


Figure 1.10: Block error rate P_B and probability of erasure P_e in function of the tail size t

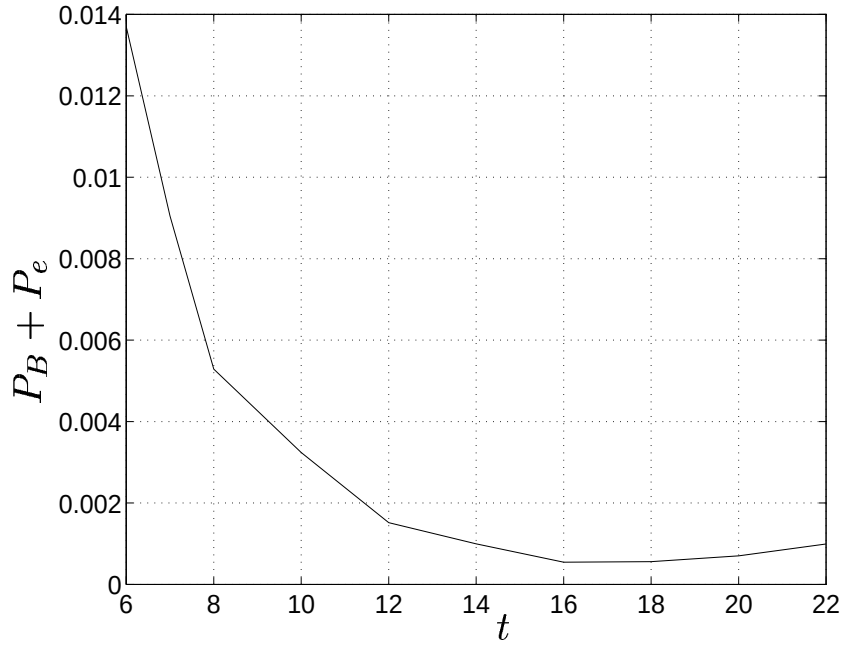


Figure 1.11: $P_B + P_e$ in function of the tail size t

a binary memoryless channel, the term in the metric corresponding to Fano's metric when u_i is the correct tail digit becomes

$$\mathcal{L}_{\text{Fano}} = w(e)\log\varepsilon + (n - w(e))\log(1-\varepsilon) + n\log 2, \tag{1.16}$$

and the term corresponding to the postponed metric remains the same as in (1.13).

Figure 1.9 shows the bit error rate measured in a simulation where the size of the tail was varied using (1.15) while all other parameters were kept constant. The parameters of this simulation are given in Table 1.6. The bit error rate gives an incomplete picture of the performance of the encoder because it does not take erasures into consideration. In all the results shown, an erased block *does not* contribute to the bit or to the block error rate. Figure 1.10 shows the block error rate P_B and the probability of erasure P_e measured in the same simulation. It appears that the encoder trades block errors for erasures as the tail size is increased. This tendency is analyzed in Figure 1.11 where the sum of P_B and P_e is plotted in function of the tail size t. As the plot shows, it appears that a tail length of 16 is virtually optimal in the sense that it minimizes the sum of P_B and P_e. This length depends on the parameters of the code, of the channel and of the decoder involved.

The effect of the placing of the source intervals

A simulation was performed in order to investigate the effect of how the source intervals are placed in the unit interval. Since the metric derived assumes that the intervals are placed symmetrically around the middle of the unit interval, we did not try to place the intervals asymmetrically. The tail size t was fixed at 16 which, given the effective rate of $1/2$ and the block sizes chosen, yields a source interval size of $p_0 = 0.272626933$ for both source intervals. The sum of the block error rate P_B and the probability of erasure P_e was measured in a series of 150 simulations using the parameters indicated in Table 1.7. In each simulation, a number x was selected at random between 0 and $1/2 - p_0$, and the source intervals were chosen to be $[x, x+p_0)$ for the source digit '0' and $[1-x-p_0, 1-x)$ for the source digit '1'. The resulting measurements are plotted in Figure 1.12 in function of the lower endpoint x of the left source interval.

On its left endpoint, the graph includes a measurement for the intervals $[0, p_0)$ and $[1-p_0, 1)$ which yields $P_B + P_e = 10^{-3}$. However, the measurement for the right endpoint of the graph corresponding to the source intervals $[1/2-p_0, 1/2)$ and $[1/2, 1/2+p_0)$ is not included in the graph because it yielded $P_B + P_e = 2.8 \times 10^{-2}$ which is an order of magnitude worse than

Input block size K	128
Output block size N	256
Tail size t	16
Effective rate R_{eff}	1/2
E_b/N_0	5.59 (1 dB above R_0)
Crossover probability ε	.0285
Stack size	32'000
Maximum steps before erasure	16'000

Table 1.7: Parameters for the simulation results in Figure 1.12

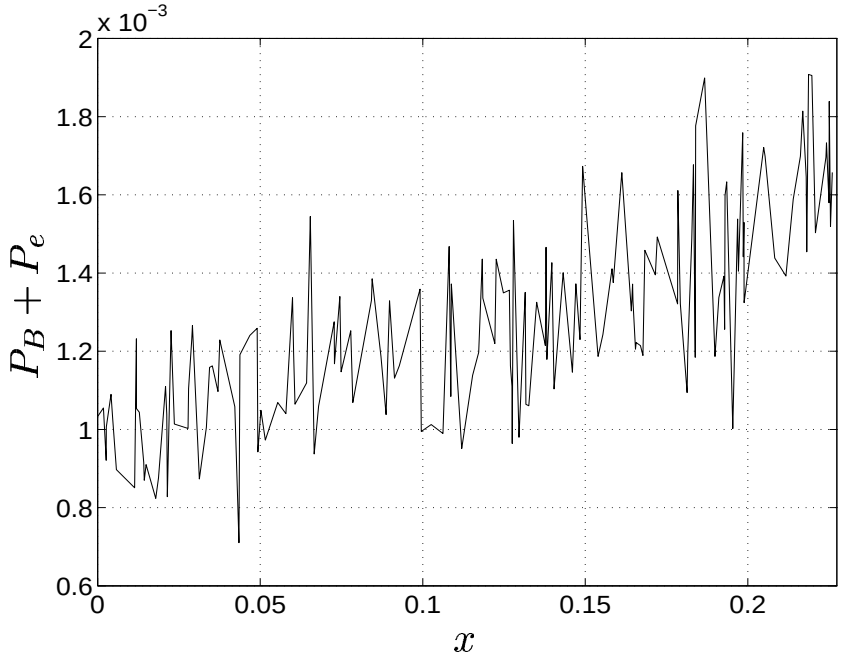


Figure 1.12: $P_B + P_e$ in function of the left endpoint x of the left source interval

any other measurement in the graph. The graph shows a slight tendency for better performance when the source intervals are placed wider apart on the unit interval. The best measurement in the graph was obtained for the intervals $[0.0434650148, 0.3160919478)$ and $[0.683908052, 0.956534985)$ which yielded $P_B + P_e = 7.1 \times 10^{-4}$.

Apart from excluding the bad choice of adjacent intervals, the results of these measurements encourage an exhaustive search for good arithmetic encoders. The performance of the encoder appears to be very sensitive to the choice of intervals. It may be that encoders using asymmetric intervals would perform even better than the encoders considered, but our metric would have to be modified for this case. There are endless variations of encoders that could be investigated, as for example encoders using more than two source intervals or encoders using conditional coding distributions. The results of the measurements performed provide a motivation to investigate further such variations.

Comparison of arithmetic coding with convolutional coding

A simulation was performed in order to compare the performance of an arithmetic channel encoder with the performance of convolutional encoders. Such a comparison is made difficult by the difference in the effective code rate between convolutional encoders, which varies as the memory of the encoder determines the tail length. Furthermore, in order to make a fair comparison between the bit error rates of various encoders when the a sequential decoder is used, the probability of erasure must be identical for the encoders considered.

Figure 1.13 shows the results of the simulation, whose main purpose was to compare the performance of an arithmetic encoder (curve 4) with the performance of the $R = 1/2$ non-systematic convolutional encoder of memory 12 with an optimum distance profile (curve 2) as specified in [32, Table 12.1D]. To ensure a fair comparison, the effective rate of the arithmetic encoder was chosen to match the effective rate of the convolutional encoder exactly, i.e., $K = 128$ and $N = 280$ for both encoders. Furthermore, the parameters of the arithmetic encoder were fine-tuned so that its probability of erasure matches the probability of erasure of the memory 12 convolutional encoder very closely over the whole range considered. This required that the tail size of the arithmetic encoder be set to $t_a = 17$ (as compared to the tail size $t_{c_{12}} = 2 \times 12 = 24$ of the memory 12 convolutional encoder) and that the sequential decoder for the arithmetic encoder be allowed a larger stack size and a greater number of computations be-

Convolutional Encoders	Curves 1, 2, 3
R_{eff} for memory 6	$\frac{128}{2*(128+6)} = .4776$
R_{eff} for memory 12	$\frac{128}{2*(128+12)} = .4571$
R_{eff} for memory 16	$\frac{128}{2*(128+16)} = .4444$
Stack Size	32'000
Maximum steps before erasure	16'000
Arithmetic Encoder	Curve 4
Input block size K	128
Output block size N	280
Tail size t	17
Effective rate R_{eff}	.4571
Stack size	64'000
Maximum steps before erasure	50'000

Table 1.8: Parameters for the simulation results in Figure 1.13

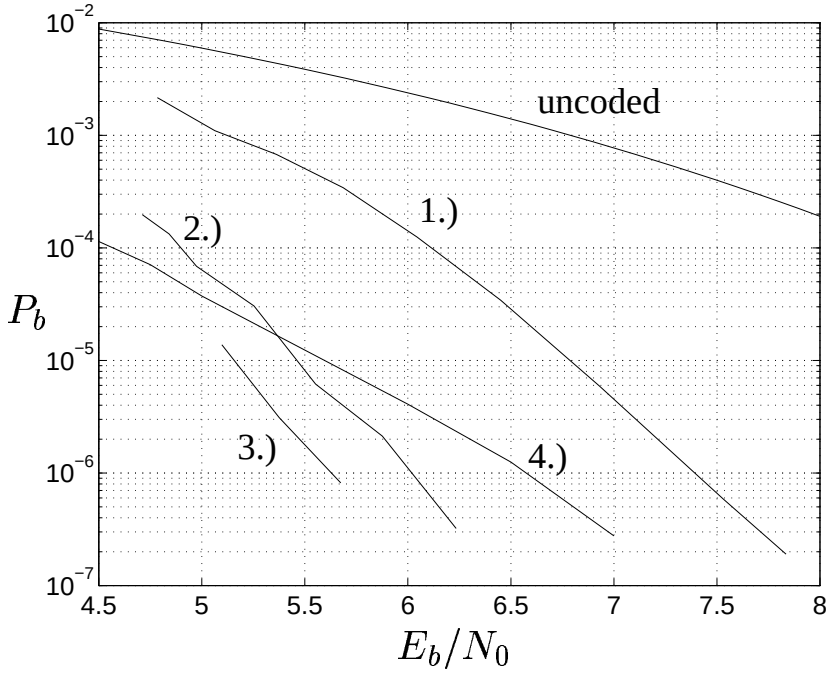


Figure 1.13: Bit Error Rate P_b in function of E_b/N_0 for convolutional encoders of memory 6 (1.), 12 (2.), 16 (3.) and for an arithmetic encoder (4.)

fore declaring an erasure. The resulting encoder parameters are shown in Table 1.8.

Figure 1.13 also shows the performance for uncoded transmission, and the performance of the $R = 1/2$ non-systematic convolutional encoders of memory 6 (curve 1) and of memory 16 (curve 3) with optimal distance profiles as specified in [32, Table 12.1D]. Since E_b/N_0 depends on the effective rate of transmission, each curve is automatically shifted according its effective rate of transmission. However, the probabilities of erasure for the encoders of various memory sizes do not coincide in general, and there are no erasures at all for uncoded transmission[12].

Based on the results of this simulation, we would argue that the performance of the arithmetic encoder compares well with the performance of the convolutional encoders tested. The comparison with the memory 12 convolutional encoder is particularly significant, since we have taken appropriate care to allow a fair comparison. In this respect, it should be pointed out that the convolutional encoder considered is the optimal encoder of the type and dimensions specified, while the arithmetic encoder has not been optimized in any way. The comparison shows that the bit error rate of the arithmetic encoder falls less steeply than that of the convolutional encoder as a function of E_b/N_0. On the other hand, the arithmetic encoder performs significantly better for low values of E_b/N_0. This gives a clear incentive to investigate the use of other decoders, better suited for very noisy channels than a sequential decoder, in conjunction with arithmetic channel encoders.

1.6 Discussion and Open Problems

We have presented a new coding method for noisy channels based on the well-known arithmetic encoder used in source coding. We summarize the characteristics of this coding method and state a few of the open problems surrounding it:

- The rate R of an arithmetic channel encoder can be any real number. Equation (1.6) relates the rate of the encoder to the factor by which the source probabilities are shrunk to yield coding intervals. The rate could even be modified continuously while encoding an information sequence.

[12] In addition, the memory 6 encoder attains its cutoff rate R_0 for $E_b/N_0 = 4.52$dB, the memory 12 encoder and the arithmetic encoders attain their cutoff rate for $E_b/N_0 = 4.45$dB and the memory 16 encoder attains its cutoff rate for $E_b/N_0 = 4.41$dB. Therefore, the memory 16 encoder has an advantage over the memory 12 encoder and the memory 6 encoder has a disadvantage.

- The encoding is specified by the endpoints of the coding intervals. These are real-valued parameters which can easily be tuned while encoding an information sequence. This may be an asset when considering this method for adaptive coding or for coding for channels with feedback. Indeed, there appears to be a close similarity between the coding method presented here and Horstein's coding algorithm for channels with noiseless feedback. This parallel has yet to be investigated.

- Arithmetic coding for noisy channels can be seen as a generalized coding method that subsumes both block and convolutional coding.

- When designing our method, we have specifically assumed that we were dealing with channels whose capacity-achieving distribution is the uniform distribution. If this were not the case, the use of an arithmetic decoder to transform the output distribution of an arithmetic channel encoder into the capacity-achieving distribution of the channel could be investigated.

- Our preliminary measurements of the performance of an arithmetic channel encoder indicate that this encoder may be particularly useful for channels with a low signal-to-noise ratio. It was not possible to investigate this more closely because of the limitations of the sequential decoder used. This suggests that one should search for other decoders which can be used in conjunction with an arithmetic channel encoder. In particular, the possibility of using a Viterbi decoder should be investigated.

- A thorough investigation of the code generated by an arithmetic channel encoder and how it relates to the parameters of the encoder would contribute considerably to the search for good arithmetic channel encoders.

- Though random coding arguments certainly apply to a general arithmetic encoder due to the inclusion of block coding and convolutional coding as special cases of this encoder, it would be interesting to develop such arguments specifically for arithmetic channel encoders.

Finally, since this channel coding algorithm is based on a source coding algorithm, it can be used for joint source and channel coding in a very natural way. In fact, we have always assumed that we were encoding the output of a discrete stationary source in our derivation of the encoder. The metric we derived for the sequential decoder is based on a-posteriori probabilities. Therefore, the metric and the decoder can easily be used for

joint channel and source decoding. To illustrate this aspect of the coding algorithm presented, we conclude this chapter with a tongue-in-cheek portrayal of the divisions currently separating the source coding community from the channel coding community and how they came to be.

Of coding for unicorns.

When Shannon wrote his paper in 1948, the "information theory community" consisted of himself and a few colleagues who were aware of his work. His paper had a few sections on coding for the Noiseless Channel and a few sections on coding for the Noisy Channel. For a few years, the same people were busy finding solutions to the coding problem for both channels. Then, as time went on, these became separate fields with hundreds of researchers working separately in each field. There are separate conferences on the topic of "source coding" and "channel coding" and separate sessions at Information Theory symposia. Researchers working in one field rarely visit the conferences and sessions devoted to the other field. This would be fine if the problems of source and channel coding were truly separable as many researchers in the two fields would like to believe. However, as we will show, the requirements of both fields are fundamentally incompatible. The position of a "source coder" can be summarized as follows:

> I search for the optimal method for coding and decoding a discrete stationary source for a noiseless channel. If the "channel coders" are doing their job properly, I can count on having a truly noiseless channel to code for.

The position of a "channel coder" can be summarized as:

> I search for the optimal way of coding and decoding the output of a binary symmetric source for a noisy channel. I can make the error probability as small as desired for any rate below the capacity of my noisy channel. If the "source coders" are doing their job, I can count on having a true binary symmetric source at the input of my encoder.

Both positions are based on an unrealistic assumption about the other field. The noiseless channel that the "source coder" expects to obtain from the "channel coders" is a channel with zero error probability. As any "channel coder" will tell him, this is not possible under most circumstances. On the other hand, while the "source coders" can turn the output of almost any source into a close approximation to the output of a binary symmetric source, they cannot decode what they encoded if there is even one error left

in the code sequence they receive. Indeed, the better their source coding algorithm, the more an error will propagate in general. If presented with this fact, our "channel coder" will mutter something like:

> These "source coders" must get a grip on the problem of error propagation.

But this seems to be as difficult as turning a noisy channel into a noiseless one. This dispute will continue for as long as the "source coder" and the "channel coder" don't finally collaborate and try to solve their common problems together.

It seems as if the arithmetic encoder that we presented provides an almost ideal setting for this collaboration. The distribution of tasks would be as follows:

1. Leave the task of *encoding* the source for the channel to us "source coders". We have an excellent algorithm for doing so and we are better at providing the input distribution and rate that the channel really needs.

2. Leave the task of *decoding* the channel output to the "channel coders". They have several excellent algorithms for implementing the optimal Bayesian decoder, while the arithmetic decoder we have devised is completely useless for anything but a truly noiseless channel or a *unicorn*.

Appendix: Conversion Rules between ε and E_b/N_0

The relation between E_b/N_0 and the crossover probability ε of the binary symmetric channel is given in [34] as

$$\frac{E_b}{N_0} = \gamma_b(\varepsilon, R_{\text{eff}}) = 10\log_{10}\text{erf}^{-1}\left[\frac{(1-2\varepsilon)^2}{R_{\text{eff}}}\right] \text{ [dB]} \tag{1.17}$$

for the additive white Gaussian noise channel when binary antipodal signaling is used. R_{eff} is the effective rate of the code obtained by dividing the length of an input block (excluding tail digits) by the length of an output block (including encoded tail digits). The function $\text{erf}^{-1}(.)$ is the inverse of the error function, which is defined as

$$\text{erf}(x) \stackrel{\text{def}}{=} \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

for $x \in [0, +\infty)$ and $\mathrm{erf}(-x) = -\mathrm{erf}(x)$.

The relation (1.17) can be inverted to give

$$\varepsilon = \varepsilon_{\gamma_b}\left(\frac{E_b}{N_0}, R_{\text{eff}}\right) = \frac{1}{2} - \frac{1}{2}\mathrm{erf}\left(\sqrt{R_{\text{eff}} 10^{\frac{E_b/N0}{10}}}\right).$$

The cutoff rate R_0 of a binary symmetric channel with crossover probability ε is

$$R_0(\varepsilon) = 1 - \log_2\left(1 + \sqrt{\varepsilon(1-\varepsilon)}\right).$$

This relation can also be inverted to yield

$$\varepsilon = \varepsilon_{R_0}(R_{\text{eff}}) = \frac{1}{2} - \frac{1}{2}\sqrt{1 - 4(2^{-R_{\text{eff}}} - 1/2)^2}$$

which gives the crossover probability ε for which an encoder of a given rate R_{eff} is used at the cutoff rate of the corresponding binary symmetric channel.

Thus, the expression "x dB above R_0" is used to mean that E_b/N_0 for the channel and the code used is x dB above $\gamma_b\left[\varepsilon_{R_0}(R_{\text{eff}}), R_{\text{eff}}\right]$.

Chapter 2

Universal Arithmetic Coding

In one of his most quoted papers [3], Elias introduced *universal* representations of the integers. In effect, his representations are codes whose expected codeword length is upper bounded when used to encode any finite or countably infinite source whose probability distribution is monotone non-increasing. Although Elias was not the first to study universal source coding (see Lynch [14] and Davisson [9]), his approach has influenced all subsequent research in the field. In this chapter, universal *arithmetic* coding for finite-alphabet sources will be considered. The optimal coding distribution for any source whose probability distribution lies in a convex set known as a polytope will be sought. This includes as a special case the set of all monotone distributions for a given finite alphabet size and the set of all monotone distributions with a given expected value and a given finite alphabet size. These two cases of practical interest will find their application in the source coding algorithms discussed in the remaining of this thesis.

Many results in this chapter are not new. In unpublished work [12], Gallager discovered the optimal coding distribution for universal coding over any convex set of probability distributions. As reported in Krichevsky's book [31], Ryabko solved the same problem independently and discovered the optimal coding distribution for the set of monotone distributions for a given alphabet size. Nevertheless, there are two reasons for including this discussion in this dissertation. One reason is that the results are relevant for the source coding algorithms presented in the next chapter. The second reason is that, except where stated, the results presented

here were obtained independently of Gallager's and Ryabko's work. As a result, the approach described differs considerably from both Gallager's and Ryabko's derivations.

2.1 On Universal Coding

The field of *universal coding* has one undeniably impressive feature: its name. It is inspiring, almost magical, and conjures up thoughts of space travel and science fiction. It attracts the attention of research students who, like this writer, would much rather be working in universal coding than in "source-independent coding for any member of a class of sources". As often happens when a sensational term enters the realm of science, it is difficult to formulate a precise definition for it. We will discuss some definitions and find none of them satisfying for our purpose. We will bypass this difficulty by using the more prosaic, but clearly defined concept of *optimality* to describe the type of codes that we will be seeking in this chapter.

Plea for a better definition of the term "universal"

We distinguish between the discipline known as "universal coding" and the adjective "universal" used to modify a specific code or encoder. We claim that there is a broad consensus on the meaning of the former, while there is disagreement on the precise meaning of the latter.

The discipline known as "universal coding" can be described as follows:

Definition 2.1 *An encoder is designed for* universal coding *if it is designed to encode the output of any source in a class of sources rather than the output of one particular source.*

One strategy for universal coding consists in estimating the parameters of the source based on the observation of the received sequence and adjusting the parameters of the encoder accordingly. Whenever the parameters of the encoder are modified in function of the particular sequence being encoded, we speak of an *adaptive* encoder. In this chapter, we are only interested in *non-adaptive* encoders and adaptive encoders are excluded from the following discussion.

According to Definition 2.1, when we specify that an encoder is designed for universal coding, we do not assert anything about its performance over the given class. Any encoder can be used for universal coding as the following example will show.

Example: The trivial code with one codeword of length 1 can be used for universal coding over the class of all discrete stationary sources. The corresponding encoder will generate the length 1 code sequence when used to encode the output of any discrete stationary source, independently of the particular parameters of the source being encoded. The performance of this encoder will be terrible for most discrete stationary sources, since the best decoder will be incapable of reproducing the source sequence in almost all cases.

The trivial encoder in our example is a lossy encoder (for nontrivial sources), meaning that the source sequence cannot always be recovered exactly from the code sequence. The definition of universal coding we gave applies to lossy and lossless coding. However, we will focus only on lossless coding for the remaining of this chapter.

When the adjective "universal" is used to modify a particular encoder, it certifies that the encoder is fit to be used for universal coding. As we have seen, any code or encoder could be used for universal coding for better or for worse. Therefore, the adjective "universal" must imply something about the performance for the given class of sources of the encoder it modifies.

In his book "Universal Compression and Retrieval" [31], Krichevsky gives a simple definition of a universal code:

Definition 2.2 (Krichevsky) *A code is universal if it is good for several sources.*

This definition relies heavily on the interpretation of the term "good". A mathematician would probably reject this definition as imprecise. A practically minded engineer might translate this definition as follows: "A code is universal if there is a sufficient number of customers willing to pay to encode the output of several sources with it as to remunerate the code's inventor". An academic discussion on the universality of a given code would then necessarily involve speculation on its potential commercial success. Krichevsky's definition is clever and it shows the aim of a universal encoder in simplest terms. Yet it cannot be used to determine whether a particular code is universal or not. In a way, it is akin to our definition of universal coding as a discipline.

In [3], Elias gives the following definition:

Definition 2.3 (Elias,1975) *A countably infinite prefix set of codewords has the* universal property *if, given any countable set M of messages and any probability distribution P defined on M, assigning messages in order*

of decreasing probability to codewords in order of increasing length gives an average codeword length that is bounded above by

$$K_1 + K_2 H(P),$$

where K_1 and K_2 are constants ≥ 1 and $H(P)$ is the entropy of the distribution P.

This definition is for a restricted type of encoders[1] and one particular class of sources. Given these restrictions, the definition fulfills all of our expectations: most countably infinite prefix sets of codewords will not satisfy the bound stated. Only those which do will earn the name "universal code". As expected, the adjective "universal" implies a bound on the performance of the encoder on the class of sources. However, Elias' definition cannot be generalized to a wider class of sources and codes. For example, if we consider a class consisting only of discrete memoryless sources with a given alphabet size N, any prefix-free set of N codewords would fulfill the bound in Elias' definition.

Krichevsky's definition of a universal code corresponds to our concept of universal coding but it does not provide a precise tool to distinguish between codes that are fit and codes that are unfit for universal coding. Elias' definition gives us such a tool only for a particular instance of universal coding. Unfortunately, we found no definition which fulfills our requirement in general. Therefore, we will refrain from using the appellation "universal code" and describe the codes in which we are interested by using the concept of optimality instead of universality. In short, we can say that

> An encoder is optimal for universal coding over a class of sources if there exists an upper bound on its performance over the class such that no other encoder achieves a better such upper bound.

There are several ways one can define an upper bound on the performance of an encoder over a class of sources and we will discuss some of those bounds for universal arithmetic coding shortly. Before we do so, we must argue against another definition of a universal encoder which is based on the concept of optimality.

Whatever the particular upper bound that we select to determine the optimality of an encoder over a class of sources, it is evident that if there

[1] Though Elias' definition of the term universal seems to apply to codes rather than encoders, his requirement that codewords of increasing length be assigned to messages in order of decreasing probabilities specifies the encoding to be applied.

existed an encoder which achieved the entropy for every source in the class, this encoder would necessarily be optimal. This is because of Shannon's converse to the noiseless coding theorem which states that it is not possible to encode the output of any source with an expected codeword length below the entropy of the source. It was discovered in the early days of universal coding that some families of encoders have a surprising property which we define as follows.

Definition 2.4 *Consider a parametric family of encoders* $\mathcal{C} = (\mathcal{C}_\eta)_{\eta\in\mathbb{N}}$ *and a class of sources* $\mathcal{S}$*. For every value of the integer parameter* η*,* C_η *is an encoder which can be used for universal coding over the class* $\mathcal{S}$*. The family of encoders* $\mathcal{C}$ *is said to be* asymptotically optimal *for* $\mathcal{S}$ *if, for every source in* $\mathcal{S}$*, the performance of the encoder* $\mathcal{C}_\eta$*, considered as a function of* η*, tends towards the entropy of this source as* η *tends to infinity.*

For example, $\mathcal{S}$ could be the class of all binary memoryless sources and $\mathcal{C}_\eta$ an encoder which encodes blocks of η source digits using a specific code. The coding method proposed by Lynch [14] and Davisson [9] is asymptotically optimal, as is one of the universal representations introduced by Elias in [3], though Elias' definition of asymptotic optimality differs from the definition given here.

When the first asymptotically optimal encoder families were discovered, it became widely accepted that an encoder must be a member of an asymptotically optimal family in order to be considered universal. In [10], Davisson includes a requirement that resembles asymptotic optimality in his definition of universal coding. We argue against including asymptotic optimality in the definition of a universal encoder for the following reasons:

- Why make the adjective "universal" virtually a synonym for "asymptotically optimal" when the latter expression already describes the corresponding property in a satisfying manner?

- Asymptotic optimality is a property of a family of encoders, while the adjective "universal" was meant to describe a property of a particular encoder.

The last point in particular emphasizes the disadvantage of such a definition. A definition of the adjective "universal" which requires asymptotic optimality judges a particular encoder based on the fact that it belongs to a "good" family rather than on the qualities of the encoder itself. Popular wisdom teaches us that there are black sheep in every good family. Do we really want such an aristocratic definition of a universal encoder?

Optimal coding distributions for universal arithmetic coding

In order to define optimal universal arithmetic coding, consider the situation when the semi-infinite output sequence $Y_1, Y_2, \ldots$ of a discrete stationary source is encoded by an arithmetic encoder. Let $y_1, y_2, \ldots, y_N$ denote the symbols of the source alphabet[2]. Since the source is stationary, the marginal probability distribution $P_Y(.)$ of a source output random variable Y_k is independent of k. If $P_Y(.)$ were known, then supposing that the arithmetic encoder uses infinite precision arithmetic, we remember from (1.4) that

$$\lim_{L\to\infty} E[W] = H(Y) = -\sum_{k=1}^{N} P_Y(y_k)\log_2 P_Y(y_k),$$

where $E[W]$ denotes the average expected codeword length per input symbol and L denotes the length of the input blocks encoded. The expected average codeword length when coding the symbol y_k is $-\log_2 P_Y(y_k)$. If the arithmetic encoder actually uses a coding distribution $Q(.)$ instead of the true source distribution $P_Y(.)$, then the expected average codeword length when coding the symbol y_k becomes $-\log_2 Q(y_k)$. The overall expected average codeword length then becomes

$$\begin{aligned}
\lim_{L\to\infty} E[W] &= -\sum_{k=1}^{N} P_Y(y_k)\log_2 Q(y_k) \\
&= -\sum_{k=1}^{N} P_Y(y_k)\log_2 \frac{Q(y_k)P_Y(y_k)}{P_Y(y_k)} \\
&= -\sum_{k=1}^{N} P_Y(y_k)\log_2 P_Y(y_k) + \sum_{k=1}^{N} P_Y(y_k)\log_2 \frac{P_Y(y_k)}{Q(y_k)} \\
&= H(Y) + D(P_Y||Q), \qquad (2.1)
\end{aligned}$$

where

$$D(P||Q) \stackrel{\text{def}}{=} \sum_{x\in \text{supp}P(.)} P(x)\log_2 \frac{P(x)}{Q(x)} \qquad (2.2)$$

denotes the *information divergence* from the probability distribution P to the probability distribution Q. It is also called the *Kullback-Leibler distance* from P to Q and is finite only when $\text{supp}Q(.) \subseteq \text{supp}P(.)$.

[2]Notice that the notation differs from the previous chapter. N now denotes the size of the source alphabet rather than the output blocklength of the encoder.

Universal arithmetic coding uses one coding distribution to encode the output of any source in a class of sources $\mathcal{S}$. We suppose that these sources are either discrete memoryless sources or discrete stationary sources and that their single-letter probability distribution is known to belong to a given class of distributions. Therefore, the class of sources $\mathcal{S}$ can be viewed as a class of source probability distributions $P_Y(.)$. Equation (2.1) applies, where $P_Y(.)$ is the probability distribution of the actual source being encoded and $Q(.)$ is the coding distribution used for universal coding. The expected average codeword length can be split into the entropy $H(Y)$ of the source being coded and a redundancy term $D(P_Y||Q)$. The optimal coding distribution $Q(.)$ for universal coding over the set $\mathcal{S}$ is the distribution which minimizes the redundancy $D(P_Y||Q)$ over $P_Y \in \mathcal{S}$. Two possible scenarios must be distinguished here:

1. the sources in $\mathcal{S}$ occur according to a known prior distribution. Let us define the indicator random variable X whose value is the actual source whose output is being encoded. X takes on values in $\mathcal{S}$ according to the prior distribution $P_X(.)$. The source distributions $P_Y(.)$ can be written as conditional distributions $P_{Y|X}(.|x)$. The optimal coding distribution $Q(.)$ is the one which minimizes the expected redundancy, i.e.,

$$Q = \arg\min_{Q'} \sum_{x \in \mathcal{S}} D(P_{Y|X}(.|x)||Q')P_X(x). \tag{2.3}$$

   The summation becomes an integral over the continuous probability distribution $p_X(.)$ if $\mathcal{S}$ is a continuous set of distributions. The prior distribution is sometimes called "side-information".

2. no prior distribution is known on $\mathcal{S}$. This is the case that will be investigated in this chapter. A uniform distribution can be assumed, as in the maximum-likelihood solution to a Bayesian estimation problem. Alternatively without the knowledge of a prior distribution, the optimal coding distribution can be chosen as the one which minimizes the worst-case redundancy over $\mathcal{S}$, i.e.,

$$Q = \arg\min_{Q'} \max_{P_Y \in \mathcal{S}} D(P_Y||Q'). \tag{2.4}$$

   According to the nature of $\mathcal{S}$, the maximum or the minimum in (2.4) may not exist, in which case they must be replaced by a supremum or an infimum respectively.

The remaining of this chapter will be devoted to the search for a solution $Q(.)$ to (2.4) when $\mathcal{S}$ is a convex set known as a polytope. The next

section will give the definition of a polytope, list some of its properties, and introduce the polytopes which are of practical interest for the source coding algorithms developed later.

2.2 Polytopes and Discrete Probability Distributions

In this section, we consider discrete probability distributions of size N as plain vectors in $\mathbb{R}^N$. We will write $\mathcal{P}_N$ for the set of vectors in $\mathbb{R}^N$ which correspond to probability distributions of alphabet size N, i.e., $\mathcal{P}_N$ is the set of those vectors in $\mathbb{R}^N$ whose components are non-negative and sum to 1. $\mathcal{P}_N^*$ designates the open set of probability distributions in $\mathcal{P}_N$ whose components are all different from zero. In the analysis of these distributions, we will also use vectors in $\mathbb{R}^N$ which are not in $\mathcal{P}_N$, and vectors in $\mathcal{P}_r$, where $r \neq N$. To avoid confusion, we introduce the following notation:

- capital letters, e.g. P, denote vectors of $\mathbb{R}^N$ which are in $\mathcal{P}_N$. $p_1, p_2, \ldots, p_N$ are the elements of the vector P.
- underlined lower-case letters, e.g. $\underline{v}$, denote vectors which are not necessarily in $\mathcal{P}_N$. $v_1, v_2, \ldots, v_r$ are the elements of $\underline{v}$.
- bold capital letters, e.g. $\mathbf{A}$, denote matrices of any dimension.

It may seem strange that we have more than one way of denoting vectors, but this will help us keep in mind which vectors have a "physical" meaning as potential source probability distributions. Furthermore, we will use the notation $\underline{1}_N$ for the "all ones" vector and $\underline{0}_N$ for the "all zeros" vector in $\mathbb{R}^N$. We will usually drop the subscript N when there is no ambiguity. Finally, when we write an inequality involving vectors, we mean that the inequality holds for every component of the vectors.

We need the concept of a convex combination to define a polytope.

Definition 2.5 *Let $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$ be any r vectors in $\mathbb{R}^N$, let $\underline{\mu}$ be any vector in $\mathcal{P}_r$. Then the vector*

$$\underline{v} = \mu_1 \underline{v}_1 + \mu_2 \underline{v}_2 + \ldots + \mu_r \underline{v}_r$$

is called a convex combination *of the vectors $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$. If $\underline{\mu} \in \mathcal{P}_r^*$, then $\underline{v}$ is called a* strict *convex combination of $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$.*

We are now ready to define a polytope.

Definition 2.6 *Let* $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$ *be any* r *vectors in* $\mathbb{R}^N$ *such that no vector* $\underline{v}_i$ *is a convex combination of the other vectors. The* polytope $S \stackrel{\text{def}}{=} < \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r >$ *is the set of all convex combinations of* $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$. *The vectors* $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$ *are called the* vertices *of the polytope* S. *If* S *is a polytope, then the* open polytope $\breve{S} \stackrel{\text{def}}{=} > \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r <$ *is the set of all* strict *convex combinations of the vertices of* S.

For example a line segment, a triangle or a square in $\mathbb{R}^2$ are common polytopes with 2, 3, or 4 vertices respectively. A polytope is a convex set by the definition of convex sets, which demands that all convex combinations of points in the set be in the set. The following proposition will make it easier in some cases to verify whether r vectors are valid vertices of a polytope.

Proposition 2.1 *Any* r *linearly independent vectors of* $\mathbb{R}^N$ *are the vertices of a polytope.*

Proof: this follows immediately from the fact that a convex combination is a special case of a linear combination. □

The converse is not true in general, i.e., the vertices of a polytope need not be linearly independent vectors. We define the *rank* of a polytope S as the rank of the matrix whose columns are the vertices of S. We will call a polytope with linearly independent vertices a *regular* polytope to reflect the fact that its corresponding matrix has full column rank.

The polytopes that we are considering always have all their vertices in $\mathcal{P}_N$. In this case, we can state the following proposition:

Proposition 2.2 *Any polytope* S *whose vertices* $P_1, P_2, \ldots, P_r$ *are in* $\mathcal{P}_N$ *is a subset of* $\mathcal{P}_N$.

Proof: this can be verified by writing any element P of S as

$$P = \mu_1 P_1 + \mu_2 P_2 + \ldots + \mu_r P_r.$$

Every component of P is non-negative because it is a sum of non-negative terms. The sum of the components of P is

$$\begin{aligned} \sum_{k=1}^{N} p_k &= \sum_{k=1}^{N} \sum_{i=1}^{r} \mu_i p_{ik} \\ &= \sum_{i=1}^{r} \mu_i \sum_{k=1}^{N} p_{ik} = 1. \end{aligned}$$

Thus, S is a subset of $\mathcal{P}_N$. □

We are ready to discuss the few polytopes that will concern us for the rest of this chapter. $\mathcal{P}_N$ itself is of course a regular polytope whose vertices are the unit vectors of $\mathbb{R}^N$, i.e.,

$$\mathcal{P}_N =< U_1, U_2, \ldots, U_N >=< \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} > .$$

Its vertices are linearly independent probability vectors and any probability vector can be written as a convex combination of its vertices, where the components of the vector are the convex coefficients.

The set $\mathcal{M}_N$ of all monotone probability distributions of alphabet size N, i.e.,

$$\mathcal{M}_N = \{P | P \in \mathcal{P}_N \text{ and } p_1 \geq p_2 \geq \ldots \geq p_N\}$$

is the regular polytope

$$\mathcal{M}_N =< \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1/N \\ 1/N \\ 1/N \\ \vdots \\ 1/N \end{bmatrix} >,$$

and the corresponding open polytope $\breve{\mathcal{M}}_N$ is the set of all strictly monotone distributions. The vertices of $\mathcal{M}_N$ are linearly independent probability vectors. Every monotone probability vector P can be written as a convex combination of the vertices of $\mathcal{M}_N$ using the convex coefficients $\mu_1, \mu_2, \ldots, \mu_N$ with $\mu_N = Np_N$ and

$$\mu_k = k(p_k - p_{k+1}) \text{ for } k = 1, 2, \ldots, N-1,$$

which coefficients are all non-negative and sum to 1.

We come at last to the polytopes that will be of most interest to us. We look at the set of probability vectors whose expected value is a constant c. The expected value of a function $f(.)$ of a random variable X with probability distribution P is defined as

$$E[f(X)] = \sum_{k=1}^{N} p_k f(x_k).$$

By writing $f_k \stackrel{\text{def}}{=} f(x_k)$, we can identify the function f with the vector $\underline{f} = (f_1, f_2, \ldots, f_N)$. If we consider the set of all vectors in $\mathbb{R}^N$ whose "expected value" is equal to c, we get the set $\{\underline{v} \in \mathbb{R}^N \,|\, \underline{f}^T \cdot \underline{v} = c\}$, which is the equation for a hyperplane in $\mathbb{R}^N$. So the set of all probability vectors with expected value c is the intersection of the polytope $\mathcal{P}_N$ and a hyperplane. We call this set $\mathcal{P}_{N,\underline{f},c}$. Lemma 2.2 describes the nature of this set. In the proof of this lemma, we apply a well known lemma from convexity theory, due to the Hungarian mathematician Farkas (pronounced "Farkash"), which we state here.

Lemma 2.1 (Farkas' Lemma, 1901) *Let* $\mathbf{A}$ *be an* $M \times N$ *real matrix and* $\underline{b}$ *a vector in* $\mathbb{R}^N$. *Then the following two statements are equivalent:*

1. $\forall \underline{x} \in \mathbb{R}^N, \mathbf{A}^T\underline{x} \leq \underline{0}_N \Longrightarrow \underline{b}^T\underline{x} \leq 0$
2. $\exists \underline{\mu} \in \mathbb{R}^M$ *such that* $\mathbf{A}\underline{\mu} = \underline{b}$ *and* $\underline{\mu} \geq \underline{0}_M$

Farkas' Lemma allows us to exchange a set of equations for a set of inequalities and vice-versa. We refer to Rockafellar's book [35] for a proof of this lemma. We will prove a slightly modified version of Farkas' Lemma in the next section. We are now ready to state our lemma about the nature of $\mathcal{P}_{N,\underline{f},c}$

Lemma 2.2 *The set* $\mathcal{P}_{N,\underline{f},c}$ *of all probability vectors* P *whose expected value* $\underline{f}^T \cdot P$ *is equal to* c

(i) *is empty if* $c > \max_k f_k$ *or* $c < \min_k f_k$,

(ii) *is a polytope of rank* $N-1$ *if* $\min_k f_k < c < \max_k f_k$,

(iii) *or is a polytope of rank* r *if* $c = \max_k f_k$ *or* $c = \min_k f_k$, *where* r *is the number of indices* k *for which* $f_k = c$.

The proof of this lemma will be long, but it includes a construction for the set $\mathcal{P}_{N,\underline{f},c}$ which will be useful in practice.

Proof: for the proof, we can assume that $f_1 \leq f_2 \leq \ldots \leq f_N$. For any P in $\mathcal{P}_N$,

$$f_1 \leq \sum_{k=1}^{N} p_k f_k \leq f_N$$

so the intersection $\mathcal{P}_{N,\underline{f},c}$ we seek is non-empty if and only if

$$f_1 \leq c \leq f_N. \tag{2.5}$$

This proves **(i)**. If equality holds on both sides in (2.5), then all the vectors $P \in \mathcal{P}_N$ satisfy $\underline{f}^T \cdot P = c$ and $\mathcal{P}_{N,\underline{f},c} = \mathcal{P}_N$. If equality holds on either side of (2.5) then any probability vector which has a non-zero k-th component for which $f_k \neq c$ has $\underline{f}^T \cdot P \neq c$ and so cannot be in $\mathcal{P}_{N,\underline{f},c}$. We thus have a problem in $\mathcal{P}_r$ where r is the number of components of $\underline{f}$ which are equal to c, for which we can repeat the argument used when equality holds on both sides. This proves **(iii)**.

Consider now the case where both inequalities are strict. Let i_0 be the greatest index of $\underline{f}$ for which $f_{i_0} < c$ and j_0 be the smallest index of $\underline{f}$ for which $f_{j_0} > c$. Let $k_1, k_2, \ldots k_m$ be the indices of $\underline{f}$ for which $f_{k_l} = c$, where m can of course be zero. For every $i \leq i_0$ and every $j \geq j_0$, the vector $\underline{v}^{(i,j)}$ with components

$$\begin{aligned} v_i^{(i,j)} &= \frac{f_j - c}{f_j - f_i} \\ v_j^{(i,j)} &= \frac{c - f_i}{f_j - f_i} \\ v_k^{(i,j)} &= 0 \text{ for } k = 1, 2, \ldots N, k \neq i, k \neq j \end{aligned} \tag{2.6}$$

is in $\mathcal{P}_{N,\underline{f},c}$ because its components are non-negative, their sum is 1 and $\underline{f}^T \cdot \underline{v}^{(i,j)} = c$. For $l = 1 \ldots m$, the vector $\underline{v}^{(k_l)}$ with components

$$\begin{aligned} v_{k_l}^{(k_l)} &= 1 \\ v_k^{(k_l)} &= 0 \text{ for } k = 1, 2, \ldots N, k \neq k_l \end{aligned} \tag{2.7}$$

is also in $\mathcal{P}_{N,\underline{f},c}$ because it is in $\mathcal{P}_N$ and $\underline{f}^T \cdot \underline{v}^{(k_l)} = c$. All in all, there are $i_0(N - j_0 + 1) + m$ vectors $\underline{v}^{(i,j)}$ and $\underline{v}^{(k_l)}$, where $i_0 + m + 1 = j_0$. No $\underline{v}^{(i,j)}$ can be a convex combination of other vectors because the only other vectors which have a positive i-th component have a j-th component which is equal to zero. No $\underline{v}^{(k_l)}$ can be a convex combination of other vectors $\underline{v}$ because no other vector has a non-zero k_l-th component. Thus, the vectors defined in (2.6) and (2.7) are the vertices of a polytope S. Every element $\underline{v}$ of S is a probability vector by Proposition 2.2 and its expected value $\underline{f}^T \cdot \underline{v}$ is equal to c because $\underline{v}$ is a convex combination of the vertices of S which all have the same expected value c. Thus, we can state that $S \subseteq \mathcal{P}_{N,\underline{f},c}$.

Let any N vertices of S define the rows of a square matrix $\mathbf{A}$. Then the equation $\mathbf{A}\underline{x} = \underline{1}_N$ has at least two distinct solutions, namely $\underline{x} = \underline{1}_N$ and $\underline{x} = (1/c)\underline{f}$. The determinant of the matrix must therefore be zero, and the rank of the polytope S can be at most $N - 1$. The $N - 1$ vectors

$$\underline{v}^{(1,j_0)}, \underline{v}^{(2,j_0)}, \ldots, \underline{v}^{(i_0,j_0)}, \underline{v}^{(k_1)}, \ldots, \underline{v}^{(k_m)}, \underline{v}^{(i_0,j_0+1)}, \ldots, \underline{v}^{(i_0,N)}$$

$$
\begin{array}{c}
1 \\
\vdots \\
\vdots \\
i_0 \\
k_1 \\
\vdots \\
k_m \\
j_0 \\
\vdots \\
\vdots \\
N
\end{array}
\left[\begin{array}{ccc|c|ccc|ccc}
X & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & X & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\hline
0 & \cdots & 0 & X & 0 & \cdots & 0 & X & \cdots & X \\
\hline
0 & \cdots & 0 & 0 & X & \cdots & 0 & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & X & 0 & \cdots & 0 \\
\hline
X & \cdots & X & X & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\hline
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & X & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & X
\end{array}\right]
$$

Figure 2.1: the $N-1$ columns of this matrix are linearly independent.

are linearly independent as can be seen by looking at the positions of their non-zero elements, which are indicated as X's in the matrix in Figure 2.1. Thus, the polytope S has rank $N-1$.

To prove the converse, namely that $\mathcal{P}_{N,\underline{f},c} \subseteq S$, we will use Farkas' Lemma 2.1. Let $\mathbf{V}$ denote the matrix whose columns are the vertices of S. Take any vector $\underline{x} \in \mathbb{R}^N$ such that

$$\mathbf{V}^T \underline{x} \leq \underline{0}_N. \tag{2.8}$$

For $j = j_0, \ldots, N$, consider the inequality $\underline{v}^{(i_0,j)T} \cdot \underline{x} \leq 0$. We can write this inequality as

$$\frac{f_j - c}{f_j - f_{i_0}} x_{i_0} + \frac{c - f_{i_0}}{f_j - f_{i_0}} x_j \leq 0.$$

This implies that

$$x_j \leq \frac{c - f_j}{c - f_{i_0}} x_{i_0}. \tag{2.9}$$

Now consider the inequality $\underline{v}^{(i,j_0)T} \cdot \underline{x} \leq 0$ for $i = 1, \ldots, i_0$. This inequality implies

$$x_i \leq \frac{c - f_i}{c - f_{j_0}} x_{j_0} \leq \frac{c - f_i}{c - f_{j_0}} \frac{c - f_{j_0}}{c - f_{i_0}} x_{i_0} = \frac{c - f_i}{c - f_{i_0}} x_{i_0}, \tag{2.10}$$

where the second inequality comes from using (2.9). Finally, the inequality $\underline{v}^{(k_l)T} \cdot \underline{x} \leq 0$ for $l = 1, \ldots, m$ implies that $x_{k_l} \leq 0$, which we can also put

in a form similar to (2.9) and (2.10), viz.

$$x_{k_l} \leq \frac{c - f_{k_l}}{c - f_{i_0}} x_{i_0} \tag{2.11}$$

because $c - f_{k_l} = 0$ for $l = 1, \ldots, m$. Now take any vector $\underline{v} \in \mathcal{P}_{N,\underline{f},c}$ and consider its scalar product with $\underline{x}$. Using (2.9), (2.10) and (2.11), we can write

$$\begin{aligned} \underline{v}^T \cdot \underline{x} &= \sum_{k=1}^{N} v_k x_k \\ &\leq \sum_{k=1}^{N} v_k \frac{c - f_k}{c - f_{i_0}} x_{i_0} = \\ &= \frac{x_{i_0}}{c - f_{i_0}} \left(c \sum_{k=1}^{N} v_k - \sum_{k=1}^{N} f_k v_k \right) \\ &= 0, \end{aligned} \tag{2.12}$$

where the last equality comes from the fact that $\underline{v}$ is a probability distribution so that the sum of its elements is 1 and its expected value $\underline{f}^T \cdot \underline{v}$ is equal to c. By Farkas' Lemma 2.1, since $\mathbf{V}^T \underline{x} \leq \underline{0}$ implies that $\underline{v}^T \cdot \underline{x} \leq 0$ for all $\underline{x}$, there must be a vector $\underline{\mu} \in \mathbb{R}^N$ such that $\underline{\mu} \geq \underline{0}_N$ and $\mathbf{V}\underline{\mu} = \underline{v}$. Since the expected value of every column of $\mathbf{V}$ is equal to c and the expected value of $\underline{v}$ is also equal to c, we know that the sum of the components of $\underline{\mu}$ must be 1. We have proved that every element of $\mathcal{P}_{N,\underline{f},c}$ can be written as a convex combination of the columns of $\mathbf{V}$, which implies that $\underline{v} \in S$, and hence that $\mathcal{P}_{N,\underline{f},c} \subseteq S$. This concludes the proof. □

As promised, the proof gave us a method for constructing the polytope $\mathcal{P}_{N,\underline{f},c}$. Perhaps surprisingly, it turned out that such a polytope may have up to $N^2/4$ vertices. Intuition is misled by the picture we have of cutting the triangle $\mathcal{P}_3$ in $\mathbb{R}^3$ with a plane, which will yield a line segment in the best case, i.e., a polytope with only two vertices.

Example: The polytope $\mathcal{P}_{N,\underline{f},c}$ for $N = 5$, $\underline{f} = [0, 1, 2, 3, 4]^T$ and $c = 3/2$ has six vertices. We can compute them using (2.6) to obtain

$$< \begin{bmatrix} 1/4 \\ 0 \\ 3/4 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \end{bmatrix}, \begin{bmatrix} 5/8 \\ 0 \\ 0 \\ 0 \\ 3/8 \end{bmatrix}, \begin{bmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3/4 \\ 0 \\ 1/4 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 5/6 \\ 0 \\ 0 \\ 1/6 \end{bmatrix} > .$$

Now consider the intersection of the polytope $S =< \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_s >$ with the hyperplane $H_{\underline{h},c} = \{\underline{x} \in \mathbb{R}^N | \underline{h}^T \cdot \underline{x} = c\}$. Let $\mathbf{V}$ denote the matrix whose columns are the vertices of S. Every vector $\underline{v}$ in the polytope can be expressed as

$$\underline{v} = \mathbf{V}\underline{\mu},$$

where $\underline{\mu} \in \mathcal{P}_s$. Every element $\underline{v}$ in the intersection of the polytope with the hyperplane $H_{\underline{h},c}$ must satisfy

$$\underline{h}^T \cdot \underline{v} = c = \underline{h}^T \mathbf{V}\underline{\mu}.$$

Let $\underline{f} = \mathbf{V}^T\underline{h}$. We see that the convex coefficients $\underline{\mu}$ of the vertices of S which generate points in the intersection of S with $H_{\underline{h},c}$ must satisfy $\underline{f}^T\underline{\mu} = c$. The set of all convex combinations which satisfy this condition is the set defined above as $\mathcal{P}_{s,\underline{f},c}$. By Lemma 2.2, we know the nature of this set. If it is non-empty, it is a polytope. Let us write $\mathbf{A}$ for the matrix whose r columns are its vertices. Then, every element $\underline{\mu}$ of $\mathcal{P}_{s,\underline{f},c}$ can be written as $\underline{\mu} = \mathbf{A}\underline{\nu}$, where $\underline{\nu} \in \mathcal{P}_r$. We see that the intersection of S with $H_{\underline{h},c}$ is the set of all vectors $\underline{v} \in \mathbb{R}^N$ which satisfy

$$\underline{v} = \mathbf{V}\mathbf{A}\underline{\nu} \text{ where } \underline{\nu} \in \mathcal{P}_r.$$

This is again a polytope. We have proved the following theorem:

Theorem 2.1 *Let S be any polytope in $\mathbb{R}^N$ and let $\mathbf{V}$ be the matrix whose columns are the s vertices of S. Let $H_{\underline{h},c} = \{\underline{v} \in \mathbb{R}^N | \underline{h}^T \cdot \underline{v} = c\}$ be a hyperplane in $\mathbb{R}^N$. Define the vector $\underline{f} = \mathbf{V}^T\underline{h}$. Then the intersection of S with $H_{\underline{h},c}$*

(i) *is empty if $c > \max_k f_k$ or $c < \min_k f_k$,*

(ii) *is a polytope of rank $s-1$ if $\min_k f_k < c < \max_k f_k$ and S is regular,*

(iii) *is a polytope of rank r if $\min_k f_k < c < \max_k f_k$, where $r \leq rank(S)$,*

(iv) *or is a polytope of rank r if one of the inequalities in (i)-(iii) holds with equality, where $r \leq \min\{rank(S), m\}$ and m is the number of indices k for which $f_k = c$.*

Proof: the proof follows immediately from the results above and from the fact that the rank of the matrix product $\mathbf{VA}$ is at most equal to $\min\{\text{rank}(\mathbf{V}), \text{rank}(\mathbf{A})\}$ and thus it is equal to the rank of $\mathbf{A}$ if $\mathbf{V}$ is non-singular. □

In particular, the set of all monotone N-ary distributions with a given expected value, which we shall denote as $\mathcal{M}_{N,\underline{f},c}$, is a polytope of rank $N-1$. We can construct this set based on the methods indicated in the proof of Lemma 2.2 and of Theorem 2.1.

Example: Suppose we wish to construct the set $\mathcal{M}_{N,\underline{h},c}$ for $N = 5$, $\underline{h} = [0,1,2,3,4]^T$ and $c = 3/2$. $\mathcal{M}_{N,\underline{h},c}$ is the intersection of the polytope $\mathcal{M}_5$ with the hyperplane $H_{\underline{h},c}$. As indicated in the proof of Theorem 2.1, we must compute the vector $\underline{f} = \mathbf{V}^T\underline{h}$ where $\mathbf{V}$ is the matrix whose columns are the vertices of the polytope $\mathcal{M}_5$. This yields

$$\underline{f} = [0, 1/2, 1, 3/2, 2]^T.$$

We now compute the matrix $\mathbf{A}$ whose columns are the vertices of the polytope $\mathcal{P}_{N,\underline{f},c}$ using the construction method given in the proof of Lemma 2.2. This yields

$$\mathbf{A} = \begin{bmatrix} 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 3/4 & 2/3 & 1/2 \end{bmatrix}.$$

Finally, the matrix product $\mathbf{VA}$ yields a matrix whose columns are the vertices of the polytope $\mathcal{M}_{N,\underline{h},c}$ we were seeking, i.e.,

$$< \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 0 \end{bmatrix}, \begin{bmatrix} 2/5 \\ 3/20 \\ 3/20 \\ 3/20 \\ 3/20 \end{bmatrix}, \begin{bmatrix} 3/10 \\ 3/10 \\ 2/15 \\ 2/15 \\ 2/15 \end{bmatrix}, \begin{bmatrix} 4/15 \\ 4/15 \\ 4/15 \\ 1/10 \\ 1/10 \end{bmatrix} > .$$

In the next sections, we seek the optimal probability distribution for universal coding over polytopes of probability distributions. In practice, we will usually be interested in the optimal coding distributions for the polytopes $\mathcal{M}_N$ and $\mathcal{M}_{N,\underline{f},c}$.

Before proceeding further, it is worthwhile to point out that Theorem 2.1 can be extended to cover the intersection of a polytope with any number of hyperplanes. This is stated in the following corollary:

Proposition 2.3 (Corollary to Theorem 2.1) *The intersection of a polytope S with any number of hyperplanes is either empty or a polytope.*

Proof: this follows immediately by applying Theorem 2.1 recursively. □

2.3 Optimal Universal Coding for a Polytope

We are now ready to discuss the solution of (2.4) when the set considered is a polytope $S \subseteq \mathcal{P}_N$. The probability distribution $Q \in \mathcal{P}_N$ is sought such that the worst-case redundancy ρ is minimized over all distributions in $\mathcal{P}_N$, i.e.,

$$\rho = \min_{Q \in \mathcal{P}_N} \max_{P \in S} D(P||Q). \tag{2.13}$$

In particular, the solutions will be investigated closely for the following polytopes:

- $\mathcal{P}_N$, i.e., the optimal coding distribution is sought for universal arithmetic coding when nothing is known about the source except that it is stationary with a single-letter distribution of alphabet size N.
- $\mathcal{M}_N$, i.e., the source is known to be stationary with a single-letter distribution $P_Y(y_k), k = 1, \ldots, N$ that is monotone non-increasing in k.
- $\mathcal{P}_{N,\underline{f},c}$, i.e., the source is known to be stationary and its expected value satisfies $E[f(Y)] = c$. Particular attention will be paid to the case when $f(y_k) = k$, for $k = 0, 1, 2, \ldots, N-1$.
- $\mathcal{M}_{N,\underline{f},c}$, i.e., the source is known to be stationary with a monotone non-increasing single-letter distribution $P_Y(y_k)$ and $E[f(Y)] = c$.

Some insight is gained by noting the following fundamental property of the information divergence.

Proposition 2.4 *the information divergence $D(P||Q)$ is strictly convex-$\cup$ in both its variables P and Q.*

Proof: the expression for the divergence can be written as

$$D(P||Q) = -H(P) - \sum_{x \in \mathrm{supp}P} P(x) \log_2 Q(x), \tag{2.14}$$

where the notation $H(P)$ is used somewhat imprecisely to denote the entropy of a random variable whose probability distribution is $P(.)$. Both $H(.)$ and $\log(.)$ are known to be strictly convex-$\cap$ (concave) functions in their respective variable. With respect to its first variable P, (2.14) shows the divergence to be a sum of a strictly convex-$\cup$ function and a linear function. With respect to its second variable Q, the divergence is the sum

of a constant, $-H(P)$, and a sum of strictly convex-$\cup$ functions. Both these combinations are strictly convex-$\cup$. □

The following property holds for the maximization of any convex-$\cup$ function over a polytope. In particular, it holds for the inner maximization of the divergence in (2.13).

Proposition 2.5 *let f be any convex-$\cup$ function whose domain includes a polytope $S =< \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r >$. Then*

$$\max_{\underline{x} \in S} f(\underline{x}) = \max_k f(\underline{v}_k). \tag{2.15}$$

In other words, the function attains its maximum over the polytope S on a vertex of S.

Proof: it is known from the theory of convex functions ([35], Chapter 32) that the supremum of a convex-$\cup$ function f over a closed convex set $\mathcal{C}$ is equal to the supremum of f on the relative boundary[3] of $\mathcal{C}$. In the case of a polytope, the relative boundary is a union of polytopes. Now the maximum of f is sought over this union of polytopes. It must lie on the boundary of one of the component polytopes. The boundaries of the component polytopes are also unions of polytopes. This step can be repeated until at last the function is to be maximized only over a union of line segments connecting vertices of the original polytope S. These line segments are polytopes themselves, and their boundaries are the vertices $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_r$ of S. □

The problem stated in (2.13) can be rewritten using Propositions 2.4 and 2.5. The optimal coding distribution for universal arithmetic coding over a polytope of probability distributions $S =< P_1, P_2, \ldots, P_r >$ is the distribution $Q \in \mathcal{P}_N$ which yields the minimal redundancy ρ in the expression

$$\rho = \min_{Q \in \mathcal{P}_N} \max_{k=1 \ldots r} D(P_k || Q). \tag{2.16}$$

Proposition 2.5 also shows the rationale for our interest in polytopes. In some applications, the optimal coding distribution is sought for any set of r probability distributions in $\mathcal{P}_N$. If these distributions do not fulfill the requirements we stated in Definition 2.6 for the vertices of a polytope, we can start by eliminating those distributions which can be written as a convex combination of the others. The remaining distributions will be

[3]The *relative boundary* of a convex set in $\mathbb{R}^N$ is defined in [35] as the boundary which results when the set is regarded as a subset of its affine hull.

the vertices of a polytope which contains the probability distributions that were eliminated. Therefore, the optimal coding distribution for the original r distributions is equal to the optimal coding distribution for the vertices of the remaining polytope. In other words, the problem of finding the optimal coding distribution for a finite set of probability distributions is equivalent to the problem of finding the optimal coding distribution for a polytope.

There is a considerable progress from (2.13) to (2.16). What was a combined minimization and maximization of a function of two continuous variables is now seen as the minimization of a function of one variable whose value at every point is the maximum of r convex functions. The following proposition (see [35, Theorem 5.5]) will show the nature of a function constructed in this way.

Proposition 2.6 *let $f_1, f_2, \ldots, f_r$ be convex-$\cup$ functions defined over the convex region $\mathcal{C}$. Let f be a real-valued function defined over $\mathcal{C}$ such that, for every $\underline{v} \in \mathcal{C}$,*

$$f(\underline{v}) = \max_{k=1\ldots r} f_k(\underline{v}),$$

i.e., the function f is equal in every point to the maximum of the component functions $f_1, \ldots, f_r$. Then f is a convex-$\cup$ function. Furthermore, if $f_1, f_2, \ldots, f_r$ are strictly convex-$\cup$, then f is strictly convex-$\cup$.

Proof: Let $\underline{v}$ and $\underline{w}$ be any two points in $\mathcal{C}$. Now define i and j such that

$$f(\underline{v}) = \max_k f_k(\underline{v}) = f_i(\underline{v}) \tag{2.17}$$

$$f(\underline{w}) = \max_k f_k(\underline{w}) = f_j(\underline{w}). \tag{2.18}$$

For any $\lambda \in [0,1]$, define n such that

$$f(\lambda\underline{v} + (1-\lambda)\underline{w}) = \max_k f_k(\lambda\underline{v} + (1-\lambda)\underline{w}) = f_n(\lambda\underline{v} + (1-\lambda)\underline{w}).$$

Since $f_n(.)$ is convex-$\cup$, the following inequality holds

$$f_n(\lambda\underline{v} + (1-\lambda)\underline{w}) \leq \lambda f_n(\underline{v}) + (1-\lambda) f_n(\underline{w}).$$

But (2.17) and (2.18) imply that $f_i(\underline{v}) \geq f_n(\underline{v})$ and $f_j(\underline{w}) \geq f_n(\underline{w})$, which in turn implies that

$$f(\lambda\underline{v} + (1-\lambda)\underline{w}) \leq \lambda f_i(\underline{v}) + (1-\lambda) f_j(\underline{w}) = \lambda f(\underline{v}) + (1-\lambda) f(\underline{w}).$$

Therefore, $f(.)$ is convex-$\cup$. Strict convexity is proved by replacing all inequalities in the proof by strict inequalities. □

Proposition 2.6 shows our problem as stated in (2.16) to be a simple minimization of a convex-$\cup$ function over a convex-$\cup$ region, albeit a convex-$\cup$ function defined as the maximum of r component convex functions. The theory of convex-$\cup$ functions tells us that a convex-$\cup$ function will always have a minimum over a convex region and that this minimum will be unique. Therefore, all that remains to do is to find this minimum for all polytopes.

We will define a property which will help us to discover the minimum we seek for some polytopes.

Definition 2.7 *The probability distribution $Q \in \mathcal{P}_N$ is said to be* Kullback-Leibler equidistant *from the probability distributions $P_1, P_2, \ldots, P_r$ in $\mathcal{P}_N$ if*

$$D(P_1||Q) = D(P_2||Q) = \ldots = D(P_r||Q) = C$$

for some $C \in \mathbb{R}^+$.

Based on this definition, we will prove a theorem that will allow us to determine the optimal coding distribution for some polytopes. In the proof of this theorem, we will use the following lemma which is closely related to Farkas' Lemma 2.1.

Lemma 2.3 *Let $\mathbf{A}$ be an $N \times M$ matrix and $\underline{b}$ be an $N \times 1$ vector. Then one and only one of the following two propositions is true.*

$$\exists \underline{x} \in \mathbb{R}^N \quad \text{such that} \quad \mathbf{A}^T \underline{x} \geq^* \underline{0}_M \text{ and } \underline{b}^T \cdot \underline{x} = 0 \tag{2.19}$$

$$\exists \underline{\mu} \in \mathbb{R}^M \quad \text{such that} \quad \underline{\mu} > \underline{0}_M \text{ and } \begin{cases} \mathbf{A}\underline{\mu} = \underline{b}, \text{ or} \\ \mathbf{A}\underline{\mu} = -\underline{b}, \text{ or} \\ \mathbf{A}\underline{\mu} = \underline{0}_N. \end{cases} \tag{2.20}$$

where the notation $\geq^*$ signifies that at least one of the component inequalities is strict.

The proof of Lemma 2.3 is given in an appendix to this chapter. We are now ready to state the following theorem which provides the solution of (2.13) for a particular type of polytope.

Theorem 2.2 *Let $S =< P_1, P_2, \ldots, P_r >$ be a polytope in $\mathcal{P}_N$. If a distribution Q in the open polytope $\breve{S}$ is Kullback-Leibler equidistant from the vertices of S, then Q is the unique distribution in $\breve{S}$ with this property and Q is the solution of the problem stated in (2.13) for the polytope S.*

Proof: Consider the problem stated in (2.13). Using the convexity of the divergence and Propositions 2.5 and 2.6, we showed that this problem is equivalent to minimizing a convex function defined as

$$f(Q) \stackrel{\text{def}}{=} \max_{k=1\ldots N} D(P_k||Q).$$

Now suppose the distribution Q in the open polytope $\breve{S}$ is Kullback-Leibler equidistant from the vertices of S but it is not the minimum of $f(.)$. Then, since a convex function can have no local minima, there must be at least one direction in $\mathcal{P}_N$ relative to which all component functions $D(P_k||.)$ of $f(.)$ are decreasing at the position Q. In other words,

$$\exists \underline{d} \in \mathbb{R}^N \text{ such that } \begin{cases} \underline{d}^T \cdot \underline{1}_N = 0 \text{ and} \\ \underline{d}^T \cdot \text{grad}D(P_k||Q) \leq 0 \text{ for } k = 1\ldots N, \end{cases} \tag{2.21}$$

where at least one of the inequalities is strict. The gradient of the function $D(P_k||Q)$ whose N variables are the components of the probability vector Q are easily computed as

$$\text{grad}D(P_k||Q) = \frac{1}{\ln 2}\left[-\frac{p_{k1}}{q_1}, -\frac{p_{k2}}{q_2}, \ldots, -\frac{p_{kN}}{q_N}\right]^T.$$

Using the notation

$$\hat{\underline{d}} \stackrel{\text{def}}{=} \left[\frac{d_1}{q_1}, \frac{d_2}{q_2}, \ldots, \frac{d_N}{q_N}\right]^T,$$

and letting $\mathbf{P}$ be the matrix whose k-th row is the probability vector P_k^T, we can rewrite the system of inequalities and one equality in (2.21) in matrix form as

$$\mathbf{P}\hat{\underline{d}} \geq^* \underline{0}_N \tag{2.22}$$

$$\hat{\underline{d}}^T \cdot Q = 0. \tag{2.23}$$

This corresponds to the statement of Lemma 2.3. Since Q is in the open polytope $\breve{S}$, it can be written as a strict convex combination of the vertices of S. In other words, there exists a vector $\underline{\mu} > 0$ such that

$$\mathbf{P}^T\underline{\mu} = Q.$$

Therefore, the second proposition (2.20) of Lemma 2.3 is satisfied. This implies that the first proposition of Lemma 2.3 cannot be satisfied. In

other words, any vector $\underline{d}$ which satisfies the inequalities (2.22) cannot satisfy the equality (2.23). Thus, the function $f(.) = \max_k D(P_k||.)$ has a local and a global minimum at the Kullback-Leibler equidistant distribution Q. Furthermore, Q is the only Kullback-Leibler equidistant vector from the vertices of S because the minimum of a convex-$\cup$ function is unique. □

Theorem 2.2 provides the solution to the problem stated in (2.13) for a special type of polytope. We will use this theorem to find the solution for all polytopes. For this, we need to show another property of a function $f(.)$ defined as the maximum of component convex-$\cup$ functions.

Lemma 2.4 *Let $f_1, f_2, \ldots, f_r$ be convex-$\cup$ functions defined over the convex region $\mathcal{C}$. Let $f(.)$ be a function defined for every vector $\underline{v}$ in $\mathcal{C}$ by $f(\underline{v}) = \max_{k=1\ldots r} f_k(\underline{v})$ and let $\underline{v}_0$ be the vector for which $f(.)$ is minimized over $\mathcal{C}$. Then one of the two following statements must hold:*

1. *$\exists i$ such that $\min_{\underline{v}\in\mathcal{C}} f_i(\underline{v}) = f_i(\underline{v}_0)$ and $f_j(\underline{v}_0) < f_i(\underline{v}_0)$ for all $j \neq i$*
2. *$\exists i$ and j such that $f_i(\underline{v}_0) = f_j(\underline{v}_0) = f(\underline{v}_0)$.*

Proof: The first statement holds if $\underline{v}_0 = \arg\min_{\underline{v}} f_i(\underline{v})$ and $f_j(\underline{v}_0) < f_i(\underline{v}_0)$ for all $i \neq j$. In this case, $f(\underline{v}) = \max_j f_j(\underline{v}) = f_i(\underline{v})$ in a neighborhood of $\underline{v}_0$ because a convex-$\cup$ function is continuous. Therefore, since the minimum of $f_i(.)$ is attained for $\underline{v}_0$, $f(.)$ has a local minimum at $\underline{v}_0$. But this local minimum is also the global minimum of $f(.)$ because $f(.)$ is convex-$\cup$ due to Proposition 2.6.

If the first statement does not hold, there must be at least two component functions $f_i(.)$ and $f_j(.)$ whose values are equal for the argument $\underline{v}_0$ because if $f(.)$ was equal to only one component function $f_i(.)$ at $\underline{v}_0$ then that function would have a local minimum at $\underline{v}_0$ which would necessarily also be its global minimum. □

The following proposition is one of the fundamental inequalities of Information Theory [29, Theorem 2.6.3].

Proposition 2.7 *$D(P||Q) \geq 0$ with equality if and only if $P = Q$.*

Proposition 2.7 shows that, except for a trivial polytope consisting of only one probability distribution, the minimum of the function $f(.) = \max_k D(P_k||.)$ is never located at the minimum of an individual divergence $D(P_i||.)$. The individual divergences in our problems attain their minima at the vertices of the polytope for which their value is zero while the other

divergences are necessarily positive. Therefore, it is always the second case in Lemma 2.4 that holds for the function $f(.)$ in our problem.

Using this fact, we can finally state the general solution of the problem stated in (2.13).

Theorem 2.3 (Corollary to Theorem 2.2) *Let the polytope $S =< P_1, P_2, \ldots, P_r >$ in $\mathcal{P}_N$ be such that there is no Kullback-Leibler equidistant distribution in the open polytope $\breve{S}$ as required by Theorem 2.2 and let $\mathcal{V}$ be the set $\{P_1, \ldots, P_r\}$ containing the vertices of S. Then there exists a subset $\mathcal{V}'$ of $\mathcal{V}$ containing at least two elements such that the vertices in $\mathcal{V}'$ form a polytope S' for which there is a Kullback-Leibler equidistant distribution $Q \in \breve{S}'$, i.e.,*

$$\begin{aligned} D(P_i||Q) &= C \text{ for } i = 1, \ldots r \text{ and } P_i \in \mathcal{V}', \text{ and} \\ D(P_i||Q) &< C \text{ for } i = 1, \ldots r \text{ and } P_i \notin \mathcal{V}'. \end{aligned}$$

This distribution Q is the solution of the problem stated in (2.13) for the polytope S.

Proof: If there exists a subset $\mathcal{V}'$ which satisfies the conditions in Theorem 2.3, then

$$f(.) \stackrel{\text{def}}{=} \max_{P \in \mathcal{V}} D(P||.) = f'(.) \stackrel{\text{def}}{=} \max_{P \in \mathcal{V}'} D(P||.)$$

in a neighborhood of the distribution Q which is Kullback-Leibler equidistant from the vertices in $\mathcal{V}'$. In this neighborhood, $f'(.)$ has its global minimum at Q, so $f(.)$ must have a local minimum. But, since $f(.)$ is convex-$\cup$, this local minimum is also its global minimum.

It remains to be shown that such a subset $\mathcal{V}'$ always exists. We have seen in Proposition 2.6 that a solution Q to (2.13) always exists and Lemma 2.4 and Proposition 2.7 have shown that there is a set $\mathcal{V}'' \subset \mathcal{V}$ containing at least two distributions for which $D(P||Q) = C$ for all $P \in \mathcal{V}''$. Consider the polytope S'' formed by the distributions in $\mathcal{V}''$. Since $D(P||Q) < C$ for all $P \in \mathcal{V}$ which are not in $\mathcal{V}''$, the solution of (2.13) for the polytope S'' must be the same as the solution Q for the polytope S. Q is Kullback-Leibler equidistant from the vertices of S'' by definition. The only thing remaining to be proved to show that S'' and Q satisfy the requirements of Theorem 2.2 is to show that the solution Q lies in the open polytope $\breve{S}''$.

Suppose that Q is outside the open polytope $\breve{S}''$. This brings us back to the equality (2.23) and the inequalities (2.22) in the proof of Theorem 2.2. Only this time, we have assumed that there is no $\underline{\mu} > \underline{0}$ which satisfies

$\mathbf{P}^T\underline{\mu} = Q$, since Q is not in the open polytope $\breve{S}''$. There can be no $\underline{\mu} > \underline{0}$ for which $\mathbf{P}^T\underline{\mu} = -Q$ or $\mathbf{P}^T\underline{\mu} = \underline{0}$ because the columns of $\mathbf{P}$ are probability distributions and hence $P_i \geq^* \underline{0}$ for each column P_i of $\mathbf{P}$. Thus, by Lemma 2.3, there exists a vector $\hat{\underline{d}}$ which satisfies (2.23) and (2.22). In other words, there is a direction in $\mathcal{P}_N$ relative to which all the component divergences $D(P||.)$ for $P \in \mathcal{V}''$ are decreasing for Q. Therefore, there can be no minimum of $f(.)$ located at Q if Q lies outside the open polytope $\breve{S}''$. □

Optimal universal coding for some regular polytopes

Now that we have shown the nature of the solution of the problem we stated in (2.13) for all types of polytopes, we would like to proceed to find this solution for the polytopes that we are practically interested in. Unfortunately, Theorem 2.3 does not help us to *find* the solution Q for any particular polytope. We will have to wait until our discussion of Gallager's Redundancy-Capacity Theorem in the next section to obtain such a constructive method. Gallager's approach differs from ours in that it begins by exchanging the "min-max" problem we stated in (2.13) for an equivalent "max-min" problem.

Before we surrender the solution of our problem to Gallager's masterful guidance, we use the approach we developed so far to state the solution of our problem for some of the polytopes we are interested in, using a property that we now describe.

Proposition 2.8 *A regular polytope $S =< P_1, P_2, \ldots, P_N >$ in $\mathcal{P}_N$ always has a unique Kullback-Leibler equidistant distribution Q in $\mathcal{P}_N$.*

Proof: The proof of this proposition will show us how to compute the Kullback-Leibler equidistant distribution Q. Any vector Q which is Kullback-Leibler equidistant from the vertices of S must satisfy the system of equations

$$\begin{aligned}
p_{11}\log\frac{1}{q_1} + p_{12}\log\frac{1}{q_2} + \ldots + p_{1N}\log\frac{1}{q_N} &= C + H(P_1)\\
p_{21}\log\frac{1}{q_1} + p_{22}\log\frac{1}{q_2} + \ldots + p_{2N}\log\frac{1}{q_N} &= C + H(P_2)\\
&\vdots \quad \vdots \quad \vdots\\
p_{N1}\log\frac{1}{q_1} + p_{N2}\log\frac{1}{q_2} + \ldots + p_{NN}\log\frac{1}{q_N} &= C + H(P_N)
\end{aligned}$$

where p_{ij} denotes the j'th element of the probability vector P_i. Since $\log(1/x)$ is a bijective function mapping the positive real numbers onto $\mathbb{R}$,

we can perform the variable transformation $l_i = \log(1/q_i)$ for $i = 1, \ldots, N$. Writing $\underline{l}$ for the vector whose i-th component is l_i, $\mathbf{P}$ for the matrix whose i-th row is the probability distribution P_i, and $\underline{h}$ for the vector whose i-th element h_i is the entropy $H(P_i)$, we can rewrite this new system of equations in matrix form as

$$\mathbf{P}\underline{l} = C\underline{1}_N + \underline{h}.$$

The matrix $\mathbf{P}$ is invertible because its rows are the vertices of a regular polytope. Therefore, there is always a unique solution to the equation stated and this solution can be written as

$$\underline{l} = C\mathbf{P}^{-1}\underline{1}_N + \mathbf{P}^{-1}\underline{h}.$$

Since $\mathbf{P}$ is a stochastic matrix, it satisfies $\mathbf{P}\underline{1} = \underline{1}$. Its inverse similarly satisfies $\mathbf{P}^{-1}\underline{1} = \mathbf{P}^{-1}(\mathbf{P}\underline{1}) = (\mathbf{P}^{-1}\mathbf{P})\underline{1} = \underline{1}$. Thus, we can simplify this solution to obtain

$$\underline{l} = C\underline{1}_N + \mathbf{P}^{-1}\underline{h}. \tag{2.24}$$

If we define $\underline{g} \stackrel{\text{def}}{=} \mathbf{P}^{-1}\underline{h}$ and invert the variable transformation, each component of the solution vector Q can be written as

$$q_k = 2^{-C}2^{-g_k}.$$

For any C, each component of Q is positive and there is exactly one choice for C which will yield a vector Q whose components sum to one, i.e., $Q \in \mathcal{P}_N$. □

Using Proposition 2.8 and the construction method stated in its proof, we can compute the Kullback-Leibler equidistant distribution $Q \in \mathcal{P}_N$ for any regular polytope S. If we are lucky and $Q \in \breve{S}$, then by Theorem 2.2 we know that we have found the solution of (2.13) for S. We will try this approach on two of the polytopes we are interested in.

Consider the polytope $\mathcal{P}_N$ itself. Its vertices form the $N \times N$ identity matrix and the entropies of its vertices are all zero, i.e., $\underline{h} = \underline{0}_N$. Using (2.24), we obtain a Kullback-Leibler equidistant distribution Q whose components are

$$q_k = 2^{-C} = \frac{1}{N} \text{ for } k = 1 \ldots N.$$

This distribution lies inside the open polytope $\breve{\mathcal{P}}_N$ so it is the solution of the problem stated in (2.13). In other words, if you know nothing

about the sources whose output you are encoding except that the size of their alphabet is N, then it is best to design your code for the uniform distribution of alphabet size N. The corresponding worst-case redundancy is $\rho = \log_2 N$. Though this is an obvious result, it is pleasant that our theory gives the answer we expected for this simple case.

Now consider the polytope $\mathcal{M}_N$ of all monotone non-increasing probability distributions. We saw earlier that this set corresponds to the regular polytope

$$\mathcal{M}_N =< \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1/N \\ 1/N \\ 1/N \\ \vdots \\ 1/N \end{bmatrix} > .$$

By transforming the corresponding system of equations, we obtain

$$q_k = 2^{-C} \frac{(k-1)^{k-1}}{k^k} \text{ for } k = 1 \ldots N, \tag{2.25}$$

where C must be chosen such that $\sum_{k=1}^{N} q_k = 1$. The corresponding worst-case redundancy is

$$\rho = D(P_1||Q) = \log_2 \frac{1}{Q(1)} = \log_2 \sum_{k=1}^{N} \frac{(k-1)^{k-1}}{k^k}.$$

For $k = 2, \ldots, N$, we can write (2.25) as

$$q_k = \frac{2^{-C}}{k-1} \left(1 - \frac{1}{k}\right)^k.$$

Since this distribution Q is monotone and strictly decreasing, it lies in the open polytope $\breve{\mathcal{M}}^N$ and is thus the solution of (2.13) for the polytope of all monotone non-increasing probability distributions.

Example: The optimal coding distribution when coding for a binary memoryless source when the only thing known about the source is that

$$P_X(0) \geq P_X(1)$$

is $Q(0) = .8$ and $Q(1) = .2$. The optimal coding distribution for a ternary monotone source is $Q = 108/151 \cdot [1, 1/4, 4/27]^T$. For a quaternary source, it is $Q = .6651 \cdot [1, 1/4, 4/27, 27/256]^T$.

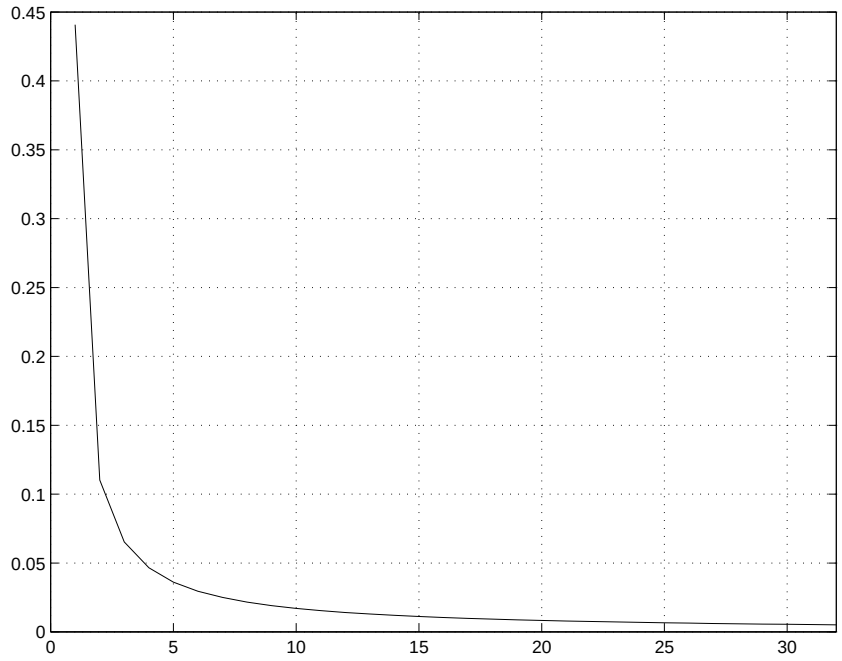


Figure 2.2: The optimal distribution for universal coding over the polytope $\mathcal{M}_{32}$

The optimal coding distribution for a monotone source of alphabet size 32 is plotted in Figure 2.2. The corresponding worst-case redundancy is $\boldsymbol{\rho = 1.1815}$.

The distribution in (2.25) was discovered by Ryabko, as reported in Krichevsky's book [31]. This distribution has a very interesting character. When k gets large, the term $(1-1/k)^k$ tends towards $1/e$. Therefore, the probabilities for large k can be approximated as

$$q_k \approx \frac{2^{-C}}{e} \cdot \frac{1}{k-1}.$$

Now imagine that you are the last symbol of a monotone non-increasing probability distribution and you harbor exasperated dreams of grandeur and jealousy for the symbols that precede you. What is the best thing that could happen to you? You dream of being the last symbol in a uniform distribution, obtaining a probability of $1/N$ like all other symbols. The expression we wrote above assigns roughly a constant times that value to the last symbol of the optimal coding distribution. In a way, the optimal distribution is very kind in its nature to the last symbols of the monotone distributions, being of the form $1/k$ that favors them most. In its tail, it scales this distribution down by a constant factor and distributes those earnings to the first symbols in the distribution. This explains why the geometric (or exponential) distribution, which is often assumed in situ-

ations when the true distribution is unknown, is a very bad choice for universal coding. Contrary to the optimal distribution we derived, the geometric distribution assigns negligible probabilities to the last symbols in the distribution.

2.4 The Redundancy-Capacity Theorem

We now proceed to describe Gallager's solution for the problem stated in (2.13). Gallager presented his solution in a paper he submitted to the IEEE Transactions on Information Theory in September 1976 [12]. The paper was never revised and was turned into an internal report in 1979. Gallager's result is simple and elegant and its implications are fundamental to the theory of universal coding[4]. Partly because the paper is unpublished and partly because the result is relevant to our work, we allow ourselves to reproduce Gallager's approach here.

We return first to our original statement of the problem of optimal arithmetic coding for a set $\mathcal{S}$ of probability distributions. In (2.3), we stated the problem of finding the optimal coding distribution for a set $\mathcal{S}$ of source probability distributions $P_{Y|X}(.|x)$ when a prior $P_X(.)$ indicates the probability of occurrence of each source in $\mathcal{S}$. The problem was one of finding the coding distribution $Q(.)$ which would minimize the expected redundancy over $\mathcal{S}$. When no prior is known as in the cases we are interested in, we argued that the optimal coding distribution should be the one which minimizes the worst-case redundancy as expressed in (2.4). Gallager's argument relies on the introduction of a dummy prior distribution which allows us to modify the problem in (2.4) into a form that resembles the problem in (2.3).

Assume we are about to encode the output of an unknown source from a finite set $\mathcal{S}$ containing r sources. Let X be the indicator random variable whose value is the index of the source whose output is being encoded. The following equation shows how the effect of the dummy prior can be canceled by maximizing over the choices for the prior distribution $P_X(.)$ to obtain the same result as in the min-max problem stated in (2.4).

$$\max_{P_X} \sum_{x=1}^{r} P_X(x) D(P_{Y|X}(.|x)||Q) = \max_{x} D(P_{Y|X}(.|x)||Q). \tag{2.26}$$

[4] With all due respect, I cannot understand why Gallager chose never to publish this fundamental result. While far more papers are published than should be, here is a paper whose main result belongs in every basic book on Information Theory, but students have to fight for a third hand copy on the "black market" rather than being able to obtain it through official channels.

This is true because the distribution P_X that maximizes the left side of (2.26) can assign non-zero probabilities only to values of X for which the divergence is equal to the maximum. We must keep in mind that the prior distribution P_X has no meaning in reality as no true prior is given on the r sources. It is introduced only to help us in the derivation. The sum in (2.26) can be expanded as follows

$$\begin{aligned}\sum_{x=1}^{r} P_X(x) D(P_{Y|X}(.|x)||Q) &= \sum_{x=1}^{r}\sum_{y=1}^{N} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)\cdot Q(y)} \\ &= D(P_{XY}||P_X \cdot Q). \end{aligned} \tag{2.27}$$

For the set $\mathcal{S}$, (2.4) can now be re-written using (2.26) and (2.27) to give

$$\rho = \min_{Q} \max_{P_X} D(P_{XY}||P_X \cdot Q). \tag{2.28}$$

The proof of Gallager's theorem relies on the existence of a saddle point[5] for the divergence $D(P_{XY}||P_X \cdot Q)$. The existence of a saddle point guarantees that the solution of the min-max problem is equal to the solution of the equivalent max-min problem when in both problems the divergence is maximized over all choices of P_X and minimized over all choices of Q. Before we prove that there is a saddle point, we investigate the max-min problem briefly. We can write the max-min problem as

$$\xi = \max_{P_X} \min_{Q} D(P_{XY}||P_X \cdot Q). \tag{2.29}$$

To see what happens to the inner minimization, we rewrite the divergence

[5] If f is a real-valued function from any non-empty product set $C \times D$, a *saddle point* of f with respect to maximizing over C and minimizing over D is a point $(u_0, v_0) \in C \times D$ for which f satisfies

$$f(u, v_0) \leq f(u_0, v_0) \leq f(u_0, v), \forall u \in C, \forall v \in D.$$

If f has a saddle point (u_0, v_0), then (see [35, Lemma 36.2])

$$\max_{u \in C} \min_{v \in D} f(u,v) = \min_{v \in D} \max_{u \in C} f(u,v) = f(u_0, v_0),$$

where the maxima and minima can be replaced by suprema and infima where appropriate.

as

$$\begin{aligned} D(P_{XY}||P_X \cdot Q) &= \sum_{x=1}^{K}\sum_{y=1}^{N} P_{XY}(x,y)\log\frac{P_{XY}(x,y)}{P_X(x)\cdot Q(y)} \\ &= \sum_{x=1}^{K}\sum_{y=1}^{N} P_{XY}(x,y)\log\frac{P_{XY}(x,y)}{P_X(x)\cdot P_Y(y)}\frac{P_Y(y)}{Q(y)} \\ &= D(P_{XY}||P_X\cdot P_Y) + \sum_{x=1}^{K}\sum_{y=1}^{N} P_{XY}(x,y)\log\frac{P_Y(y)}{Q(y)} \\ &= D(P_{XY}||P_X\cdot P_Y) + \sum_{y=1}^{N} P_Y(y)\log\frac{P_Y(y)}{Q(y)} \\ &= I(X;Y) + D(P_Y||Q), \end{aligned} \tag{2.30}$$

whose minimum is attained when $D(P_Y||Q) = 0$ for $Q = P_Y$ because of Proposition 2.7, where $P_Y(y) \stackrel{\text{def}}{=} \sum_x P_{Y|X}(y|x)P_X(x)$ for all y. Thus, the max-min problem can be expressed as

$$\xi = \max_{P_X} I(X;Y), \tag{2.31}$$

which is equivalent to the expression for the capacity of a channel. Theorem 4.5.1 in Gallager's book [30] gives an important property of channel capacity which will be of use here. This theorem is repeated here in our notation.

Theorem 2.4 *The necessary and sufficient condition on a probability distribution P_X to maximize $D(P_{XY}||P_X \cdot Q)$ is that, for some number ξ,*

$$D(P_{Y|X}(.|x)||Q) = \xi \text{ for all } x \text{ with } P_X(x) > 0, \tag{2.32}$$

$$D(P_{Y|X}(.|x)||Q) \leq \xi \text{ for all } x \text{ with } P_X(x) = 0. \tag{2.33}$$

We now show that $D(P_{XY}||P_X\cdot Q)$ indeed has a saddle point. We have already shown that $\min_Q D(P_{XY}||P_X \cdot Q)$ is achieved for $Q = P_Y$ in all cases and in particular when P_X is a distribution which achieves capacity. Furthermore, when $Q = P_Y$, $\max_{P_X} D(P_{XY}||P_X \cdot P_Y)$ is achieved for any capacity-achieving distribution P_X, which, by Gallager's Theorem 2.4, is positive only for those x for which $D(P_{Y|X}(.|x)||Q)$ is equal to the maximum redundancy. Therefore, the function $D(P_{XY}||P_X \cdot Q)$ has a saddle point when P_X is the capacity-achieving distribution and Q is equal to the corresponding channel output distribution P_Y. We have finally proved the theorem which relates universal coding to channel capacity:

Theorem 2.5 (Gallager's Redundancy-Capacity Theorem) *The min-max problem of universal coding stated in (2.28) is equivalent to the max-min problem of channel capacity stated in (2.29) and (2.31). The optimal worst-case redundancy ρ can be obtained by solving*

$$\rho = \max_{P_X} I(X;Y)$$

where $P_{Y|X}(y|k)$ is the probability distribution of the k-th source. The optimal coding distribution Q can be computed from the distribution P_X which achieves the maximum in (2.31) as

$$Q(y) = \sum_{x=1}^{K} P_{Y|X}(y|x) P_X(x)$$

for $y = 1 \ldots N$.

Thus, the problem of finding the optimal coding distribution for universal arithmetic coding over a finite set of distributions is equivalent to the problem of computing the capacity of a channel. All the results that we derived in the previous sections can be applied to compute the capacity of a channel. Conversely, all the methods known to compute the capacity of a channel can be used to compute the optimal coding distribution for universal arithmetic coding. In particular, the algorithm developed by Arimoto [5] and Blahut [7] to compute the capacity of a discrete memoryless channel can also be used to compute the optimal coding distribution for the polytopes $\mathcal{P}_{N,\underline{f},c}$ and $\mathcal{M}_{N,\underline{f},c}$. Although the Arimoto-Blahut algorithm converges towards the capacity for every discrete memoryless channel, its rate of convergence is slow in situations where the channel has a large input alphabet but only a small subset of the input alphabet obtains a non-zero probability in its capacity-achieving distribution. Remember that the polytope $\mathcal{M}_{N,\underline{f},c}$ can have up to $N^2/4$ vertices for an alphabet size of N. Therefore, computing the optimal coding distribution over $\mathcal{M}_{N,\underline{f},c}$ is equivalent to computing the capacity of a channel with up to $N^2/4$ input values and N output values. As a result, the Arimoto-Blahut algorithm converges so slowly as to become impractical for alphabet sizes greater than about 32. In practice, we are interested in the case $N = 256$. We will use the insight we gained in the previous sections to construct an alternative to the Arimoto-Blahut algorithm in the next section.

Optimal coding distribution with a prior

The solution of the problem stated in (2.3) is implicit in the proof of Gallager's redundancy-capacity Theorem 2.5 when $\mathcal{S}$ is a *finite* class of r

probability distributions and a prior distribution P_X is given on $\mathcal{S}$. Using (2.27) and (2.30), we can write (2.3) as

$$Q = \arg\min_{Q'}[I(X;Y) + D(P_Y||Q)] \tag{2.34}$$

where P_Y is derived from the given distributions P_X and $P_{Y|X}$ and its value for every y is

$$P_Y(y) = \sum_{x \in \mathcal{S}} P_{Y|X}(y|x)P_X(x).$$

The minimum of (2.34) is attained when $D(P_Y||Q) = 0$ for $Q' = P_Y$ because of Proposition 2.7. We have proved the following proposition:

Proposition 2.9 *The optimal coding distribution for universal arithmetic coding over a finite set $\mathcal{S}$ of r probability distributions $P_{Y|X}(.|x)$ for which a prior P_X over $\mathcal{S}$ is known is*

$$Q(y) = \sum_{x \in \mathcal{S}} P_{Y|X}(y|x)P_X(x)$$

for all y. If no prior is known but a uniform prior is assumed, the corresponding optimal coding distribution becomes

$$Q(y) = \frac{1}{r} \sum_{P_Y \in \mathcal{S}} P_Y(y).$$

The solution of the problem stated in (2.3) for a polytope of probability distributions cannot be reduced to a problem over the vertices of the polytope in the same manner as for (2.4).

2.5 An Iterated Arimoto-Blahut Algorithm

In the previous sections, we learned a few rules to follow when seeking the optimal coding distribution for universal arithmetic coding over any set of probability distributions. These can be summarized as follows:

- any distribution which can be expressed as a convex combination of other distributions in the set can be eliminated, since such a distribution will have no influence on the solution,
- if the distributions in the set form the vertices of a regular polytope, then we can use (2.24) and test whether the resulting distribution is contained in the corresponding open polytope.

Using these two rules, we found the solution of (2.13) for the set $\mathcal{P}_N$ of all distributions of size N and for the set $\mathcal{M}_N$ of all monotone non-increasing probability distributions of size N.

Furthermore, we have seen what conditions a probability distribution Q must fulfill to be the solution of (2.13). However, we did not find a method to determine a distribution Q which fulfills these conditions. Gallager's redundancy-capacity theorem taught us that we can use any method designed to compute the capacity of a channel to compute the solution of (2.13). Gallager's theorem also showed that the results we developed with obscure probability-geometric argumentation could be obtained in a simple and elegant derivation requiring little more than three pages. Using the terminology of Chapter 1, we can say that we took the "direct approach" to solving the min-max problem of optimal universal coding, while Gallager used a "Shannon bypass" for the same problem. However, we will use the insight we gained through our geometric approach to develop a variant of the Arimoto-Blahut algorithm to find the optimal coding distribution over polytopes which have too many vertices for the Arimoto-Blahut algorithm to handle practically, or alternatively, to find the capacity of a channel whose input alphabet is too large for the Arimoto-Blahut algorithm to handle practically.

The Arimoto-Blahut algorithm is known to converge towards the capacity-achieving input distribution for all channel transition matrices. The algorithm adapts the parameters of the channel input distribution in an iterative process. In situations where the input alphabet is very large but only a few input symbols have non-zero probabilities in the final solution, the algorithm converges too slowly to be of practical use. If we knew which symbols of the input alphabet will end up with non-zero probabilities, we could use the Arimoto-Blahut algorithm with only these symbols as an input alphabet and obtain the same solution as for the entire input alphabet, but the algorithm would converge much faster. The algorithm we present starts with an input alphabet consisting only of two symbols, and grows the alphabet until it includes all the symbols with non-zero probabilities. The Arimoto-Blahut algorithm is used at every stage in this process, but it is never used on the entire input alphabet.

The first step of the algorithm will make use of the following proposition.

Proposition 2.10 *For a polytope $S =< P_1, P_2 >$ with only two vertices, there is always a unique Kullback-Leibler equidistant distribution Q in the corresponding open polytope $\breve{S}$.*

Proof: The polytope $< P_1, P_2 >$ is the line segment joining P_1 and P_2. Each point Q on this segment can be expressed as

$$Q(\theta) = \theta P_1 + (1 - \theta)P_2 \text{ for } \theta \in [0, 1].$$

The function of one variable $D(P_1||Q(\theta))$ is monotone decreasing in θ (its derivative is non-positive) and attains the value 0 for $\theta = 1$. The function $D(P_2||Q(\theta))$ is monotone increasing in θ and it starts at the value 0 for $\theta = 0$. Therefore, these two functions must meet exactly once on the line segment joining P_1 and P_2. □

In the simple case of a polytope $< P_1, P_2 >$ with two vertices, it is not necessary to use the Arimoto-Blahut algorithm to determine the Kullback-Leibler equidistant distribution Q. Instead, a binary search algorithm can be used[6]. The binary search algorithm works faster and better than the Arimoto-Blahut algorithm for a polytope with two vertices.

The Iterated Algorithm

We are ready to describe the iterated algorithm which we will use to compute the optimal coding distribution Q for a polytope $S =< P_1, P_2, \ldots, P_r >$ with a large number of vertices (or, alternately, to compute the capacity of a channel with many inputs):

1. Find an i and a j which maximize $D(P_i||Q_{ij})$ where Q_{ij} is the Kullback-Leibler equidistant distribution in the open polytope $> P_i, P_j <$. Let S' be the resulting polytope with two vertices and Q' the corresponding Kullback-Leibler equidistant distribution.

2. Compute $D(P_k||Q')$ for all vertices P_k of S which are not vertices of S'. If all these divergences are smaller than or equal to the divergence from the vertices of S' to Q', then Q' is the optimal coding distribution for the polytope S by virtue of Theorem 2.3 and the algorithm can be terminated.

[6]The algorithm can be implemented as follows

1. left := 0 ; right := 1
2. θ := (right + left) / 2; $Q := \theta P_1 + (1 - \theta)P_2$
3. IF $(D(P_1||Q) > D(P_2||Q))$ THEN left := θ ELSE right := θ
4. IF $(|D(P_1||Q) - D(P_2||Q)| < \epsilon)$ THEN STOP
5. GOTO step 2

3. Add one of the vertices P_k of S which maximized $D(P_k||Q')$ to the set of vertices of S'. Use the Arimoto-Blahut algorithm to recompute the optimal coding distribution Q' for the new polytope S'. Return to step 2.

Whenever the algorithm terminates at step 2, we can be sure that the solution obtained is the correct solution since it fulfills the conditions stated in Theorem 2.3. The algorithm will always terminate and discover the correct solution, because in the worst case it will do so after including all the vertices of S into the set of vertices of S', in which case the last iteration corresponds to the Arimoto-Blahut algorithm applied to the polytope S.

Though the algorithm is bound to terminate eventually, we are not sure whether it will generate only those vertices with non-zero prior probabilities in all cases. We will say that the algorithm has introduced unwanted letters if, after it terminates, it has assigned zero probabilities to some of the vertices in its partial polytope S'. So far, we have not been able to find an example of a polytope or a channel transition matrix for which the algorithm introduces unwanted letters. In particular, the algorithm was successful for the polytope $\mathcal{M}_{N,\underline{f},c}$, thereby achieving a considerable saving in computing time compared with the conventional Arimoto-Blahut algorithm. The algorithm achieves no saving when computing the optimal coding distribution for the polytope $\mathcal{P}_{N,\underline{f},c}$ in general because, as it turns out, all the vertices of this polytope may have a non-zero probability in its capacity-achieving prior distribution. However, the algorithm can still be used to compute a sub-optimal solution for $\mathcal{P}_{N,\underline{f},c}$ when the number of vertices is too large to compute the optimal distribution using the conventional Arimoto-Blahut algorithm. We will discuss these results shortly.

Before we proceed, a few words must be said about the implementation of the iterated algorithm. We learned in Proposition 2.10 that there is always a Kullback-Leibler equidistant distribution in the open polytope corresponding to a polytope with two vertices. The obvious way to execute step 1 is to compute the Kullback-Leibler equidistant distribution for every pair of vertices and find the pair that maximizes the corresponding Kullback-Leibler distance. However, even using the fast binary search approach, this is a task that will use a considerable amount of computing time for polytopes with many vertices. In all the examples that we tested, we observed that it was sufficient to use a two-pass algorithm which can be described as follows:

1. Pick i' at random,

2. find j that maximizes $D(P_{i'}||Q_{i'j})$ where $Q_{i'j}$ is the Kullback-Leibler

equidistant distribution from $P_{i'}$ and P_j in the open polytope $> P_{i'}, P_j <$,

3. find i that maximizes $D(P_i||Q_{ij})$ where Q_{ij} is the Kullback-Leibler equidistant distribution from P_i and P_j in the open polytope $> P_i, P_j <$.

The resulting P_i, P_j and Q_{ij} can be used as a starting point in the iterated algorithm. In some cases, we can guess which two vertices P_i and P_j maximize $D(P_i||Q_{ij})$ by observing the solutions for a few polytopes, thereby simplifying the iterated algorithm by eliminating its first step. This will be the case for the polytope $\mathcal{M}_{N,\underline{f},c}$.

Finally, from the second iteration onwards, the prior distribution obtained in the previous iteration can be used to provide an initial prior for the Arimoto-Blahut algorithm in step 3 of the iterated algorithm. However, the new vertex added must be assigned a non-zero probability in this initial prior, because the Arimoto-Blahut algorithm cannot adapt an initial parameter of zero (see [5]). In our implementation, we tried to solve this problem in various ways. For example, we tried assigning a probability of $1/r'$ to the new vertex and rescaling the prior obtained at the previous iteration by a factor $(1 - 1/r')$ for the remaining vertices, where r' is current the number of vertices of the polytope S'. Surprisingly, none of the initial priors we tested allowed a significant improvement over using simply the uniform distribution at each iteration as an initial prior for the Arimoto-Blahut algorithm.

The Optimal Coding Distribution for $\mathcal{M}_{N,\underline{f},c}$

The optimal coding distribution was determined for $\mathcal{M}_{N,\underline{f},c}$ when $\underline{f} = [0, 1, \ldots, N-1]^T$ for various values of N and c. Figure 2.3 shows the optimal coding distribution for $\mathcal{M}_{N,\underline{f},c}$ for $N = 32$ for various values of c. For $c = 8$, $\mathcal{M}_{N,\underline{f},c}$ has 241 vertices. Of those, only 10 vertices have non-zero probabilities in the solution. The iterated algorithm required 5.32 seconds of CPU-time to determine this solution, while the conventional Arimoto-Blahut algorithm used 764 seconds of CPU-time to converge to the same solution. For $N = 64$ and $c = 16$, $\mathcal{M}_{N,\underline{f},c}$ has 993 vertices. The iterated algorithm terminated after 47 seconds of CPU-time, after finding that the solution depended only on 19 of those vertices. The conventional Arimoto-Blahut algorithm had to be interrupted after running for four hours without finding the solution.

We observed that the two vertices found in the first step of the iterated

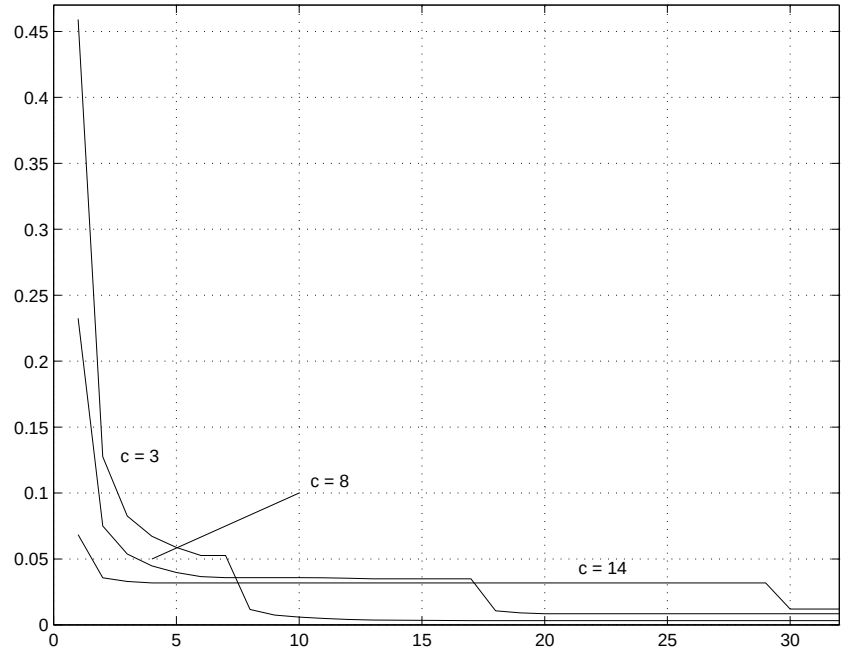


Figure 2.3: The optimal coding distribution for $\mathcal{M}_{N,\underline{f},c}$ for $N = 32$ and $c = 3, 8$ and 14.

algorithm for the polytope $\mathcal{M}_{N,\underline{f},c}$ are always

$$P_1 = [1 - \alpha, \alpha, \alpha, \ldots, \alpha]^T \tag{2.35}$$

and

$$P_2 = [\beta, \ldots, \beta, \beta', 0, \ldots, 0]^T \tag{2.36}$$

where α, β and β' must be chosen such that P_1 and P_2 are monotone non-increasing probability distributions and $\underline{f}^T \cdot P_i = c$ for $i = 1, 2$. This observation effectively eliminates the need to perform the first step of the iterated algorithm when using it to determine the optimal coding distribution for the polytope $\mathcal{M}_{N,\underline{f},c}$.

It is interesting to observe how the sub-optimal coding distributions Q' evolve from one iteration to the next in the iterated algorithm as more and more vertices are taken into consideration. Figure 2.4 shows the coding distributions obtained by the iterated algorithm for $N = 32$ and $c = 8$ after two, three, four and five vertices have been included into the set of vertices of the polytope S'. Figure 2.5 shows the worst-case redundancy $\max_{P \in S} D(P||Q')$ as a function of the number of vertices of the polytope S' for which Q' is the optimal coding distribution. The figure shows that the coding distribution obtained after the fourth iteration based on a polytope S' with only five vertices already achieves a worst-case redundancy which is very close to the worst-case redundancy of the optimal coding

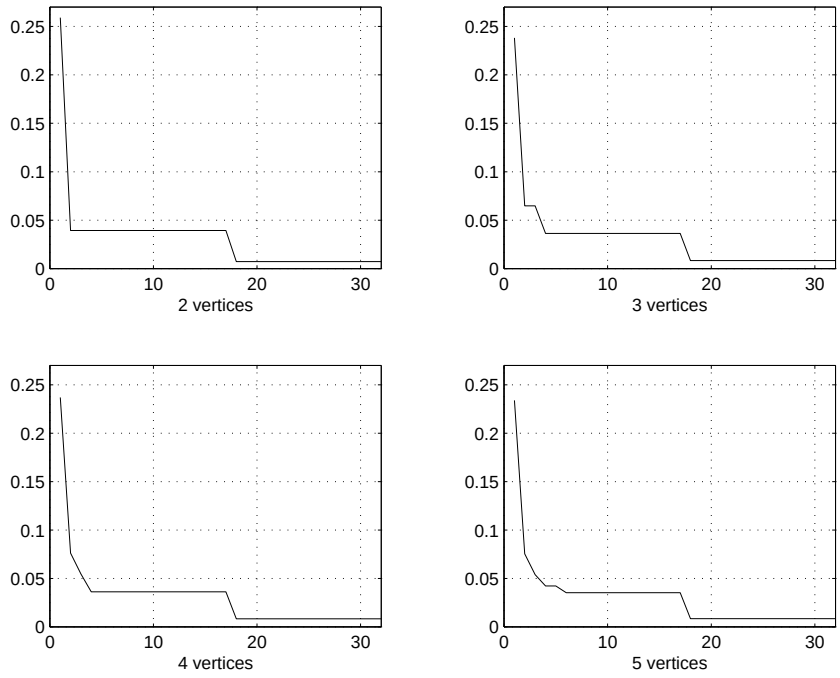


Figure 2.4: Coding distributions after partial completion of the iterated algorithm for $N = 32$ and $c = 8$.

distribution, which depends on 10 vertices. This provides the rationale for using the iterated algorithm to determine a coding distribution for universal coding over a polytope with many vertices even when the optimal solution assigns non-zero probabilities to all the vertices, as is often the case for the polytope $\mathcal{P}_{N,\underline{f},c}$. When the conventional Arimoto-Blahut algorithm is unable to find the optimal solution in a reasonable time, we can use the iterated algorithm to produce a sub-optimal solution whose worst-case redundancy will be close to that of the optimal solution. This effect becomes more pronounced as the alphabet size increases. This can be seen in Figure 2.6 which plots the worst-case redundancy $\max_{P \in S} D(P||Q')$ as a function of the number of vertices for the polytope $\mathcal{M}_{N,\underline{f},c}$ where $N = 256$ and $c = 36$. This polytope has $13'105$ vertices. The iterated algorithm obtains a coding distribution whose worst-case redundancy is comparable to that of the optimal solution after including only 20 of the 97 significant vertices of the optimal solution. Finally, Figure 2.7 shows the worst-case redundancy ρ for the optimal coding distribution for the polytope $\mathcal{M}_{N,\underline{f},c}$ for $N = 32$ as a function of the average c. Figure 2.8 shows the same graph for the polytope $\mathcal{P}_{N,\underline{f},c}$, again for $N = 32$.

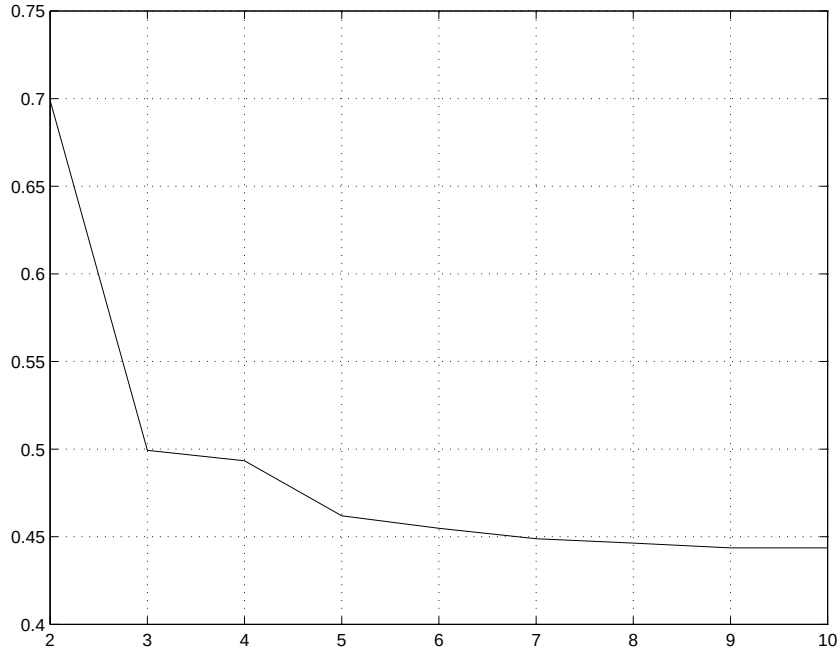


Figure 2.5: Worst-case redundancy as a function of the number of vertices in S' for $N = 32$ and $c = 8$.

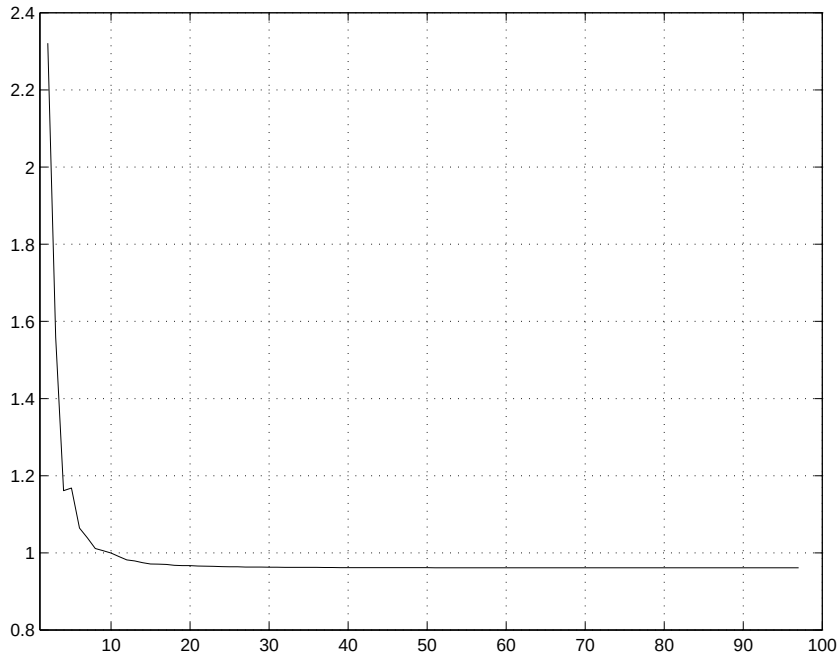


Figure 2.6: Worst-case redundancy as a function of the number of vertices in S' for $N = 256$ and $c = 36$.

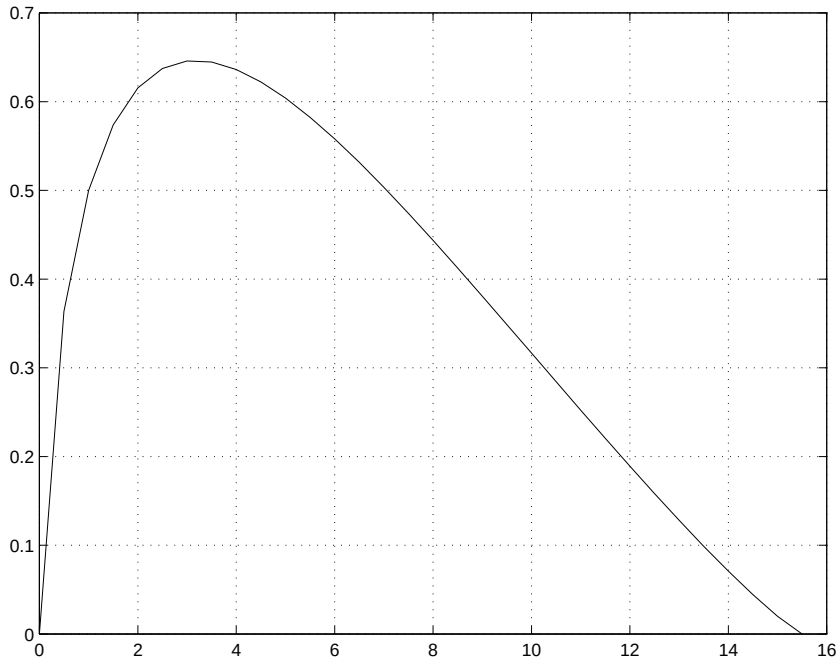


Figure 2.7: Worst-case redundancy ρ of the optimal coding distribution for $\mathcal{M}_{N,\underline{f},c}$ as a function of the average c for $N = 32$.

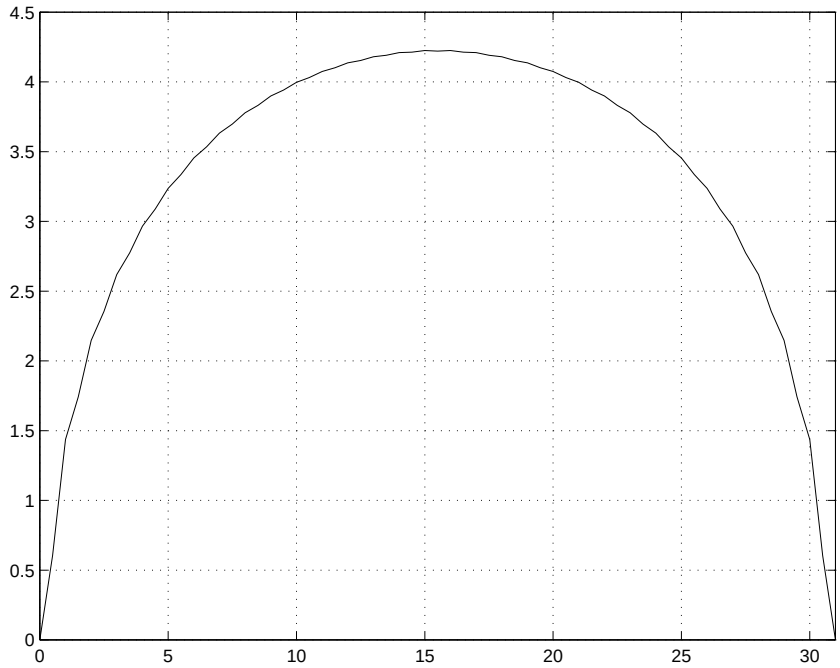


Figure 2.8: Worst-case redundancy ρ of the optimal coding distribution for $\mathcal{P}_{N,\underline{f},c}$ as a function of the average c for $N = 32$.

2.6 Coding Techniques

Let us assume that we want to use arithmetic coding to encode the output of an unknown source for which we are only told that its alphabet size is $N = 32$. We can

- encode the output of the source using a block code of 5 binary digits per source symbol, i.e., the optimal coding distribution for the polytope $\mathcal{P}_{32}$. The expected redundancy will be at most $\log_2 32 = 5$.
- or estimate one parameter of the source distribution, namely its average c and use the optimal coding distribution for the polytope $\mathcal{P}_{32,\underline{f},c}$ in an arithmetic encoder. For every value of the average c, the expected redundancy is upper bounded by the corresponding value of the curve in Figure 2.8, whose maximum over c is approximately $\rho \leq 4.2$ for $c = 15.5$.

If we know that the probability distribution of the source is monotone non-increasing, we can

- encode the source based on the optimal distribution for universal coding over the polytope $\mathcal{M}_{32}$. The expected redundancy ρ will be at most 1.1815.
- or estimate the average c. For every value of c, the redundancy will be upper-bounded by the corresponding value of the curve in Figure 2.7, whose maximum over c is approximately $\rho \leq .646$ for $c = 3$.

The two options stated are based on the principles of universal coding which we developed in this chapter. We will use this approach in a practical coding algorithm in the next chapter, where we will present a device which has the property that it transforms the output of any discrete memoryless source into a source with a monotone non-increasing probability distribution.

In practice, we will encode sequences of bytes, i.e., symbols taken from an alphabet of size $N = 256$. Most of our examples so far were for an alphabet size of $N = 32$ because this was the largest alphabet size for which we could compare the results obtained with the iterated Arimoto-Blahut algorithm to the results of the conventional Arimoto-Blahut algorithm. We can now conclude this chapter with the parameter values that will be significant for our practical implementation:

- the block code of block length 8 which corresponds to the optimal coding distribution for $\mathcal{P}_{256}$ has a worst-case redundancy of **8** binary

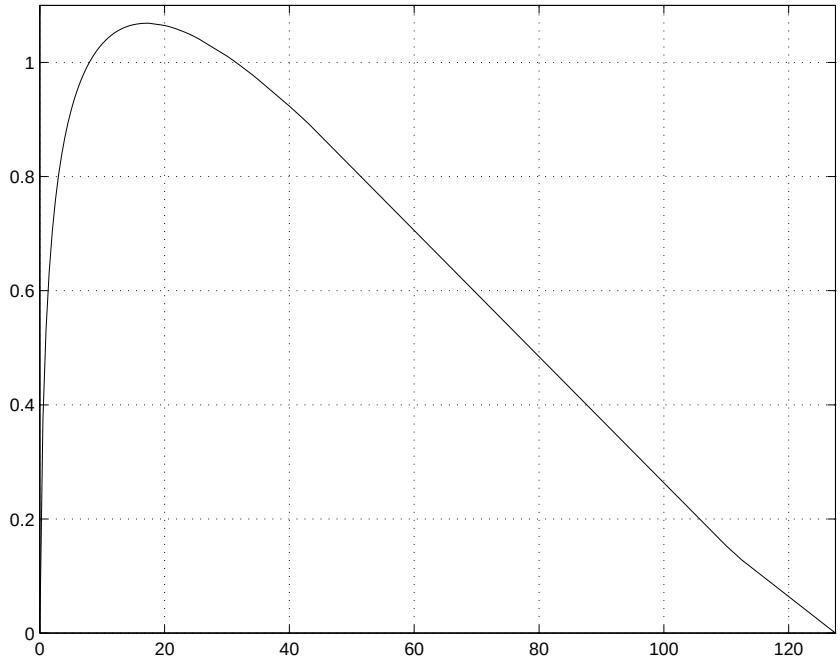


Figure 2.9: Worst-case redundancy ρ of the optimal coding distribution for $\mathcal{M}_{N,\underline{f},c}$ as a function of the average c for $N = 256$.

digits. In the practical coding examples of the next chapter, the inputs files that we wish to encode are already encoded using this trivial block code. Therefore, the usefulness of the universal coding distributions which we computed can be assessed by comparing them to the worst-case redundancy of the block code.

- the worst-case redundancy ρ for arithmetic coding over $\mathcal{M}_{256}$ when using the corresponding optimal coding distribution is $\boldsymbol{\rho = 1.60}$. This corresponds to a gain of 6.4 binary digits when compared to the block code of length 8.

- the worst-case redundancy ρ for optimal universal arithmetic coding over $\mathcal{M}_{N,\underline{f},c}$ when $N = 256$ is plotted in Figure 2.9 as a function of the average c. The worst-case redundancy varies as a function of c between $\boldsymbol{\rho = 0}$ for $c = 0$ and $c = 127.5$ and $\boldsymbol{\rho = 1.07}$ for $c = 16.5$. This corresponds to a gain of between 6.93 and 8 binary digits compared to the block code of length 8.

We will not implement universal arithmetic coding over the polytope $\mathcal{P}_{N,\underline{f},c}$ for $N = 256$.

Appendix: Proof [7] of Lemma 2.3

We begin by repeating the statement of Lemma 2.3.

Lemma 2.3 *Let* $\mathbf{A}$ *be an* $N \times M$ *matrix and* $\underline{b}$ *be an* $N \times 1$ *vector. Then one and only one of the following two propositions is true.*

$$\exists \underline{x} \in \mathbb{R}^N \quad \textit{such that} \quad \mathbf{A}^T \underline{x} \geq^* \underline{0}_M \textit{ and } \underline{b}^T \cdot \underline{x} = 0 \tag{2.19}$$

$$\exists \underline{\mu} \in \mathbb{R}^M \quad \textit{such that} \quad \underline{\mu} > \underline{0}_M \textit{ and } \begin{cases} \mathbf{A}\underline{\mu} = \underline{b}, \textit{ or} \\ \mathbf{A}\underline{\mu} = -\underline{b}, \textit{ or} \\ \mathbf{A}\underline{\mu} = \underline{0}_N. \end{cases} \tag{2.20}$$

where the notation $\geq^*$ *signifies that at least one of the component inequalities is strict.*

Proof: The proof of this lemma will encompass two parts: first we will prove that (2.20) $\Longrightarrow$ $\neg$(2.19). In a second part, we will prove that $\neg$(2.20) $\Longrightarrow$ (2.19).

Assume first that (2.20) is true and that there exists a vector $\underline{\mu} > \underline{0}$ such that $\mathbf{A}\underline{\mu} = \underline{b}$ or $\mathbf{A}\underline{\mu} = -\underline{b}$. Then, for any vector $\underline{x} \in \mathbb{R}^N$ for which $\mathbf{A}^T\underline{x} \geq^* \underline{0}$, we can write

$$\begin{aligned} \underline{x}^T \cdot \underline{b} &= \underline{x}^T \cdot (\pm \mathbf{A}\underline{\mu}) \\ &= \pm(\mathbf{A}^T\underline{x})^T\underline{\mu}. \end{aligned}$$

The last expression cannot be zero if $\underline{\mu} > 0$ and $\mathbf{A}^T\underline{x} \geq^* 0$. Therefore, $(\mathbf{A}^T\underline{x} \geq^* \underline{0}) \implies (\underline{x}^T \cdot \underline{b} \neq 0)$ in this case.

Now assume that there exists a vector $\underline{\mu} > \underline{0}$ such that $\mathbf{A}\underline{\mu} = \underline{0}$. Then, for any vector $\underline{x} \in \mathbb{R}^N$, we can write

$$\underline{0}_N^T \cdot \underline{x} = 0 = (\mathbf{A}\underline{\mu})^T \cdot \underline{x} = \underline{\mu}^T \cdot (\mathbf{A}^T\underline{x}).$$

Since $\underline{\mu} > \underline{0}$, this equation cannot hold if $\mathbf{A}^T\underline{x} \geq^* \underline{0}$. Therefore, $\nexists \underline{x} \in \mathbb{R}^N$ such that $\mathbf{A}^T\underline{x} \geq^* \underline{0}$ in this case. This completes the first part of the proof.

In the second part, we assume that there is no vector $\underline{\mu} \in \mathbb{R}^M$ such that $\underline{\mu} > \underline{0}$ and $\mathbf{A}\underline{\mu} = \underline{b}$, $\mathbf{A}\underline{\mu} = -\underline{b}$ or $\mathbf{A}\underline{\mu} = \underline{0}$.

As reported in [33, p. 34], there is a theorem by Tucker which states that, for the matrices $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$, exactly one of the following statements must hold

[7] Many thanks are due to my officemate Zsolt Kukorelly for completing this proof. It is only fitting that the proof of a variant of Farkas' famous Lemma should be obtained with the help of another valiant Hungarian.

1. $\exists \underline{x}$ such that $\mathbf{B}\underline{x} \geq^* \underline{0}$, $\mathbf{C}\underline{x} \geq \underline{0}$ and $\mathbf{D}\underline{x} = \underline{0}$.
2. $\exists \underline{\mu}_2 > 0$, $\underline{\mu}_3 \geq 0$ and $\underline{\mu}_4$ such that $\mathbf{B}^T\underline{\mu}_2 + \mathbf{C}^T\underline{\mu}_3 + \mathbf{D}^T\underline{\mu}_4 = \underline{0}$.

If we choose $\mathbf{B} = \mathbf{A}^T$, and $\mathbf{C} = -\underline{b}^T$ and $\mathbf{D} = 0$, $\underline{\mu}_4$ becomes irrelevant, μ_3 becomes a scalar, $\underline{\mu}_2$ is a vector in $\mathbb{R}^M$ and $\underline{x}$ is a vector in $\mathbb{R}^N$. Tucker's theorem now states that exactly one of the following statements must hold

1. $\exists \underline{x} \in \mathbb{R}^N$ such that $\mathbf{A}^T\underline{x} \geq^* \underline{0}_N$ and $\underline{b}^T \cdot \underline{x} \leq 0$.
2. $\exists \underline{\mu}_2 \in \mathbb{R}^M$ and $\mu_3 \in \mathbb{R}$ such that $\underline{\mu}_2 > \underline{0}_M$, $\mu_3 \geq 0$ and $\mathbf{A}\underline{\mu}_2 = \mu_3 \underline{b}$.

The second statement is equivalent to the following statement:

$$\exists \underline{\mu} \in \mathbb{R}^M \text{ such that } \underline{\mu} > \underline{0}_M \text{ and } \begin{cases} \mathbf{A}\underline{\mu} = \underline{b} \text{ or} \\ \mathbf{A}\underline{\mu} = \underline{0}_M \end{cases} \tag{2.37}$$

where $\underline{\mu} = \underline{\mu}_2/\mu_3$ if $\mu_3 > 0$ and $\underline{\mu} = \underline{\mu}_2$ if $\mu_3 = 0$.

In our assumption, (2.37) is false both for $\underline{b}$ and for $-\underline{b}$. Therefore, by Tucker's theorem, there must be a vector $\underline{x}_1 \in \mathbb{R}^N$ such that

$$\mathbf{A}^T\underline{x}_1 \geq^* \underline{0}_N \text{ and } \underline{b}^T \cdot \underline{x}_1 \leq 0$$

and a vector $\underline{x}_2 \in \mathbb{R}^N$ such that

$$\mathbf{A}^T\underline{x}_2 \geq^* \underline{0}_N \text{ and } -\underline{b}^T \cdot \underline{x}_2 \leq 0.$$

Since $\underline{b}^T \cdot \underline{x}_1 \geq 0$ and $\underline{b}^T \cdot \underline{x}_2 \leq 0$, there is a $\lambda \in [0, 1]$ such that

$$\lambda(\underline{b}^T \cdot \underline{x}_1) + (1 - \lambda)(\underline{b}^T \cdot \underline{x}_2) = 0.$$

Define the vector $\underline{x} = \lambda\underline{x}_1 + (1 - \lambda)\underline{x}_2$. Because of the linearity of the scalar product,

$$\underline{b}^T \cdot \underline{x} = \lambda\underline{b}^T \cdot \underline{x}_1 + (1 - \lambda)\underline{b}^T \cdot \underline{x}_2 = 0.$$

Furthermore,

$$\mathbf{A}^T\underline{x} = \lambda\mathbf{A}^T\underline{x}_1 + (1 - \lambda)\mathbf{A}^T\underline{x}_2 \geq^* \underline{0}_N.$$

Therefore, $\underline{x}$ satisfies the requirements of (2.19). This completes the proof.

□

Chapter 3

Coding by Source Transformation

The previous chapter showed how to encode the output of a source when we are told that its marginal probability distribution lies in a given polytope. In this chapter, we are aiming to construct a practical source coding system based on the results of the previous chapter. An urgent question arises: *who* tells us that the marginal distribution of the source we are about to encode lies within a given polytope? Do we wait for a friendly genie to pop up and volunteer this essential information? Or are the results of the previous section doomed to be of no use in practical systems, a pure intellectual exercise with no applications in the real world? Once again, we find the answer to this question in Elias' pioneering work. Not content with having devised universal representations of the integers, Elias went on to ponder how one could use those representations to encode the output of any discrete stationary source. In his paper [4], he shows that by encoding the interval since the last occurrence of the current symbol rather than encoding the current symbol itself, the codes he presented in his earlier paper [3] could be used to achieve an upper bound which tends towards the entropy of any discrete stationary source as the alphabet size tends towards infinity. An even better performance can be achieved by encoding what he calls the *recency-rank* of the current symbol. Effectively, he had devised a universal coding method which is asymptotically optimal for the class of discrete stationary sources. By the time of his second paper [4], Elias was not alone in considering this type of encoding. In [23], Willems presented a version of interval coding in which intervals are encoded when their value is less than the alphabet size, otherwise the sym-

bol itself is encoded using a fixed prefix followed by a block code. Willems uses a simpler alternative to Elias' representations of the integers to encode the resulting intervals. Ryabko introduced a coding method which is related to recency-ranking in [20], as did Bentley et al. in [6].

In this chapter, we will start by generalizing Elias' approach of using universal coding on a larger class of sources than the class it was designed for. The concept of "source transformation" will be introduced. Two devices for source transformation will be considered and analyzed: Elias' recency-rank calculator, and a closely related device which we shall call the "competitive list transformer". While these devices will be studied under the assumption that they are applied to the output of a discrete memoryless source, in practice we plan to encode the output of discrete stationary sources which are not necessarily memoryless. The concept of conditional coding will be introduced, or coding based on a context tree, to transform the output of some discrete stationary sources into an array of essentially memoryless sources. A universal source coding algorithm using conditional coding in conjunction with competitive list transformers will be introduced. Finally, we will present the compression results achieved by this algorithm when used to encode a standard collection of files.

3.1 Source Estimation and Source Transformation

In the previous chapter, we defined the concept of universal coding as describing the use of one encoder to encode the output of any source in a given class of sources. In practice, the sources to be encoded will seldom be restricted to a class for which there is a good universal encoder. One way to get around this problem is to map the large class of sources that we wish to encode into a class for which there is a good universal encoder. This can be done through an operation we will call "source transformation". In order to describe the role of source transformation, we will start by clarifying the situation when encoding the output of a known source and when encoding the output of an unknown source using adaptive coding.

When the source to be encoded is known, the coding situation is illustrated in Figure 3.1. The encoder is applied directly to the output sequence of the source, and the decoder generates a replica of the source sequence. Both the encoder and the decoder depend on the parameters of the source. In other words, they have been designed to encode and decode the output of this particular source.

If the parameters of the source to be encoded are not known, adaptive

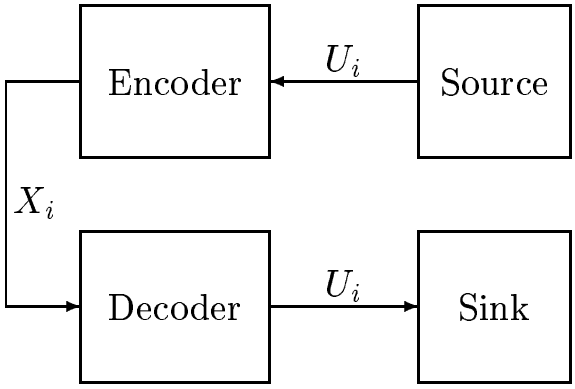


Figure 3.1: Coding situation when encoding the output of a known source.

coding can be used. In adaptive coding, all the parameters of the source are estimated on-line while the output of the source is being encoded. This situation is illustrated in Figure 3.2. Once again, the encoder is

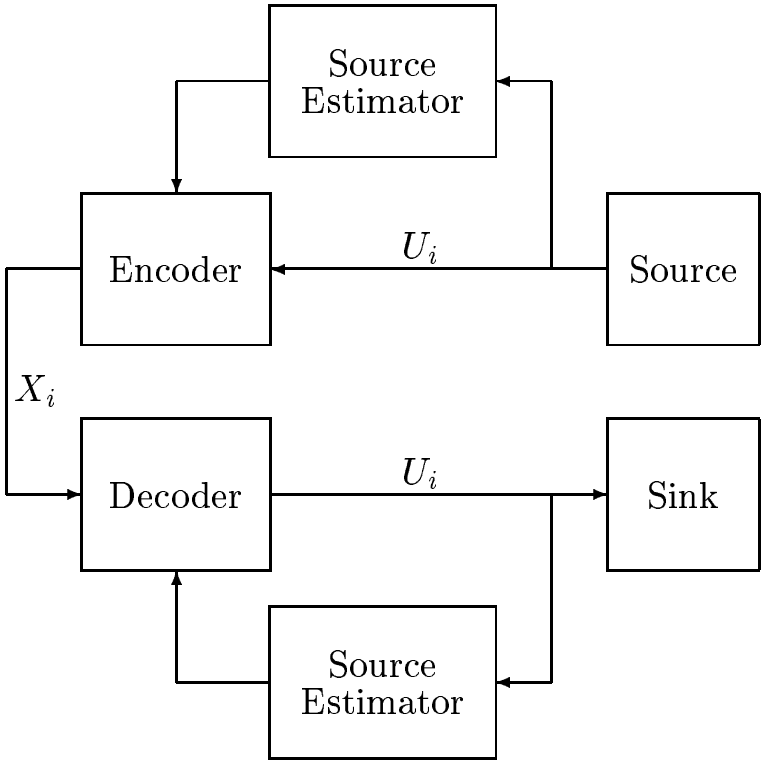


Figure 3.2: Adaptive coding.

applied directly to the output sequence of the source and the decoder generates a replica of the source sequence. The source estimator estimates the parameters of the source based on its observation of the source output sequence, and the source encoder is modified after each encoded symbol to reflect the updated source parameters. The condition for decodability

is that the estimation for the source parameters in the encoder and in the decoder coincide at every coding step. This is ensured by allowing the estimated parameters of the source for the k-th encoding operation to depend only on the observation of the source output sequence up to its $(k-1)$-th element. Since the decoder has generated a replica of the source sequence up to and including its $(k-1)$-th element before performing the k-th decoding operation, it can replicate the source parameters estimated by the encoder.

We are now ready to discuss the lesson we learned from Elias' approach of using universal coding on another class than the class of sources the encoder was designed for. The coding situation for this approach is illustrated in Figure 3.3. This time, the encoder is not applied directly to

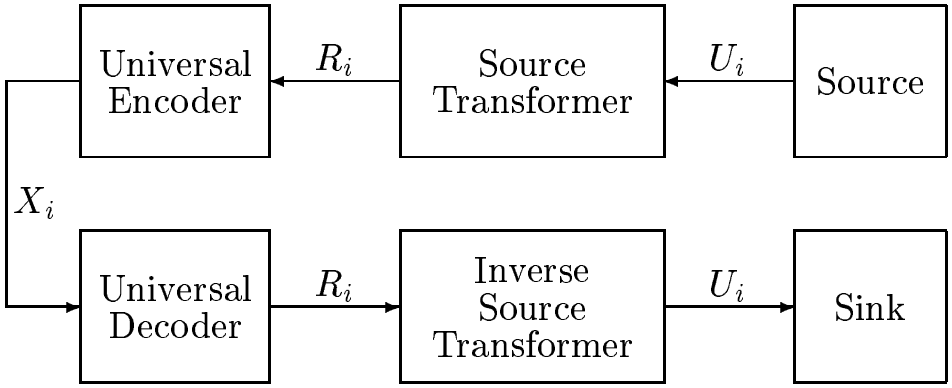


Figure 3.3: Universal coding with source transformation.

the output sequence of the source. Instead, the encoder is fed with the output sequence of a source transformer. From the point of view of the encoder, the source is now "transformed", i.e., the source consists of the block including the real source and the transformer. The role of the source transformer is to perform an invertible mapping from the possible output sequences of the source to sequences for which universal coding achieves a good performance. In other words,

> if the sources we wish to encode belong to a class for which there is no good universal encoder, we can seek an invertible source transformation which maps the sources in the class into sources in a new class for which there exists a good universal encoder.

There are several examples of invertible source transformations that can be used for this type of encoding.

Example: In the method we described earlier as "interval coding", the transformer replaces the current output symbol of the source with the dis-

crete time interval that elapsed since the last occurrence of the current symbol. For example, if the past output sequence of a source with alphabet $\{A, B, C, D\}$ ended with the sequence "$\ldots, C, D, A, A, B, C, A$" and the next symbol to be processed was a 'D', the transformer would forward an interval value of 6 to the encoder, because the symbol D occurred 6 symbols earlier in the source output. Supposing that the source to be encoded is a discrete *memoryless* source whose alphabet is $\mathcal{A} = \{u_1, u_2, \ldots, u_N\}$, the marginal probability distribution of the intervals R at the output of the source transformer can be expressed for $r = 1, 2, 3, \ldots$ as

$$\begin{aligned} P_R(r) &= \sum_{i=1}^{N} P_U(u_i) P_{R|U}(r|u_i) \\ &= \sum_{i=1}^{K} [P_U(u_i)]^2 (1 - P_U(u_i))^{r-1}. \end{aligned}$$

It is easy to see that this probability distribution is a monotone non-increasing function of r. Therefore, the source transformer in interval coding maps all sources in the class of discrete memoryless sources into a class of sources whose marginal probability distributions are monotone non-increasing. It does so at the price of an alphabet expansion: while the sources in this example have an alphabet size of N, the interval since the last occurrence of a symbol can take on any value between 1 and $+\infty$. Willems' version of interval coding and Elias' recency-ranking circumvent the problem of the alphabet expansion. We will describe recency-ranking and a related method in the next section. Elias' universal representations of the integers can be used to encode the intervals generated by the source transformer in our example.

It should be pointed out that Elias' derivation of interval coding makes no use of the fact that the probability distribution of the intervals is monotone non-increasing. Elias uses the expected value of the interval given the current symbol to state an upper bound on the performance of his universal representations when used to encode intervals. As a result, Elias' upper bound applies to all discrete stationary ergodic sources and not just to memoryless sources.

There are not many examples in the real world where we know in advance that a source we wish to encode belongs to a class of sources for which there is a good universal encoder. The method of source transformation gives us the chance to apply universal coding to sources in wider classes, like the class of all discrete memoryless sources or the class of all discrete stationary sources. The source transformer is the genie who pops

up and volunteers the information without which we could have confined the optimal coding distributions of the previous chapter into the realm of entertaining but useless theories.

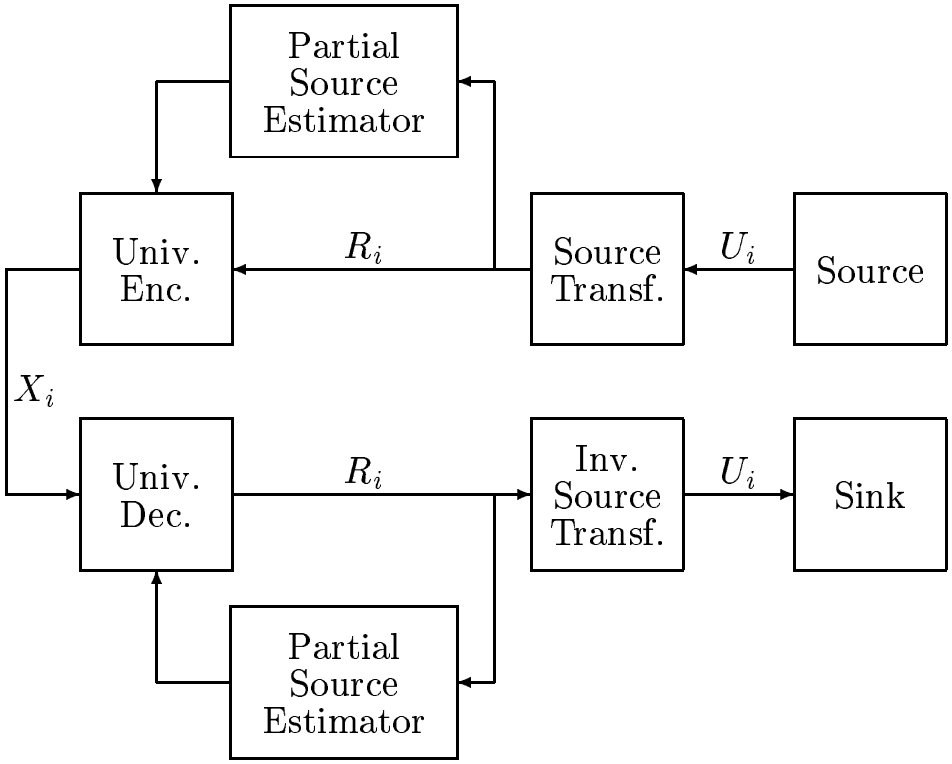


Figure 3.4: Universal coding with source transformers and partial source estimators.

We can combine the technique of source transformation with a partial source estimation. This situation is illustrated in Figure 3.4. Contrary to the source estimation performed in adaptive coding, a partial source estimator does not estimate all the parameters of the source. Its aim is to estimate one or a few parameters which will allow the use of a universal encoder designed for a more restricted class of sources, yielding a better performance.

Example: Suppose that the output of the source transformer is known to have a marginal probability distribution which lies in the set $\mathcal{M}_N$ of all monotone non-increasing distributions of alphabet size N. The partial source estimator can be used to estimate the average $c = E[f(R_i)]$ of the output sequence. Using this information, the optimal coding distribution for the set $\mathcal{M}_{N,\underline{f},c}$ of all monotone non-increasing distributions whose average $E[f(.)]$ is equal to c can be used to achieve a better worst-case redundancy than is achievable for the set $\mathcal{M}_N$.

In the next section, we discuss recency-ranking and introduce the competitive list transformer. Our aim is to use these methods as source transformers for universal coding. We show that the marginal distribution of the output sequence of either transformer is a monotone non-increasing distribution whenever the device is applied to the output sequence of a discrete memoryless source. This will allow us to use the results obtained in the previous chapter regarding optimal universal coding for sources with a monotone non-increasing probability distribution.

3.2 Recency Ranking and Competitive Lists

We described Elias' method of coding the discrete interval since the last occurrence of the current source output symbol. In this scheme, the interval encoded is the number of symbols emitted by the source since the source last emitted its current output symbol plus one. The recency-rank is defined in a very similar way: it is the number of *distinct* symbols emitted by the source since the last occurrence of its current output symbol plus one. In other words, symbols that have appeared more than once in the source output in the meantime are counted only once when determining the recency-rank of the current symbol, whereas they are counted as many times as they have occurred when determining the interval of the current symbol.

Example: Consider a source with alphabet $\{A, B, C, D\}$ whose output ended with the sequence "$\ldots, C, D, A, A, B, C, A$". This situation was treated in a previous example to illustrate interval coding. When the next symbol to be processed is a 'D', we found the corresponding interval to be 6 because the source emitted the five symbols A, A, B, C, A since it last emitted a 'D'. The recency-rank of 'D' in this case is 4 because the three symbols $\{A, C, B\}$ occurred in the source output since the last occurrence of the symbol 'D'. The fact that 'A' occurred three times in the meantime makes no difference to the recency-rank.

By definition, the recency-rank is always at most equal to the corresponding interval. Therefore, Elias' upper bound on universal coding for the intervals which, as mentioned earlier, is based on the expected value of the interval given the current output symbol must hold for the recency-rank as well. Furthermore, since the recency-rank counts only distinct symbols in the past output of the source, it is at most equal to the size of the source alphabet. Therefore, the recency-rank does away with the alphabet expansion which is a drawback of interval coding.

The recency-rank can be computed on-line by a device we call the "recency-rank calculator". The recency-rank calculator transforms a sequence of symbols from a source with an alphabet of size N into a sequence of integers between 1 and N. The device holds an ordered list of the symbols of the source. At every step, the device seeks the current output symbol of the source in its list, outputs the position r of that symbol in the list, and updates the list by moving the current symbol to the top of the list. This explains the term "move-to-front" which Ryabko in [20] and Bentley & al. in [6] use to describe this coding method. In this operation, the symbols which were located at positions 1 to $r-1$ are all shifted down one position in the list. If the source output symbol is already at the top position in the list, the device outputs a 1 and leaves the list unchanged. The operation of the recency-rank calculator is shown for a 5-ary alphabet $\{A, B, C, D, E\}$ in Figure 3.5. When the source output symbol is found at

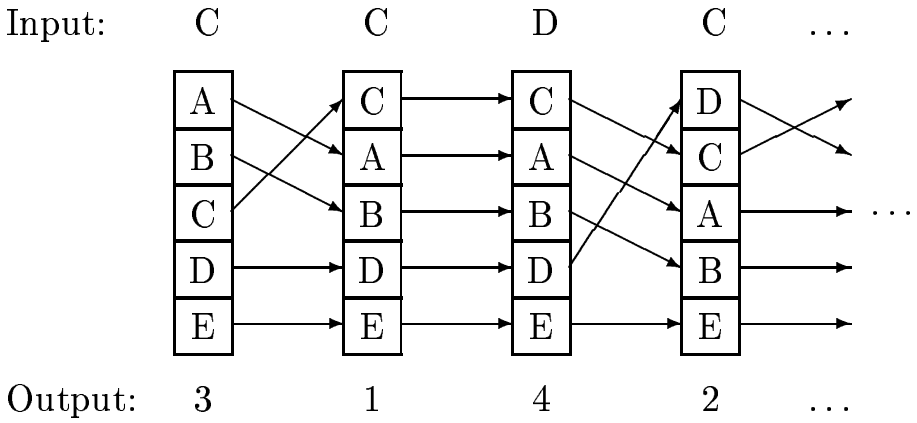


Figure 3.5: The Recency-Rank Calculator "in action"

position r, the symbols at positions 1 to $r-1$ must have occurred at least once in the source output sequence since the last occurrence of the current symbol, since this is the only way those symbols could have jumped to a position above the current symbol in the list. Therefore, the output of the recency-rank calculator is indeed the recency-rank defined above.

The inverse recency-rank calculator holds a list which is always identical to the list in the recency-rank calculator. Whenever the inverse device receives a recency-rank, it looks up the corresponding symbol in the list and updates the list by moving this symbol to the front of the list. If both devices are started in the same state, the inverse device will always be able to recover the source sequence from the sequence of recency-ranks. Therefore, the recency-rank calculator and its inverse satisfy the requirements we set for devices to be used as a source transformer and an inverse source transformer in a universal coding scheme.

The competitive list transformer is a device very similar to the recency-rank calculator. It also holds a list of the source alphabet. Whenever a symbol is received from the source, it locates that symbol in the list and outputs its position r. Instead of moving the symbol to the top of the list as would a recency-rank calculator, it exchanges the symbol with the symbol preceding it in the list. When the symbol received is already at the top of the list, the competitive list transformer outputs a 1 and leaves the list unchanged. The operation of the competitive list transformer is shown for the alphabet $\{A, B, C, D, E\}$ in Figure 3.6. The competitive list

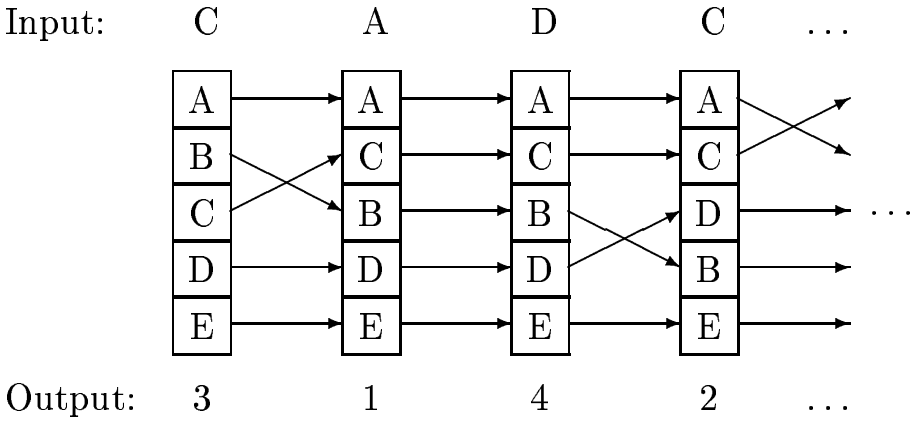


Figure 3.6: The Competitive List Transformer "in action"

transformer is a non-expanding transformer, since it will output integers between 1 and N when it is applied to the output sequence of a source whose alphabet size is N. Upon receiving a rank, the inverse competitive list transformer retrieves the corresponding symbol in the list and proceeds to update the list as in the transformer. If the initial state of the inverse transformer is chosen to be identical to the initial state of the transformer, the source sequence can be recovered from the sequence of ranks. Therefore, this device also fulfills the requirements for a source transformer to be used for universal coding.

The name "competitive list" was chosen to reflect the process where symbols are competing to rise in the list and symbols which occur in the output of the source receive a bonus towards that end. The competitive list was first studied by Rivest in [19] as a method to order the entries in a database so as to minimize the time required to retrieve an entry. Rivest calls the competitive list a "self-organizing list using the transposition heuristic". We prefer the name "competitive list" because it is shorter and it evokes a better image of how the list functions.

While it has been shown that both the competitive list transformer and the recency-rank calculator are valid source transformers in the sense

defined previously, no thought has been given to the outcome of the transformation realized by those two devices. According to the approach described in the previous section, we plan to encode the output of a source transformer using a universal encoder. Which type of universal encoder can be used to encode the output of a competitive list transformer or of a recency-rank calculator? We will soon give an analytical answer to this question for both transformers. Before we proceed, it is worth giving an intuitive description of the effect we expect the recency-rank calculator and the competitive list transformer to have on the output distribution of a source.

In both devices, symbols are allowed to rise in the list only when they occur in the output sequence of the source. Therefore, once the influence of the initial ordering of the list has faded away, symbols that occur frequently will be found more often near the top of the list, while symbols that occur infrequently will in general be found near the bottom of the list. This effect is stronger in the competitive list, where the occurrence of an infrequent symbol has a minor effect on the ordering of the list, while such an event causes a stronger disturbance in the recency-rank calculator. On the other hand, a recency-rank calculator will reach its steady state sooner, i.e., the influence of the initial ordering of the list will fade away sooner than it will for the competitive list transformer. Since frequent symbols will mostly be found near the top of the list, we can also expect small ranks to be found more frequently than larger ranks in the output of those source transformers. Conceptually, we expect a decreasing marginal distribution at the output of the source transformers. The output distribution of the competitive list transformer is expected to be "more decreasing" than the output distribution of the recency-rank calculator.

We will show that the output distributions of a competitive list transformer and of a recency-rank calculator which have reached their steady state are monotone non-increasing when those transformers are applied to the output of a discrete memoryless source. Though it is not clear what "more decreasing" means, we will quantify the difference between the two transformers in the results of a simulation.

In practice, we are mostly interested in encoding the output of a discrete stationary source which is not necessarily memoryless. The results we present apply only to memoryless sources. In the next section, we discuss an approach which can convert some stationary sources into an array of essentially memoryless sources. Together with the results we are about to present, this serves as a justification for the practical source coding algorithm presented in the next section.

Steady-State Analysis

We now consider the situation when the competitive list transformer is applied to the output of a discrete memoryless source with an alphabet $\mathcal{A}$ of size N. This situation is illustrated in Figure 3.7. The competitive list

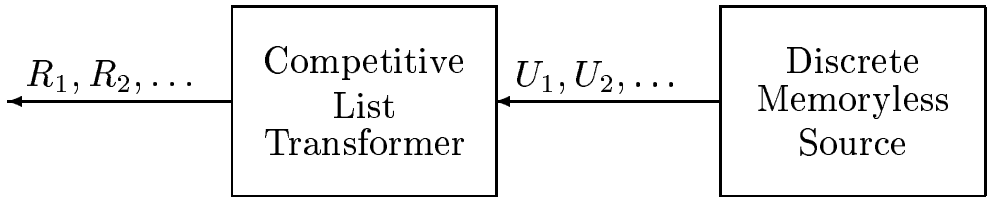


Figure 3.7: The situation for the steady-state analysis.

transformer has $N!$ possible states corresponding to all possible orderings of the alphabet $\mathcal{A}$. We write $\{s_1, s_2, \ldots, s_{N!}\}$ for the set of possible states and S for the random variable whose value is the state of the list.

Steady-State Analysis for the 2 and 3-Symbol Alphabets

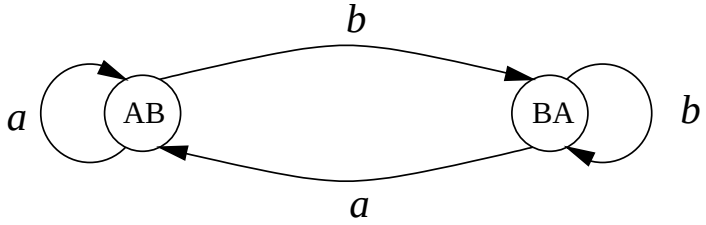


Figure 3.8: The state-diagram of a 2-symbol competitive list transformer

We first study the behavior of the competitive list transformer for a binary memoryless source with an alphabet $\mathcal{A} = \{A, B\}$ and with probabilities a and b respectively. This list has two possible states, $s_1 =$ "AB" and $s_2 =$ "BA". The probabilities of being in these two states at any time can be written as a vector $P_S = [P_S(s_1), P_S(s_2)]^T$. The state-transition diagram of this system is shown in Figure 3.8.

The transition matrix of the competitive list transformer

$$\mathbf{T} = \begin{bmatrix} a & a \\ b & b \end{bmatrix}$$

maps the current state probability vector P_S into the state probability vector at the next step. The steady-state probability vector satisfies the

relation

$$P_S = \mathbf{T} P_S$$

and the additional constraint that the sum of the the state probabilities must be 1. Together, those two relations yield the solution

$$P_S = [a, b]^T.$$

The steady-state probabilities of the output ranks can then be computed as

$$P_R(r) = P_{R|S}(r|s_1)P_S(s_1) + P_{R|S}(r|s_2)P_S(s_2)$$

for $r = 1 \ldots 2$. But $P_{R|S}(r|s_i)$ is precisely the probability of the symbol at the r-th position in the list corresponding to state s_i. This gives us the steady-state rank distribution

$$P_R = [P_R(1), P_R(2)]^T = [a^2 + b^2, 2ab]^T.$$

It is easy to verify that $P_R(1) \geq P_R(2)$ for any choice of the probabilities a and b, which in this simple 2-symbol case confirms our claim that the output distribution of the competitive list is monotone non-increasing.

The state-transition diagram of the competitive list transformer for a 3-symbol alphabet $\mathcal{A} = \{A, B, C\}$ with probabilities a, b and c is given in Figure 3.9. The transition matrix is

$$\mathbf{T} = \begin{bmatrix} a & a & 0 & 0 & 0 & b \\ b & b & a & 0 & 0 & 0 \\ 0 & c & b & b & 0 & 0 \\ 0 & 0 & c & c & b & 0 \\ 0 & 0 & 0 & a & c & c \\ c & 0 & 0 & 0 & a & a \end{bmatrix}.$$

Though this matrix is relatively sparse and has a clear symmetric pattern, it is almost impossible to find the solution of the equation $P_S = \mathbf{T} P_S$ either by hand or with one of the powerful symbolic software tools available. With a little luck and by analyzing the numerical solutions for a few examples, we found the steady-state probability vector to be

$$P_S = \begin{bmatrix} P_S(\text{“}ABC'') \\ P_S(\text{“}BAC'') \\ P_S(\text{“}BCA'') \\ P_S(\text{“}CBA'') \\ P_S(\text{“}CAB'') \\ P_S(\text{“}ACB'') \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} a^2 b \\ b^2 a \\ b^2 c \\ c^2 b \\ c^2 a \\ a^2 c \end{bmatrix}$$

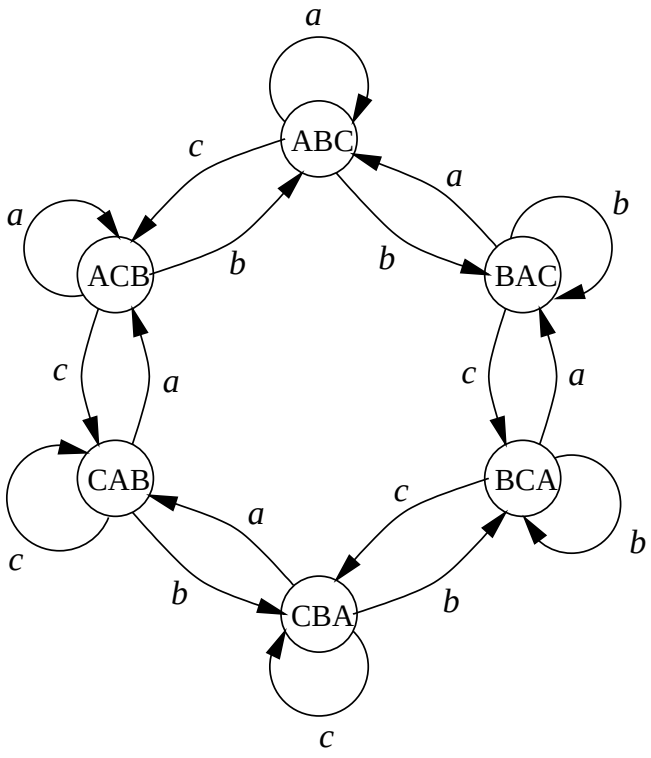


Figure 3.9: The state-diagram of a 3-symbol competitive list transformer

where $\Delta = a^2(b+c)+b^2(a+c)+c^2(a+b)$. The output probabilities can be computed as in the 2-symbol case, giving a complicated expression which we will spare the reader.

It appears to be very difficult for even moderately large N to compute the steady-state probability vector directly from the equation $P_S = \mathbf{T}P_S$ because the transition matrix $\mathbf{T}$ is an $N! \times N!$ matrix. Luckily, we can guess the general solution based on the two examples just derived, thereby avoiding the troublesome burden of solving such huge systems of equations explicitly.

Steady State Solutions for N-ary Alphabets

The general solution is given in the following proposition, which was stated by Rivest in [19] in a slightly different form[1].

Proposition 3.1 *The steady-state probabilities of a competitive list applied to the output of a discrete memoryless source with alphabet* $\mathcal{A} =$

[1]Unaware of Rivest's solution, I sought the invaluable help of my colleague Thomas Ernst with whom we were able to discover this solution independently, albeit 20 years after Rivest's original solution. Nonetheless, I cannot resist the temptation of thanking yet another friend for the exciting collaboration which has led to this belated proposition.

$\{u_1, u_2, \ldots, u_N\}$ and corresponding probabilities $\alpha_1, \alpha_2, \ldots, \alpha_N$ are given as

$$P_S(u_{i_1} u_{i_2} \ldots u_{i_N}) = \frac{1}{\Delta} \alpha_{i_1}^{N-1} \alpha_{i_2}^{N-2} \cdots \alpha_{i_{N-1}}^{1} \alpha_{i_N}^{0} \tag{3.1}$$

for any $(i_1, i_2, \ldots, i_N) \in S_N$, where S_N is the set of all permutations of the numbers $1, 2, \ldots N$. The normalization factor Δ is

$$\Delta = \sum_{(i_1, i_2, \ldots, i_N) \in S_N} \alpha_{i_1}^{N-1} \alpha_{i_2}^{N-2} \cdots \alpha_{i_{N-1}}^{1} \alpha_{i_N}^{0}. \tag{3.2}$$

Proof: Transitions to a given state $s = u_{i_1} u_{i_2} \ldots u_{i_N}$ of an N-ary competitive list are possible only from state s itself, or from states s' in which two adjacent symbols of the list are permuted, i.e., there must be a k such that $s' = u_{i_1} \ldots u_{i_{k+1}} u_{i_k} \ldots u_{i_N}$. These correspond to the non-zero elements in the row pointing to s in the transition matrix. There are N non-zero elements, one pointing from s itself, and $N-1$ pointing from the states in which two adjacent list elements are permuted. The steady-state constraint corresponding to state s is thus

$$\begin{aligned} P_S(s) &= \alpha_{i_1} P_S(s) + \alpha_{i_1} P_S(u_{i_2} u_{i_1} \ldots u_{i_N}) \\ &\quad + \cdots + \alpha_{i_k} P_S(u_{i_1} \ldots u_{i_{k+1}} u_{i_k} \ldots u_{i_N}) \\ &\quad + \cdots + \alpha_{i_{N-1}} P_S(u_{i_1} \ldots u_{i_N} u_{i_{N-1}}). \end{aligned}$$

Inserting the solution stated in the proposition, we obtain

$$\begin{aligned} P_S(s) &= \frac{1}{\Delta} [\alpha_{i_1} \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0} \\ &\quad + \alpha_{i_1} \alpha_{i_2}^{N-1} \alpha_{i_1}^{N-2} \cdots \alpha_{i_N}^{0} \\ &\quad + \cdots + \alpha_{i_k} \alpha_{i_1}^{N-1} \cdots \alpha_{i_{k+1}}^{N-k} \alpha_{i_k}^{N-k-1} \cdots \alpha_{i_N}^{0} \\ &\quad + \cdots + \alpha_{i_{N-1}} \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{1} \alpha_{i_{N-1}}^{0}] \\ &= \frac{1}{\Delta} \alpha_{i_1}^{N-1} \alpha_{i_2}^{N-2} \cdots \alpha_{i_N}^{0} (\alpha_{i_1} + \alpha_{i_2} + \cdots + \alpha_{i_N}) \\ &= P_S(s), \end{aligned}$$

where we have used the fact that

$$\sum_{k=1}^{N} \alpha_{i_k} = \sum_{k=1}^{N} P_U(u_{i_k}) = 1.$$

Thus, the solution stated satisfies the equation $P_S = \mathbf{T}P_S$. The division by Δ ensures that the additional constraint $\sum_s P_S(s) = 1$ is satisfied. The uniqueness of the steady-state probabilities ensures that the solution stated is the only solution. □

As before, it is possible to write the probability distribution of the ranks based on the steady-state distribution of the list. This results in the following proposition:

Proposition 3.2 *The steady-state output probabilities of a competitive list transformer applied to the output sequence of a discrete memoryless source with alphabet* $\mathcal{A} = \{u_1, u_2, \ldots, u_N\}$ *and corresponding probabilities* $\alpha_1, \alpha_2, \ldots, \alpha_N$ *are*

$$\begin{aligned} P_R(r) &= \sum_{k=1}^{N!} P_{R|S}(r|s_k) P_S(s_k) \\ &= \frac{1}{\Delta} \sum_{(i_1,\ldots i_N)\in S_N} \alpha_{i_r} \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0}. \end{aligned} \tag{3.3}$$

for $r = 1 \ldots N$.

Although this is a precise formulation of the output distribution of a competitive list given its input distribution, it does not give much insight into the character of the output distribution. In an appendix to this chapter, we state an alternative formulation which gives more insight into the mathematical structure of the output probabilities derived by relating it to a matrix operator known as a *permanent*.

We are now ready to prove the following theorem:

Theorem 3.1 *When a competitive list transformer is used on the output sequence of a discrete memoryless source with strictly positive probabilities* $\alpha_1, \alpha_2, \ldots, \alpha_N$ *to yield the output sequence* $R_1, R_2, \ldots$, *the steady-state distribution* $P_R(.)$ *of the output ranks* R *satisfies the inequality*

$$\max_i \alpha_i \geq P_R(1) \geq P_R(2) \geq \ldots \geq P_R(N) \geq \min_i \alpha_i \tag{3.4}$$

with equality everywhere only when the input distribution is $\alpha_i = 1/N$ *for* $i = 1, 2, \ldots, N$. *Otherwise all inequalities are strict.*

Proof: We begin by proving the outer inequalities. For $r = 1, 2, \ldots, N$, the sum in (3.3) can be partitioned as follows

$$P_R(r) = w_{r1}\alpha_1 + w_{r2}\alpha_2 + \cdots + w_{rN}\alpha_N,$$

where where $w_{rj} > 0$ for all j and $\sum_{j=1}^{N} w_{rj}$ is the sum of the state probabilities which is equal to one. Thus,

$$P_R(r) \leq \left(\sum_{k=1}^{N} w_{rk} \right) \max_i \alpha_i = \max_i \alpha_i.$$

Similarly,

$$P_R(r) \geq \left(\sum_{k=1}^{N} w_{rk} \right) \min_i \alpha_i = \min_i \alpha_i.$$

We have proved that $\max_i \alpha_i \geq \max_r P_R(r)$ and $\min_i \alpha_i \leq \min_r P_R(r)$. Now, for any r between 1 and $N-1$, we define

$$\xi(r) \stackrel{\text{def}}{=} P_R(r) - P_R(r+1).$$

Using (3.3), we can write

$$\begin{aligned} \xi(r) &= \frac{1}{\Delta}\Big(\sum_{(i_1,\ldots i_N)\in S_N} \alpha_{i_r} \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0} \\ &\quad - \sum_{(i_1,\ldots i_N)\in S_N} \alpha_{i_{r+1}} \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0} \Big) \\ &= \frac{1}{\Delta} \sum_{(i_1,\ldots i_N)\in S_N} (\alpha_{i_r} - \alpha_{i_{r+1}}) \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0}. \end{aligned}$$

We partition the set S_N of all permutations of the N-tuple $(1, 2, \ldots, N)$ into two sets S_N^1 and S_N^2 of equal cardinality in the following way: pick a permutation $(i_1, i_2, \ldots, i_N)$ in S_N at random, and put it in S_N^1. Now select the permutation $(i_1, i_2, \ldots, i_{r+1}, i_r, \ldots, i_N)$ that is the same in every position as the previous permutation except that positions r and $r+1$ are exchanged, and put it in S_N^2. For example, if $N = 5$, $r = 3$ and we picked the permutation $(3, 2, 5, 4, 1)$ to put in S_5^1, we must put $(3, 2, 4, 5, 1)$ in S_N^2. We can separate the sum above into two sums

$$\begin{aligned} \xi(r) &= \frac{1}{\Delta} \sum_{(i_1,\ldots i_N)\in S_N^1} (\alpha_{i_r} - \alpha_{i_{r+1}}) \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0} \\ &\quad + \frac{1}{\Delta} \sum_{(i_1,\ldots i_N)\in S_N^2} (\alpha_{i_r} - \alpha_{i_{r+1}}) \alpha_{i_1}^{N-1} \cdots \alpha_{i_N}^{0}. \end{aligned}$$

Since the permutations in S_N^1 and S_N^2 differ only in the r-th and the $r+1$-th position, we can reduce these two sums to one sum over S_N^1

$$\begin{aligned}
\xi(r) &= \frac{1}{\Delta}\sum_{S_N^1}\Big[(\alpha_{i_r}-\alpha_{i_{r+1}})\alpha_{i_1}^{N-1}\cdots\alpha_{i_N}^{0} \\
&\quad +(\alpha_{i_{r+1}}-\alpha_{i_r})\alpha_{i_1}^{N-1}\cdots\alpha_{i_{r+1}}^{N-r}\alpha_{i_r}^{N-r-1}\cdots\alpha_{i_N}^{0}\Big] \\
&= \frac{1}{\Delta}\sum_{S_N^1}(\alpha_{i_r}-\alpha_{i_{r+1}})(\alpha_{i_1}^{N-1}\cdots\alpha_{i_r}^{N-r}\cdot \\
&\quad \alpha_{i_{r+1}}^{N-r-1}\cdots\alpha_{i_N}^{0}-\alpha_{i_1}^{N-1}\cdots\alpha_{i_{r+1}}^{N-r}\alpha_{i_r}^{N-r-1}\cdots\alpha_{i_N}^{0}) \\
&= \frac{1}{\Delta}\sum_{S_N^1}(\alpha_{i_r}-\alpha_{i_{r+1}})\alpha_{i_1}^{N-1}\cdots\alpha_{i_{r-1}}^{N-r+1}\cdot \\
&\quad \alpha_{i_r}^{N-r-1}\alpha_{i_{r+1}}^{N-r-1}\cdots\alpha_{i_N}^{0}(\alpha_{i_r}-\alpha_{i_{r+1}}).
\end{aligned}$$

We can finally write the following equation

$$\xi(r)=\frac{1}{\Delta}\sum_{S_N^1}\frac{\alpha_{i_1}^{N-1}\cdots\alpha_{i_N}^{0}}{\alpha_{i_r}}(\alpha_{i_r}-\alpha_{i_{r+1}})^2. \tag{3.5}$$

Since all the probabilities are strictly positive, we have proved that $P_R(r)\geq P_R(r+1)$. Equality holds only when all the terms of the sum in (3.5) are zero, which is true only if all the probabilities α_i are equal, i.e., $\alpha_i=1/N$ for $i=1,2,\ldots,N$. □

Theorem 3.1 confirms our claim that the output distribution of a competitive list transformer applied to the output of a discrete memoryless source is monotone non-increasing. This will not necessarily be true when the competitive list transformer is applied to the output of a discrete stationary source which is not memoryless. A notorious counter-example is the source which outputs the semi-infinite sequence $0,1,0,1,0,1,\ldots$ with a probability of $1/2$ and the semi-infinite sequence $1,0,1,0,1,\ldots$ with a probability of $1/2$. This source is stationary, but the competitive list transformer will always output the constant rank 2 with probability 1. However, simulations indicate that the output distribution of a competitive list transformer applied to most practical sources tends to be monotone non-increasing.

Steady-state probabilities of the recency-rank calculator

We will now derive the state probabilities and the output probability distribution of a recency-rank calculator when it is applied to the output of

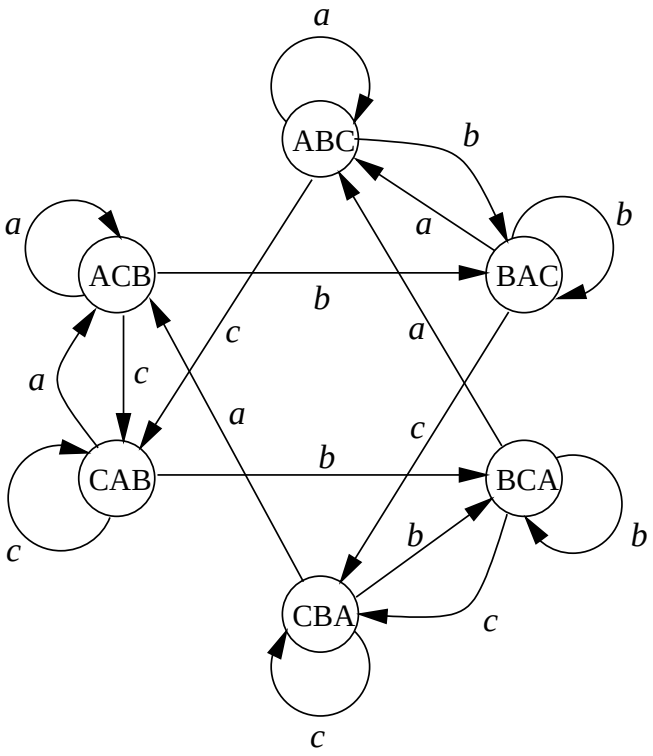


Figure 3.10: The state-diagram of a 3-symbol recency-rank calculator

a discrete memoryless source. For comparison, the state-transition diagram of a recency-rank calculator applied to a discrete memoryless source with alphabet $\mathcal{A} = \{A, B, C\}$ and corresponding probabilities a, b and c is shown in Figure 3.10. The corresponding state transition matrix is

$$\mathbf{T} = \begin{bmatrix} a & a & a & 0 & 0 & 0 \\ b & b & 0 & 0 & 0 & b \\ 0 & 0 & b & b & b & 0 \\ 0 & c & c & c & 0 & 0 \\ c & 0 & 0 & 0 & c & c \\ 0 & 0 & 0 & a & a & a \end{bmatrix}.$$

The steady-state probabilities of the recency-rank calculator are obtained by solving the equation $P_S = \mathbf{T}P_S$ as for the competitive list transformer. Again, the solution of this equation was discovered by observing the numerical solution for a few simple examples.

Proposition 3.3 *The steady-state probabilities of a recency-rank calculator applied to the output of a discrete memoryless source with alphabet* $\mathcal{A} = \{u_1, u_2, \ldots, u_N\}$ *and corresponding probabilities* $\alpha_1, \alpha_2, \ldots, \alpha_N$ *are*

given as

$$P_S(u_{i_1}u_{i_2}\dots u_{i_N}) = \frac{\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_{N-1}}}{(1-\alpha_{i_1})(1-\alpha_{i_1}-\alpha_{i_2})\cdots(1-\alpha_{i_1}-\ldots-\alpha_{i_{N-2}})} \tag{3.6}$$

for any $(i_1, i_2, \dots, i_N) \in S_N$, *where* S_N *is the set of all permutations of the numbers* $1, 2, \dots N$.

Proof: The proof is carried out by verifying that the state probabilities stated sum to one and that they satisfy the equation $P_S = \mathbf{T}P_S$ for the state-transition matrix $\mathbf{T}$ of a recency-rank calculator. We leave out the actual derivation, since it is even more tedious than the derivation of Proposition 3.1. □

As for the competitive list transformer, the output probabilities of the recency-rank calculator are determined by the state probabilities (3.6).

Proposition 3.4 *The steady-state output probability distribution of a recency-rank calculator applied to the output sequence of a discrete memoryless source with alphabet* $\mathcal{A} = \{u_1, u_2, \dots, u_N\}$ *and corresponding probabilities* $\alpha_1, \alpha_2, \dots, \alpha_N$ *satisfies*

$$\begin{aligned} P_R(r) &= \sum_{k=1}^{N!} P_{R|S}(r|s_k)P_S(s_k) \\ &= \sum_{(i_1,\dots,i_N)\in S_N} \alpha_{i_r} \frac{\alpha_{i_1}\cdots\alpha_{i_{N-1}}}{(1-\alpha_{i_1})\cdots(1-\alpha_{i_1}-\ldots-\alpha_{i_{N-2}})} \end{aligned} \tag{3.7}$$

for $r = 1 \dots N$.

The following theorem shows that the output distribution of a recency-rank calculator is monotone non-increasing.

Theorem 3.2 *When a recency-rank calculator is used on the output sequence of a discrete memoryless source with strictly positive probabilities* $\alpha_1, \alpha_2, \dots, \alpha_N$ *to yield the output sequence* $R_1, R_2, \dots$, *the steady-state distribution* $P_R(.)$ *of the output ranks* R *satisfies the inequality*

$$\max_i \alpha_i \geq P_R(1) \geq P_R(2) \geq \ldots \geq P_R(N) \geq \min_i \alpha_i \tag{3.8}$$

with equality everywhere only when the input distribution is $\alpha_i = 1/N$ *for* $i = 1, 2, \dots, N$. *Otherwise all inequalities are strict.*

Proof: The proof of this theorem follows the same outline as the proof of Theorem 3.1. We refer to that proof whenever the derivation is identical. For a start, the proof of the outer inequalities is the same as for Theorem 3.1.

Regarding the inner inequalities, we will treat the last inequality separately. For $r = 1, 2, \ldots, N-2$, we can write

$$\begin{aligned}\xi(r) &\stackrel{\text{def}}{=} P_R(r) - P_R(r+1) \\ &= \sum_{(i_1 \ldots i_N) \in S_N} (\alpha_{i_r} - \alpha_{i_{r+1}}) \frac{\alpha_{i_1} \cdots \alpha_{i_{N-1}}}{(1-\alpha_{i_1}) \cdots (1-\alpha_{i_1} - \ldots - \alpha_{i_{N-2}})}.\end{aligned}$$

As in the proof of Theorem 3.1, we partition the set S_N into the set S_N^1 and the set S_N^2, where each permutation $(i_1, \ldots, i_N)$ in S_N^1 has a "twin" permutation $(i_1, \ldots, i_{r+1}, i_r, \ldots, i_N)$ in S_N^2 which is identical up to the elements at positions r and $r+1$ which are exchanged. We can now write $\xi(r)$ as a sum over the set S_N^1, which, after a few transformations, yields

$$\xi(r) = \sum_{(i_1, \ldots, i_N) \in S_N^1} \frac{(\alpha_{i_r} - \alpha_{i_{r+1}})^2}{1 - \alpha_{i_1} - \ldots - \alpha_{i_{r-1}} - \alpha_{i_{r+1}}} P_S(u_{i_1} u_{i_2} \ldots u_{i_N}).$$

Every term in this sum is non-negative which implies that the sum is non-negative. Furthermore, the sum is equal to zero if and only if every term in the sum is equal to zero, which is the case only when the source probabilities $\alpha_1, \ldots, \alpha_N$ are all equal[2].

The same approach can be used to prove the last inner inequality and its condition for equality, except that a different expression is obtained:

$$\begin{aligned}\xi(N-1) &\stackrel{\text{def}}{=} P_R(N-1) - P_R(N) \\ &= \sum_{(i_1, \ldots, i_N) \in S_N^1} \frac{(\alpha_{i_r} - \alpha_{i_{r+1}})^2}{\alpha_{i_{N-1}}} P_S(u_{i_1} u_{i_2} \ldots u_{i_N}).\end{aligned}$$

This sum confirms the last inner inequality. □

As for Theorem 3.1, Theorem 3.2 will not hold in general for discrete stationary sources which are not memoryless. The counter-example stated for the competitive list will produce the same result with a recency-rank calculator. In simulations, it appeared that the output distribution of a

[2]If it was possible to prove the relation $\xi_{\text{CLT}}(r) \geq \xi_{\text{RRC}}(r)$ for $r = 1, \ldots, N-1$, it would demonstrate in what sense the output distribution of the competitive list transformer is "more decreasing" than that of the recency-rank calculator. I do not know whether this relation holds.

recency-rank calculator has a generally decreasing shape for most practical sources, but it is seldom monotone non-increasing. In particular, many sources were observed for which the probability $P_R(2)$ that the rank be equal to two is much smaller than the probabilities of subsequent ranks.

Comparing the competitive list and the recency-rank

Both the competitive list transformer and the recency-rank calculator realize a non-expanding invertible transformation of the source output sequence, i.e., they produce one output rank per source output symbol, and the alphabet $\{1, 2, \ldots, N\}$ of the ranks has the same size as the source alphabet. For a block $U_1 \ldots U_M$ of source symbols and the corresponding block $R_1 \ldots R_M$ of ranks, we can write

$$\begin{aligned} H(R_1 \ldots R_M U_1 \ldots U_M) &= H(R_1 \ldots R_M | U_1 \ldots U_M) + H(U_1 \ldots U_M) \\ &= H(U_1 \ldots U_M | R_1 \ldots R_M) + H(R_1 \ldots R_M). \end{aligned}$$

The initial state of the list is a fixed system parameter known both to the encoder and the decoder. Thus, we can write

$$H(R_1 \ldots R_M | U_1 \ldots U_M) = 0,$$

reflecting the fact that the transformation is a deterministic operation. Similarly,

$$H(U_1 \ldots U_M | R_1 \ldots R_M) = 0,$$

since it is possible to recover the original source output sequence using the sequence of ranks, i.e., the inverse transformation is also a deterministic operation. Thus, we have

$$H(U_1 \ldots U_M) = H(R_1 \ldots R_M).$$

If we let M go to infinity, we obtain

$$H_\infty(R) = H_\infty(U). \tag{3.9}$$

When the input to the transformer is the output of a discrete memoryless source, we have in addition

$$H(U) = H_\infty(U),$$

i.e., the single-letter entropy of the source is equal to its entropy rate. However, for the output of the source transformer, in general

$$H(R) \geq H_\infty(R).$$

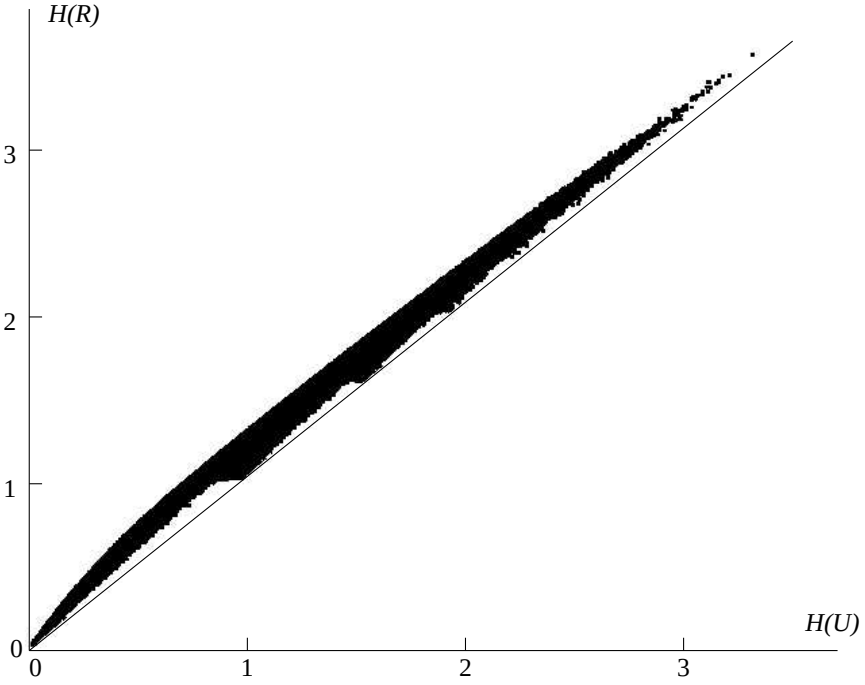


Figure 3.11: Output Entropy vs. Input Entropy of the Competitive List Transformer

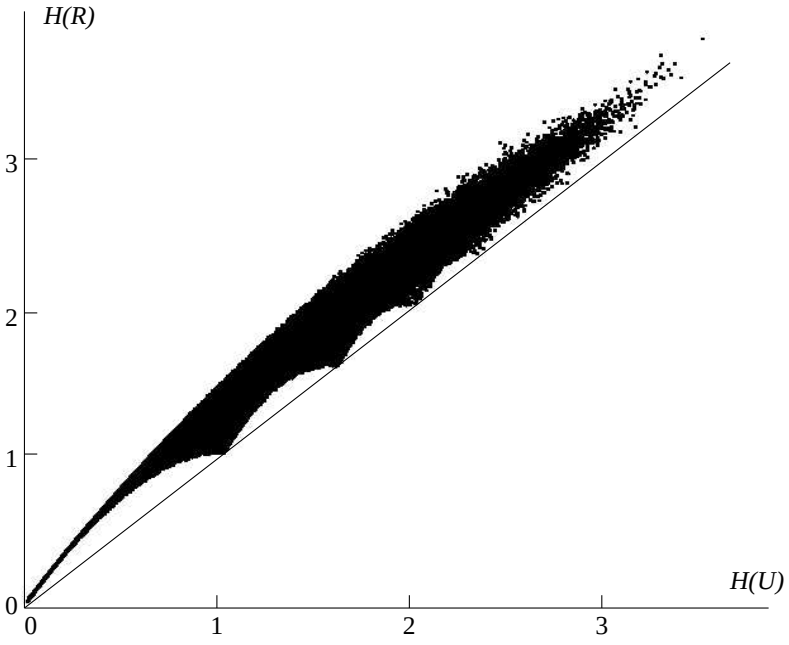


Figure 3.12: Output Entropy vs. Input Entropy of the Recency-Rank Calculator

This shows that the source transformation may induce an increase of the single-letter entropy compared to the entropy of the source. Since the universal encoder we plan to apply to the output of the source transformer is based solely upon its marginal probability distribution, the single-letter entropy of the output ranks is a lower bound for the expected codeword length of the universal encoder. Therefore, the competitive list transformer and the recency-rank calculator can be compared based on the single-letter entropy of their output ranks in function of the entropy of the source.

Figure 3.11 represents the results of a simulation where the entropy of the output of a competitive list transformer was estimated for 100'000 randomly chosen discrete memoryless sources of alphabet size 64. Each source generated an output sequence of 10^6 symbols which were processed by the competitive list transformer. The graph plots the measured single-letter entropy at the output of the competitive list transformer versus the entropy of the discrete memoryless source. Each point in the graph represents one such measurement, i.e., the graph contains 100'000 points. For comparison, Figure 3.12 shows the outcome of the same simulation where a recency-rank calculator was used instead of a competitive list transformer. The recency-rank calculator was applied to the same 100'000 source output sequences as the competitive list transformer. The graph shows a considerably larger spread, or increase of entropy for the recency-rank calculator than for the competitive list transformer. What the graph does not show is that the measured output entropy of the recency-rank calculator was greater than the measured output entropy of the competitive list transformer for each of the 100'000 sources simulated. It is not known whether this relation can be shown analytically. Furthermore, we observe that the output entropy of both transformers "touches" the line $H(U) = H(R)$ for some sources. This happens only for entropies $H(U) = \log_2(L)$ for $L = 1, 2, 3 \ldots$. This effect is explained by considering the statements of Theorems 3.1 and 3.2: when the probability distribution of the source approaches a uniform distribution of size L (i.e., $P_U(u_k) = 1/L$ for some k and $P_U(u_k) = 0$ for all other k), the output distribution also approaches a uniform distribution of size L and the output entropy approaches the input entropy.

The results of this simulation indicate that the competitive list transformer will achieve better results in practice than the recency-rank calculator, tough both transformers realize a mapping of all discrete memoryless sources into the class of sources whose marginal distribution is monotone non-increasing. As pointed out earlier, the recency-rank calculator is expected to converge towards its steady state faster than the competitive

list transformer. However, we will overcome this disadvantage of the competitive list transformer by starting with an empty competitive list and letting its size grow as new symbols are observed in the source output. In the next section, we show how this can be realized in practice, while still allowing the inverse transformer to recover those new symbols from the sequence of ranks. Before we proceed to the discussion of our practical coding algorithm, we conclude this section heavy with propositions and theorems by stating a list of the "theorems" that we were unable to prove:

- We could not show in what sense the output distribution of the competitive list transformer is "more decreasing" than that of the recency-rank calculator.
- The graphs in Figures 3.11 and 3.12 seem to indicate that there exists an upper bound for the single-letter entropy of the output of both source transformers given the entropy of the discrete memoryless source they are applied to. We were not able to determine such an upper bound analytically.
- We could not show that the output entropy of the competitive list is always smaller than that of the recency-rank calculator though our simulation results indicate that this may be true.
- Though we showed that the output distribution of both source transformers is monotone non-increasing, we do not know whether every monotone non-increasing distribution can be the output distribution of a competitive list transformer or of a recency-rank calculator.
- Finally, though simulations indicate that the output distribution of a competitive list transformer tends to be monotone non-increasing for a larger class of sources than the class of all discrete memoryless sources, we were unable to determine such a class.

3.3 Conditional Competitive Lists

In the previous section, we described two source transformers whose steady-state output distribution is monotone non-increasing when they are applied to a discrete memoryless source. In this section, we show how *conditional coding* can be used to transform the output of some discrete stationary source into an array of essentially memoryless sources. Furthermore, we show how the convergence of a competitive list transformer towards its steady state can be accelerated in a practical context-tree source coding algorithm. Finally, we return to our discussion of coding distributions

for the output of a source transformer and discuss an alternative to the optimal coding distributions presented in the previous chapter.

Context-tree algorithms

We begin by introducing the concept of a context-tree algorithm the way it is commonly implemented, i.e., as an adaptive algorithm where the encoding is based on estimated probability distributions. Following this, we will discuss how a context-tree algorithm can be modified to include source transformers and universal arithmetic coding.

Context-tree source coding algorithms encode the output of a source based on estimates of its *conditional* probability distributions. The context of length w of the source at any given moment is defined as the sequence of w past output symbols of the source. A context-tree algorithm holds a list of possible source contexts, which can be represented as a tree. In this tree, the root represents the empty context ; the nodes at depth 1 from the root represent the last symbol emitted by the source ; the nodes at depth k represent the k-th past output symbol of the source. An example context-tree for a ternary source with alphabet {A,B,C} is

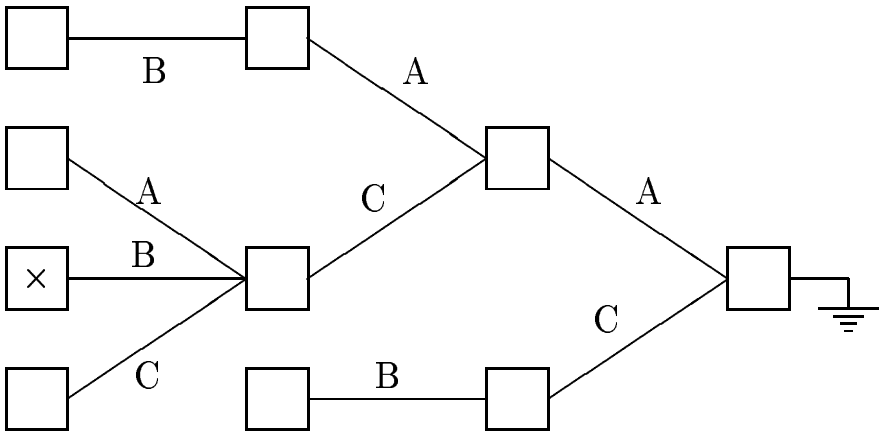


Figure 3.13: An example context-tree for a ternary source

shown in Figure 3.13. At every node, a context-tree algorithm holds an estimate of the conditional probability distribution of the source given the corresponding context. When a source symbol is to be encoded, an arithmetic encoder uses the estimated distribution in the deepest node corresponding to the current source context. For example, if the past output sequence of the source ended with the sequence " . . . , B, A, B, C, A", the node selected in the tree of Figure 3.13 is the node corresponding to the path A-C-B which is marked with a cross in the figure. On the other hand, if the past output sequence of the source ended with the letter "B", the estimated probabilities in the root context would be used

for the encoding as there is no path starting from the root corresponding to the letter "B". A context-tree algorithm must be designed in such a way that the tree and the probability estimates are identical at every step in the encoder and in the decoder. Once a symbol is encoded using the probability estimated in a given node, the probability estimates can be updated and the tree shape can be modified by adding or possibly removing nodes in the tree. The decoder decodes the received symbol based on an identical tree containing the same estimated probabilities as the tree in the encoder. Once it has determined which symbol was encoded, the decoder can perform the same modifications in the tree as the encoder.

Universal source coding algorithms are commonly compared based on three performance measures:

- the *compression rate* κ, usually measured by the average number of output bits per input byte [bpB].
- the *storage requirement* σ of the algorithm [kBytes]
- the *throughput* θ, measured by the number of kBytes compressed per second.

Context-tree algorithms such as PPM [8] and CTW [24] have achieved by far the best compression rates ever obtained. However, they have a relatively slow throughput and they require a lot of storage.

When designing a context-tree algorithm, the lengths of the contexts used are the principal parameters to consider in order to ensure a good compression rate. Longer contexts give rise to a tradeoff between one advantage and two disadvantages. The advantage of using longer contexts can be stated as follows:

> The theoretical lower bound for the compression rate at a given context is determined by the conditional entropy of the source given the context. On average, increasing the length of contexts can only reduce the conditional entropy for discrete stationary sources (see [30, Theorem 3.5.1]), thereby allowing better compression rates.

The disadvantages of longer contexts can be summarized as follows:

1. At every context, there is a transient phase during which the statistics accumulated are insufficient to provide a good compression rate. Longer contexts occur less frequently than short ones. Therefore, the length of the transient phase will increase with the length of the contexts used.

2. If N is the alphabet size of the source, there are N^w possible contexts of length w. Increasing the length of the contexts used can drastically increase the storage requirement of a context-tree algorithm.

The two context-tree algorithms cited above are designed mainly to overcome the first disadvantage stated. Both algorithms are based on clever ideas to achieve a seamless transition between shorter contexts and longer contexts as the latter fade out of their transient phase. However, these algorithms make no effort to reduce the storage required by their respective context trees.

We now present our own context-tree algorithm which is based on source transformation and universal arithmetic encoding. Since it appears to be the fashion of the trade to name algorithms using three letter acronyms, we choose to call our algorithm "CCL" to stand for "Conditional Competitive Lists".

> The CCL algorithm is a context-tree algorithm where the probability estimation in each node of the context-tree is replaced by a competitive list transformer. The role of the competitive list transformer is to convert the partial output sequence of the source processed in each context into a sequence whose first-order probability distribution is monotone non-increasing as stipulated by Theorem 3.1. The output sequence of each competitive list transformer is encoded using a universal arithmetic encoder.

There are two caveats in the scheme we just described:

1. Theorem 3.1 holds only when the input of the competitive list transformer is the output sequence of a discrete memoryless source.

2. Theorem 3.1 holds only for the steady-state output distribution of the competitive list transformer.

The second caveat stated is not particular to the CCL algorithm. As mentioned earlier, both PPM and CTW are built around two methods to overcome the transient of the probability estimator in each context. The CCL algorithm must overcome the transients of its source transformers rather than transients of probability estimators. A method which allows one to overcome this transient will be described shortly using what we shall call a "conditional competitive list". This approach is closely related to the method used by CTW to overcome the transient of its probability estimators.

The first caveat stated poses a more significant threat to the integrity of the CCL algorithm. In order to tackle this problem, it is necessary to investigate the conditions under which a context-tree algorithm can be used to encode the output of a source efficiently.

- if a source coding algorithm models sources as context-trees where each node in the tree contains the *true* conditional probability distribution of the source given the corresponding context, the algorithm approaches the entropy rate $H_\infty(U)$ of any discrete stationary source on average with every increase of the tree depth. This holds because the conditional entropy $H(U_L|U_1 \ldots U_{L-1})$ is monotone non-increasing in L and its limit is the entropy rate $H_\infty(U)$ of the source (see [30, Theorem 3.5.1]).

- when a context-tree algorithm uses *estimated* conditional probability distributions in each node as is the case for PPM and CTW, it is not enough for the source to be stationary as this does not ensure that the probability distribution estimated in each node of the context-tree converges towards the true conditional distribution of the source. A sufficient condition is for the source to be a unifilar ergodic Markov[3] source. We refer to [27] for a definition of unifilar ergodic Markov sources.

- for the CCL algorithm, it is not sufficient for the source to be a unifilar ergodic Markov source. Theorem 3.1 requires the partial sequence observed in each context to be memoryless to ensure that the output distribution of the source transformer converges towards a monotone non-increasing distribution. This can be brought about by requiring the source to be a unifilar ergodic *finite-order* Markov source and that the depth of every node in the tree in which a source transformation is performed and used by the encoder be at least equal to the order of the source for that node.

We have stated the strict conditions under which Theorem 3.1 provides a base for the CCL algorithm to encode the output of a source. In prac-

[3]There are Markov sources, Markov diagrams, Markov models and Markov chains but no one was able to tell me which illustrious Markov they are named after. I asked many experts and received many humorous answers. The closest serious reference I found is in the book "Concrete Mathematics" by Graham, Patashnik & Knuth, which lists an "Andreĭ Andreevich Markov the elder" in the index but gives no reference to his work. Among the funny answers I received, I cannot resist quoting Marc Semionovitch Pinsker's answer: "I know at least fifteen Russian mathematicians named Markov. The name does not refer to one Markov in particular but is used to honor the great dynasty of the Markov's." and Yuri Shtarkov's answer: "It's a spelling mistake. It should be called a 'Shtarkov' information source!"

tice, we cannot verify whether these conditions hold for a given source. Instead, we will rely on our observation that the output distribution of a competitive list transformer tends to be monotone non-increasing for many practical sources which are not memoryless and assume that the output distribution of a competitive list transformer is always monotone non-increasing.

The practical advantage of the CCL algorithm is a reduction of the storage requirement when compared to other context-tree algorithms. A probability estimator requires one counter for every probability estimated. A competitive list transformer requires only a list of symbols to be stored.

In order to achieve a high throughput, a source coding algorithm must process sequences of symbols at every coding step rather than encoding one symbol at a time. The Lempel-Ziv algorithm [25] is one such algorithm. Since the CCL algorithm performs one coding operation per source symbol, its throughput is expected to be in the same order of magnitude as the throughput of other context-tree algorithms.

Finally, our context-tree algorithm replaces adaptive coding, which has a redundancy of zero in the steady state, by source transformation and universal arithmetic coding, where an increase of entropy occurs in the source transformation and redundancy is added to the resulting single-letter entropy by the universal encoder. Therefore, we cannot expect the CCL algorithm to attain the compression rates achieved by PPM[4] when the same context-tree is used. Therefore, the CCL algorithm trades a worse compression rate for a reduction of the storage required.

The conditional competitive list transformer

In order to overcome the long transient phase of the competitive list transformers in every node of the context-tree, the operation of the competitive list transformer is modified during its transient phase. Instead of starting a competitive list transformer with the full alphabet of the source, the *conditional* competitive list transformer in a context-tree contains only symbols which have been encountered in the past in the corresponding context. If a symbol is received which is contained in the list, the conditional competitive list transformer operates like a normal competitive list transformer. If a symbol is received which is not yet in the list, this symbol is appended at the end of the list.

[4]The comparison with CTW is more difficult since CTW can only be implemented using a binary alphabet in every node, and the CCL algorithm must have a larger alphabet to be of any interest. Since the performance of CTW is comparable or better than that of PPM, the statement holds for CTW as well

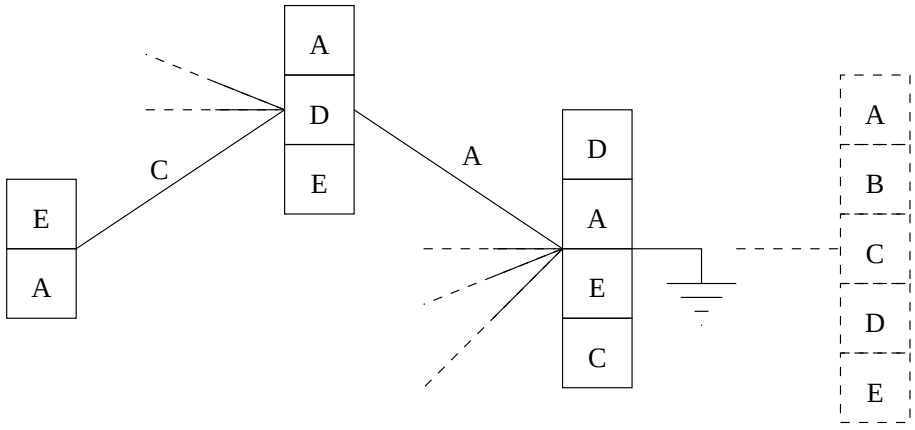


Figure 3.14: Conditional competitive list transformers in a context tree

We must provide a way for the decoder to find out which new symbols are appended to the end of the list in the conditional transformer. The solution to this problem is to extend the list in the current context using the lists contained in its parent nodes in the context tree. In this scheme, whenever a symbol is not contained in the current list, the search for this symbol is automatically continued in its parent context, its grand-parent context and so on. A "virtual" node is imagined beyond the root which contains all the symbols of the source listed in alphabetical order to ensure that the symbol will always be found.

When the parent nodes are searched, the rank is only incremented for symbols which have not been encountered in a previous list. Thus, the rank emitted will remain a number between 1 and N as for a non-conditional competitive list transformer. Based on this rank, the decoder can replicate the operation of the encoder and find the symbol encoded in the node where the encoder found it. Once the symbol is found, both the encoder and the decoder can append it to the list in the current node and in all the parent nodes visited before the symbol was found.

Example: Figure 3.14 represents part of a context-tree used to encode a source with alphabet {A, B, C, D, E}. If the past output of the source ended with the sequence " ... , C, A", then the node corresponding to the current context of the source is the deepest node represented in the figure. For the conditional competitive list transformer in this node, the assignment of source symbols to ranks is given in the following table:

Rank	Symbol
1	E
2	A
3	D
4	C
5	B

Symbols 'E' and 'A' are assigned their ranks in the competitive list of the current node. Symbol 'D' is assigned rank 3 because it is the first symbol encountered in the parent node which is not already in the current node. Symbol 'C' is assigned rank 4 because it is the next new symbol along the genealogical path of the current node. Since symbol 'B' is found in no node along the genealogical path, it must be sought in the "virtual" node beyond the root which contains all symbols ordered alphabetically.

This method bears a certain similarity to both PPM and CTW. In PPM, each node in the tree contains only estimated probabilities for the symbols which have been encountered in the past in the corresponding context. Symbols for which a probability has been estimated in the current node are encoded using this probability. When a symbol occurs for which there is no estimated probability in the current node, an "escape" symbol is encoded, thereby instructing the decoder that the probability estimates in the parent node will be used to encode the next source symbol. The estimated probabilities in the parent node are rescaled to exclude those symbols which are already in the current node. If the symbol is not found in the parent node, the grand-parent node is used, etc. If the symbol is not found in the root node, a "virtual" node beyond the root is used which assigns the uniform probability distribution to the alphabet of the source.

The CTW algorithm combines the probabilities estimated in the current node with the probabilities estimated in the nodes along the genealogical path of the current node to determine the coding probabilities used at every encoding operation. Contrary to PPM, the CCL algorithm must not encode an "escape" symbol when the source symbol to be encoded is not found in the current node. As a result, the loss which incurs when using an oversized context-tree to model the source is very small. In this sense, the method used in the CCL algorithm to overcome the transient of its source transformers is closer to the CTW algorithm than it is to the PPM algorithm.

The Average Distribution

For the CCL algorithm, there is an alternative to the optimal coding distribution for monotone non-increasing probability distributions derived in the previous chapter. The alternative consists in using the estimated average output distribution of the source transformers in the context-tree as a coding distribution to encode the output of each source transformer. This method was introduced on intuitive grounds by P. Portmann and I. Spieler in their semester project [17] and presented in [21].

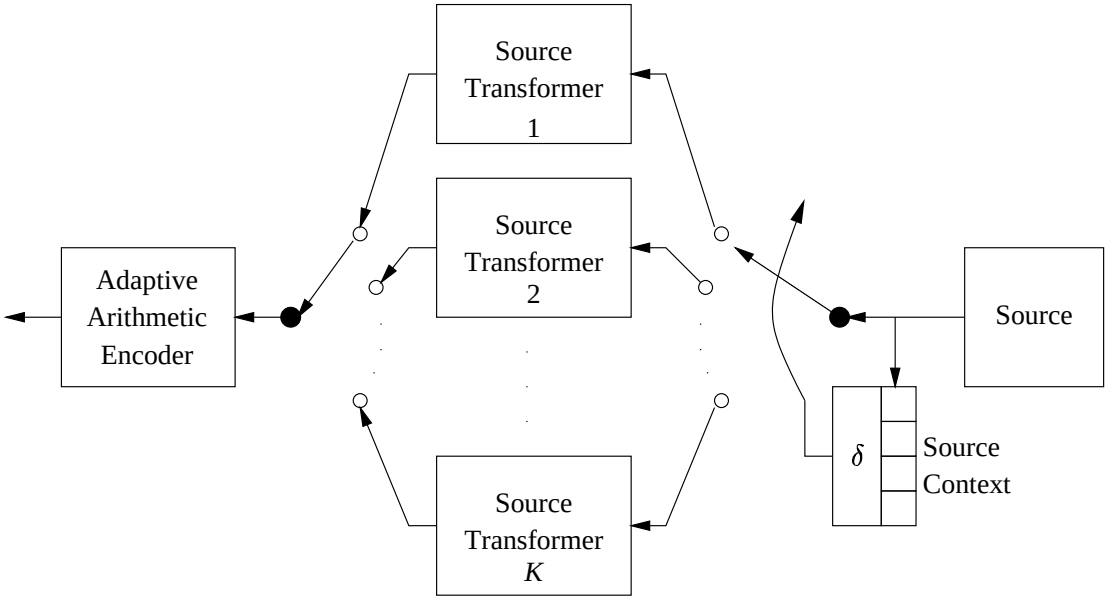


Figure 3.15: The CCL algorithm using the average coding distribution

The practical implementation of this method is shown in Figure 3.15. In the figure, the nodes of the context-tree are represented as an array of source transformers numbered from 1 to K. The output of the source is forwarded to a shift register which holds the current context. Based on this context, the decision block δ, which holds the structure of the context-tree, decides which source transformer to forward the current source output symbol to.

Instead of using the optimal coding distribution for monotone sources to encode the output sequence of each source transformer, the output symbols of each transformer are now merged back into one sequence of ranks. This "merged" sequence is encoded by an adaptive arithmetic encoder, which uses its estimate of the marginal probability distribution of the symbols in the merged sequence as a coding distribution. In the merged sequence, each consecutive rank is generated by one of the K source transformers whose marginal output distribution is assumed to be monotone non-increasing. Thus, the marginal distribution of the merged sequence is also monotone non-increasing. By estimating the marginal probability distribution of the merged sequence, we are effectively averaging the marginal output distributions of all K source transformers where each transformer is weighted by the probability of occurrence of its corresponding context.

Thus, the estimated marginal distribution of the merged sequence approaches the optimal coding distribution with a prior for the class $\mathcal{S}$ of output distributions of the source transformers in the context-tree, in the sense of Proposition 2.9. In other words, this distribution minimizes the expected redundancy of the arithmetic encoder over the class $\mathcal{S}$. Therefore, we expect a better performance of our context-tree algorithm when

the average coding distribution is used than when the optimal coding distribution for the set $\mathcal{M}_N$ of monotone sources is used, since the former minimizes the expected redundancy for the actual output distributions of the source transformers, whereas the latter minimizes the worst-case redundancy for the class of all monotone non-increasing distributions, of which the class $\mathcal{S}$ of marginal output distributions of the source transformers is only a sub-class. The results in the next section will show whether the average coding distribution also achieves a better performance than the optimal coding distribution for the class $\mathcal{M}_{N,\underline{f},c}$ of all monotone non-increasing distributions of expected value c, when a partial source estimator is used to estimate the expected value c at the output of each source transformer.

The practical advantage of the average coding distribution is that the task of "reducing" the entropy of the source with a context-tree is separated from the task of encoding the resulting sequence. The first block of the algorithm contains the source context, the decision block δ and the array of source transformers. This block performs an invertible transformation of the output sequence of the source, converting it into a sequence of the same length, the same alphabet size, and therefore the same entropy rate, but whose single-letter entropy is expected to be smaller than the single-letter entropy of the source. The second block of the algorithm is the adaptive arithmetic encoder which performs the actual compression, with an expected codeword length approaching the single-letter entropy of its input sequence.

3.4 Results and Discussion

In our discussion of context-tree algorithms, we have consciously avoided the question of how the structure of a context-tree is modified while encoding a source sequence. There appears to be no known theory dictating when it is best to add or delete a node from a context-tree. The context-tree algorithms known to us follow heuristic rather than theoretical rules to build up their context-tree.

The CCL algorithm is no different in this respect. Two parameters limit the size of the context-tree:

- the maximum tree depth $w_{\max}$
- the maximum number $K_{\max}$ of nodes in the tree

The algorithm starts with a context-tree containing only the root node. While the number of nodes in the tree is smaller than the maximum, a

new node is created whenever the depth of the node corresponding to the current source context is smaller than $w_{\max}$. Existing nodes are never deleted from the context tree. Once the maximum number of nodes is attained, the structure of the tree remains unchanged for all subsequent encoding operations.

We give an outline of the operations performed by the CCL algorithm for every source symbol encoded:

1. Locate the deepest node in the tree corresponding to the current source context.

2. Search for the input symbol in the competitive lists and compute its rank:

   - Start at the deepest node and proceed along the path to the root until the symbol is found.
   - Increment the rank only for entries in each competitive list which were not encountered in previous lists visited.
   - Append the input symbol at the end of the competitive lists in which the symbol was not found.

3. Update the competitive lists in the remaining nodes along the path to the root.

4. Add one new node in the context-tree if the depth of the current deepest node is less than $w_{\max}$ and the number of nodes in the tree is less than $K_{\max}$.

5. Estimate the coding probability of the rank obtained.

6. Encode the rank obtained using an arithmetic encoder.

It is not clear whether step 3 in the algorithm is necessary. If this step is included, the competitive list transformer in an intermediate node of the tree reflects the conditional probability of the source given the corresponding source context. If step 3 is removed, the transformer in an intermediate node reflects the conditional probability given that the source context contains this node *but no deeper node* in the tree.

In the remaining part of this section, the performance of the CCL algorithm is evaluated in terms of the three performance criteria for universal source coding algorithms described earlier. We begin by comparing a few variations of the CCL algorithm in order to determine the best choice for the coding distribution. The performance of the resulting algorithm is compared to the performance of other common universal source coding

		κ [bpB]	σ [kByte]	θ [kByte/s]
1.)	optimal for $\mathcal{M}_N$	3.28	1100	130
2.)	average distribution	2.83	1100	130
3.)	optimal for $\mathcal{M}_{N,\underline{f},c}$	2.49	1950	45
4.)	step 3 removed	2.84	1100	150

Table 3.1: The performance of variations of the CCL algorithm

algorithms. For all the results presented, the CCL algorithm follows the outline given above (except in one measurement where the possibility of removing step 3 from the algorithm is investigated). The variations of the algorithm that are compared differ only in the coding distribution which is used by the arithmetic encoder to encode the output symbols of the source transformers.

Fine-tuning the CCL algorithm

The performance of the CCL algorithm is evaluated when compressing the files[5] of the "Calgary Corpus" described in [28]. The values of the compression rate κ and the storage σ are calculated by taking the arithmetic average of the values measured individually for the 14 files of the Calgary Corpus. The value of the throughput θ is calculated by dividing the total size of the corpus (in kBytes) by the CPU-time required by the algorithm to compress all the files. Table 3.1 lists the performance measured for variations of the CCL algorithm. In the results shown, the algorithm is given a maximum number of nodes in the context-tree $K_{\max} = 32'000$. The maximum tree depth $w_{\max}$ is optimized individually for each file compressed. We now describe the variations tested, which are numbered in the text as in Table 3.1.

1. The CCL algorithm is implemented as described in the outline above. The optimal distribution for the polytope $\mathcal{M}_N$ of monotone non-increasing probability distributions derived in (2.25) is used as the coding distribution by the arithmetic encoder. The alphabet size is $N = 256$ for the files of the Calgary Corpus.

2. The CCL algorithm is implemented using the average distribution described in the previous section. The partial output sequences of

[5]The files of the Calgary Corpus are available on the Internet at the time of writing from

`ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/`

the source transformers are merged back into one sequence and the estimated marginal probability distribution of the merged sequence is used as a coding distribution by the arithmetic encoder.

3. The CCL algorithm is implemented with a partial source estimator in each node of the context-tree. Each partial estimator estimates the average of the output distribution of the source transformer in its node, using the recursive equation

$$c[k] = (1 - \delta)c[k - 1] + \delta R[k],$$

where $c[k]$ is the k-th estimate of the average and $R[k]$ is the k-th symbol in the output sequence of the source transformer. The variable δ is called the "forgetting factor" and its value is optimized individually for each file compressed. Since it is not possible to compute the optimal coding distribution for the polytope $\mathcal{M}_{N,\underline{f},c}$ for every symbol encoded, the CCL algorithm uses a lookup table containing optimal distributions for a few values of c, which have been pre-computed using the iterated Arimoto-Blahut algorithm of the previous chapter. For every symbol encoded, the arithmetic encoder selects the coding distribution in the lookup table whose average is closest to the estimated average c in the current node.

4. The CCL algorithm is implemented using the average distribution as in variation 2. Step 3 is removed from the algorithm, i.e., the algorithm does not update the competitive lists on the genealogic path to the root node once it has found the source symbol in a given node.

The compression rates obtained for variations 1 and 2 confirm our claim that the average distribution must perform at least as well as the optimal coding distribution for the polytope $\mathcal{M}_N$. However, it is worth pointing out that the difference between the compression rates measured lies far below the worst-case redundancy $\rho = 1.60$ which we computed in the previous chapter for the polytope $\mathcal{M}_{256}$.

Variation 3 of the CCL algorithm clearly outperforms all the other variations in terms of the compression rate ρ. However, the cost of loading and maintaining a lookup table of optimal distributions is reflected in the bad performance of this variation in terms of the throughput θ and the storage requirement σ. This variation of the algorithm shows what is achievable in terms of the compression rate by making the coding distribution of the arithmetic encoder node-specific. In order to be useful in practice, an alternative implementation would have to be devised which does not require a lookup table of probability distributions.

Algorithm	κ [bpB]	σ [kByte]	$\frac{\theta}{\theta_{\text{gzip}}}$
CCL1 ($K_{\max} = 13'000$)	2.92	490	.116
CCL2 ($K_{\max} = 200'000$)	2.79	2940	.108
gzip	2.71	2000	1
PPM_MA	2.24	6300	.077
PPM_CO	2.57	2100	.089
LZSS	3.74	520	.24
LZW	3.7	1250	1.24
EFLZ	4.44	1250	5.95
`compress`	3.64	970	1.61

Table 3.2: The performance of the CCL algorithm compared with the performances of other universal source coding algorithms

The difference in compression rate between variations 2 and 4 is negligible. Nevertheless, the difference is not fortuitous, because it becomes larger for larger values of $K_{\max}$. Thus, step 3 in the CCL algorithm achieves a slight improvement of the compression rate. However, the improvement achieved does not justify the cost of a lower throughput which results from updating additional competitive lists along the path to the node at each coding step.

Variation 4 of the CCL algorithm clearly comes out as the winner in the overall comparison. Hereafter, when the performance of the CCL algorithm is compared to the performance of other source coding algorithms, we will restrict ourselves to this version of the algorithm[6].

Comparison with other universal source coding algorithms

The performance of the CCL algorithm is compared to the performance of other universal source coding algorithms when compressing the 14 files of the Calgary Corpus. The performance values for the other source coding algorithms are based on the measurements realized by Tonezzer and Vontobel in the course of their diploma project [22]. Table 3.2 shows the performances measured. Since the throughput θ is machine-dependent, it is given relative to the throughput of the `gzip` command executed on the same machine.

[6] I cannot help shedding a tear of sorrow as we part with the coding distributions derived in the previous chapter. It seems as if the practical "engineering" approach has triumphed over the theoretically pleasing "optimal" coding distribution for a polytope.

The programs LZW, LZSS and EFLZ implement variations of the Lempel-Ziv algorithm. The programs `compress` and `gzip` are software packages which combine the Lempel-Ziv algorithm with various tricks to improve the effective performance. The programs PPM_MA and PPM_CO implement two variations of the PPM algorithm. Both programs follow an outline which is very similar to the outline of the CCL algorithm described earlier, except that those programs extend the deepest node to the full length $w_{\max}$ of the source context at every coding step rather than creating one node at a time as in the CCL algorithm. The maximum number of nodes in their context-tree is $K_{\max} = 13'000$ for PPM_CO and $K_{\max} = 200'000$ for PPM_MA. Both algorithms automatically determine an appropriate value of the maximum tree depth $w_{\max}$ for each file compressed. For detailed descriptions of the source coding algorithms compared, we refer to [22].

The results shown suggest a positive assessment of the CCL algorithm. The algorithm achieves its goal of providing an acceptable compression rate while reducing the storage requirement which commonly afflicts context-tree algorithms. The version of the algorithm which uses 13'000 nodes breaks the record for low storage among the algorithms tested, and its compression rate is far better than its closest contenders. This indicates that the algorithm may be of practical use in situations were storage is sparse, which are precisely the situations when a good compression algorithm is most direly needed. For example, this is the case in a hand-held computer which, being deprived of a mass storage device, must hold its operating software and the data it processes within a few MBytes of tightly allocated RAM storage.

The Planetarium Representation

An alternative representation of the performance triplet of the various source coding algorithms compared requires the reader to participate in a little thought experiment. The reader is invited to take a seat in a comfortable armchair placed at the origin of the three-dimensional space $\mathbb{R}^3$, with a view into the "sky", which is thought to be the positive eighth of $\mathbb{R}^3$. For the sake of this experiment, we kindly ask the reader to wear an eye-patch so as to neutralize his or her sense of perspective.

For every triplet $(\kappa, \sigma, 1/\theta)$ associated with a source coding algorithm, we hang a ball of a given diameter δ at the corresponding coordinates in $\mathbb{R}^3$. The throughput is represented by $1/\theta$ rather than θ because all three measures must obey the rule "the smaller the better" in this experiment. Each parameter in a measured triplet is normalized so that the component-wise average of all triplets gives the point (1, 1, 1). Since the reader

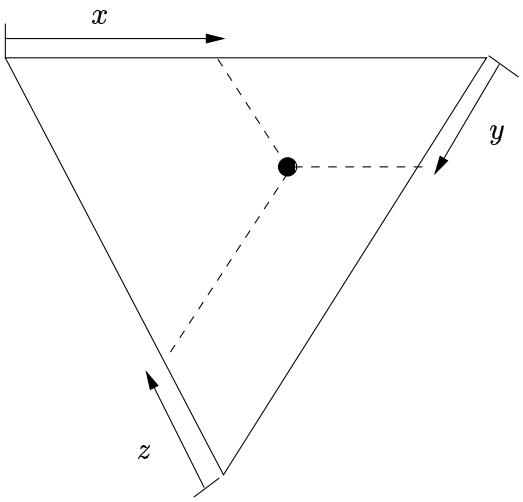


Figure 3.16: The coordinates of points on the unit triangle of $\mathbb{R}^3$

has no sense of perspective, we can represent his or her view of the balls approximately by projecting each ball onto a triangular window spanned by the unit vectors of $\mathbb{R}^3$, which we call the "unit triangle". Given that all balls have the same diameter δ, the size of each ball projected onto the unit triangle is inversely proportional to the norm of the corresponding triplet $(\kappa, 1/\theta, \sigma)$. The position of a ball in the triangle reveals how a source coding algorithm distributes its emphasis on the three performance parameters κ, $1/\theta$ and σ, and the size of a ball gives an indication as to the overall performance of the algorithm.

By a geometric argument, it is possible to show that the components of a point on the unit triangle are determined by projecting the point onto the edges of the triangle[7] in the manner indicated by Figure 3.16. The resulting representation is shown in Figure 3.17 in a somewhat artistic rendering, where spikes have been added to the balls to accentuate the planetary aspect of the illustration.

In this representation, the winners are `compress` and `gzip` which attain the best overall performance by offering an excellent tradeoff between compression rate, storage requirement and throughput. The main looser is PPM_MA which, while standing at the forefront in terms of the prestigious compression rate, neglects the other two parameters completely. The CCL1 algorithm, which uses a context-tree with 13'000 nodes achieves the best overall performance among context-tree algorithms.

While this representation illustrates the performance of the various source coding algorithms in a very graphical manner, it has the weakness of giving equal importance to the three performance parameters of each

[7]This transformation of the unit triangle was shown to me by Prof. George Yadigaroglu of the ETH Zürich, who uses it to represent a triplet of parameters which describe a thermodynamic process in the core of a nuclear reactor.

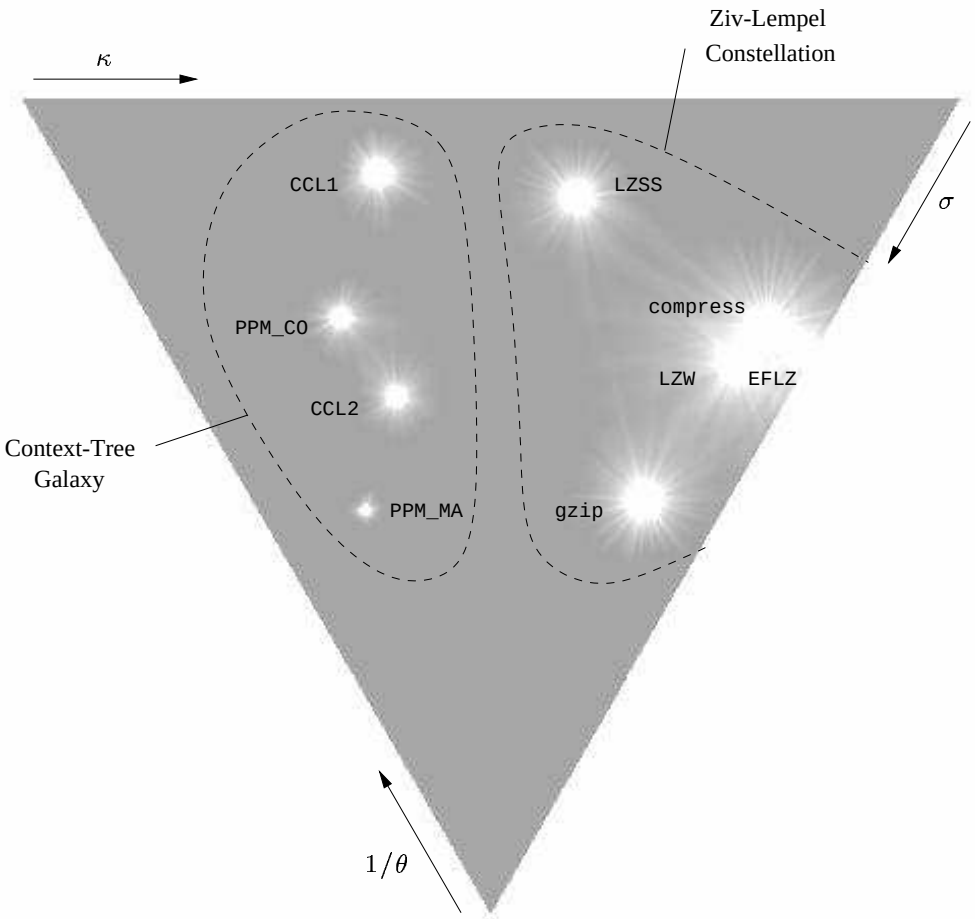


Figure 3.17: The planetarium representation of the performance of source coding algorithms

algorithm. In practice, every application dictates the importance of each parameter and a source coding algorithm is chosen accordingly.

We thank the reader for participating in this thought experiment. Please do not forget to remove your eye-patch before proceeding to the final pages of this dissertation.

Appendix: An Alternative Formulation of Proposition 3.2

We begin by defining a matrix function that was originally introduced by Cauchy in 1812:

Definition 3.1 *The* permanent *of an $N \times N$ matrix A is the sum of all products of N elements of the matrix where every element is taken from a different row and a different column. It is written $Per(A)$.*

A permanent is closely related to the determinant of the same matrix. The determinant is the sum of the same products, except that half of those products must be multiplied by -1 when computing the determinant. It might seem that the permanent should be easier to compute than the determinant, but this is not true because the transformation rules that make it easy to compute a determinant are not valid in general for the permanent. One rule that is extendable to permanents is the factorization rule by which a permanent can be written as a sum of row or column elements times the permanent of the sub-matrix obtained by removing the row and the column of the current element from the original matrix.

The only permanents and sub-permanents that we will be interested in are those of the Vandermonde matrix

$$V = \begin{bmatrix} \alpha_1^{N-1} & \alpha_2^{N-1} & \dots & \alpha_N^{N-1} \\ \alpha_1^{N-2} & \alpha_2^{N-2} & \dots & \alpha_N^{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^1 & \alpha_2^1 & \dots & \alpha_N^1 \\ \alpha_1^0 & \alpha_2^0 & \dots & \alpha_N^0 \end{bmatrix}.$$

The Δ in the state probability of a competitive list can now be expressed as

$$\Delta = \mathrm{Per}(V).$$

We write v_{ij} to denote the element at row i and column j in the Vandermonde matrix V, and we write

$$\Delta_{ij} = \mathrm{Per}(V_{ij}),$$

where V_{ij} denotes the sub-matrix obtained by removing the i-th row and the j-th column from V. We now define the matrix

$$W = \frac{1}{\Delta} \begin{bmatrix} v_{11}\Delta_{11} & v_{12}\Delta_{12} & \dots & v_{1N}\Delta_{1N} \\ v_{21}\Delta_{21} & v_{22}\Delta_{22} & \dots & v_{2N}\Delta_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ v_{N1}\Delta_{N1} & v_{N2}\Delta_{N2} & \dots & v_{NN}\Delta_{NN} \end{bmatrix},$$

which is a doubly stochastic matrix, because the sum of $v_{ij}\Delta_{ij}$ for $i = 1 \dots N$ or for $j = 1 \dots N$ is equal to the permanent of V, by the factorization rule mentioned above.

We have finally reached the "simple" expression for the steady-state rank probabilities of the competitive list. The vector corresponding to the output probability distribution $P_R = [P_R(1), P_R(2), \dots, P_R(N)]^T$ of a competitive list can be expressed as

$$P_R = \mathbf{W} P_U, \tag{3.10}$$

where $P_U = [P_U(u_1), P_U(u_2), \dots, P_U(u_N)]^T$ is the probability vector of the discrete memoryless source.

Conclusion

We give a summary of the results presented in the three chapters of this dissertation:

- In Chapter 1, we began with a tutorial on arithmetic coding, showing how this source coding method falls within the framework of coding by probability transformation. Applying the principle of coding by probability transformation to block coding for noisy channels led us to Definition 1.1 of the (N, K) block coding capacity. This definition provides a method for designing block codes which could only be implemented for small values of N and K.

  We demonstrated how an arithmetic encoder could be modified to accommodate a rate sufficiently below the capacity of a noisy channel. This arithmetic encoder for noisy channels turned out to be a generic encoder which includes the special cases of block and convolutional encoders. We derived a metric (1.10) to be used by a sequential decoder in conjunction with an arithmetic channel encoder. Finally, we presented the results of simulations in which the performance of the arithmetic encoder was evaluated and compared to the performance of a convolutional encoder.

- In Chapter 2, an alternative to Gallager's derivation of the optimal coding distribution for a finite class of distributions was given. This alternative resulted in Theorems 2.2 and 2.3. An iterated version of the Arimoto-Blahut algorithm was introduced, which is used to determine the optimal coding distribution for a polytope of probability distributions, or, alternatively, to compute the capacity of a channel with a large input alphabet. Using this algorithm, the optimal coding distribution was determined for the polytope $\mathcal{M}_{N,\underline{f},c}$ of all monotone non-increasing probability distributions P with a given average $\underline{f} \cdot P = c$.

- Chapter 3 started with a general framework showing how universal coding could be combined with source transformation. Following this, two source transformers were described, analyzed and compared: the competitive list transformer and the recency-rank calculator. Theorems 3.1 and 3.2 show that the steady-state output distributions of either transformer is monotone non-increasing when the transformer is applied to the output sequence of a discrete memoryless source. Based on this insight, a context-tree algorithm was devised using competitive list transformers in each node of its context-tree and encoding the resulting output symbols with a universal arithmetic encoder. The performance of the algorithm was evaluated and compared to the performances of other common universal source coding algorithms.

Apart from Chapter 1 which may have opened the way for further research on arithmetic coding for noisy channels, many results in the present dissertation have been disappointing. In Chapter 2, a lengthy derivation was given for the solution of a problem for which Gallager provides a much shorter equivalent solution. Chapter 3 started with the hope of using the coding distributions derived in Chapter 2 in a context-tree algorithm, but those distributions had to be dropped to make place for a more practical solution. As a consolation, we conclude this dissertation with a personal remark on the pedagogical value of such an endeavor.

As Information Theory grows into the next millennium, the main problem facing students in the field is that they are given more and more solutions before they have had proper time to think about the corresponding problems. Huffman coding seems obvious when it is learned by students who have not attempted to discover the optimal prefix-free code for a random variable themselves. The Viterbi algorithm looks trivial to a student who has not spent time trying to devise a maximum likelihood decoder for a convolutional or an arithmetic channel encoder. It is common among students to lament about the impossibility of inventing anything new in a field which has an elegant solution for every solvable problem. By returning to the roots of Information Theory and attempting to design coding methods based on its probabilistic fundamentals, I have learned to appreciate the beauty of the solutions provided by the pioneers of Coding and Information Theory over the past fifty years.

Bibliography

[1] Claude Shannon, "A Mathematical Theory of Communication," *Bell Tech. Journ.*, 1948.

[2] Peter Elias, "Error-free Coding," *IEEE Trans. on Inform. Theory*, **IT-3**, pp. 29-37, September 1954.

[3] Peter Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. on Inform. Theory*, **IT-21**, pp. 194-203, March 1975.

[4] Peter Elias, "Interval and Recency Rank Source Coding: Two On-Line Adaptive Variable-Length Schemes," *IEEE Trans. on Inform. Theory*, **IT-33**, pp. 3-10, January 1986.

[5] Suguru Arimoto, "An Algorithm for Computing the Capacity of Arbitrary Discrete Memoryless Channels," *IEEE Trans. on Inform. Theory*, **IT-18**, pp. 14-20, January 1972.

[6] J. L. Bentley, D. D. Sleator, R. E. Tarjan, W. K. Wei, "A Locally Adaptive Data Compression Scheme," *Communications of the ACM*, **Vol. 29, No. 4**, pp. 320-330, April 1986.

[7] Richard E. Blahut, "Computation of Channel Capacity and Rate-Distortion Functions," *IEEE Trans. on Inform. Theory*, **IT-18**, pp. 460-473, July 1972.

[8] J. G. Cleary and I. H. Witten, "Data Compression using Adaptive Coding and Partial String Matching," *IEEE Trans. on Communications*, **COM-32**, pp. 396-402, April 1984.

[9] Lee D. Davisson, "Comments on 'Sequence time coding for data compression'," *Proceedings of the IEEE (Corresp.)* **vol. 54**, p. 2010, December 1966.

[10] Lee D. Davisson, "Universal Noiseless Coding," *IEEE Trans. on Inform. Theory*, **IT-19**, pp. 783-795, November 1973.

[11] Robert Fano, Technical Report No. 65, The Research Laboratory of Electronics, M.I.T., March 17, 1949.

[12] Robert G. Gallager, "Source Coding With Side Information and Universal Coding," **unpublished**, Submitted to the *IEEE Trans. on Inform. Theory*, September 1976.

[13] I. M. Jacobs and E. R. Berlekamp, "A Lower Bound on the Distribution of Computation for Sequential Decoding," *IEEE Trans. on Inform. Theory*, **IT-13**, pp. 167-174, April 1967.

[14] T. J. Lynch, "Sequence time coding for data compression," *Proceedings of the IEEE (Corresp.)* **Vol. 54**, pp. 1490-1491, October 1966.

[15] James L. Massey, "Variable-Length Codes and the Fano Metric," *IEEE Trans. on Inform. Theory*, **IT-18**, pp. 196-198, January 1972.

[16] R. C. Pasco (1976), *Source Coding Algorithms for Fast Data Compression*, PhD Thesis, Dept. of EE, Stanford University, CA.

[17] Patricia Portmann and Ivo Spieler, "Generalized Recency-Rank Source coding: A Practical On-Line Adaptive Scheme," *Semester Project Nr. 9503*, Signal and Information Processing Laboratory, ETH Zürich, Wintersemester 1994/95.

[18] J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," *IBM Journal of Research and Development*, **Vol. 20**, p. 198, 1976.

[19] R. Rivest, "On Self-Organizing Sequential Search Heuristics," *Communications of the ACM*, **Vol. 19, No. 2**, pp. 63-67, February 1976.

[20] Boris Ya. Ryabko, "Data compression by means of a book stack," *Problems of Information Transmission*, **Vol. 16, No. 4**, pp. 16-21, 1980 (in Russian), **Vol. 16, No. 4**, pp. 265-269, 1981 (in English).

[21] J. Sayir, I. Spieler and P. Portmann, "Conditional Recency-Ranking for Source Coding," in *Proc. IEEE Inform. Theory Workshop* (Haifa, Israel, June 9-13, 1996), p. 61.

[22] Ralph M. Tonezzer and Pascal O. Vontobel, "Exploring the Ziv-Lempel Algorithm," *Diploma Project Nr. 6803*, Signal and Information Processing Laboratory, ETH Zürich, Wintersemester 1996/97.

[23] F. M. J. Willems, "Universal Data Compression and Repetition Times," *IEEE Trans. on Inform. Theory*, **IT-35**, pp. 54-58, January 1989.

[24] F. M. J. Willems, Y. M. Shtarkov and T. J. Tjalkens, "The Context-Tree Weighting Method: Basic Properties," *IEEE Trans. on Inform. Theory*, **IT-41**, pp. 653-664, May 1995.

[25] Jacob Ziv and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Inform. Theory*, **IT-23**, pp. 337-343, May 1977.

[26] Norman Abramson (1963), *Information Theory and Coding*, McGraw-Hill, New York.

[27] Robert B. Ash (1965) *Information Theory*, Dover Publications Inc., ISBN 0-486-66521-6.

[28] T. C. Bell, J. G. Cleary and I. H. Witten (1990), *Text Compression*, Prentice Hall, ISBN 0-13-911991-4.

[29] T. M. Cover and J. A. Thomas (1991), *Elements of Information Theory*, John Wiley and Sons, ISBN 0-471-06259-6.

[30] Robert G. Gallager (1968), *Information Theory and Reliable Communications*, John Wiley and Sons, ISBN 0-471-29048-3.

[31] Rafail Krichevsky (1994), *Universal Compression and Retrieval*, Kluwer Accademic Publishers, ISBN 0-7923-2672-5.

[32] Shu Lin and Daniel J. Costello, Jr. (1983), *Error Control Coding: Fundamentals and Applications*, Prentice Hall Inc., ISBN 0-13-283796-X.

[33] Olvi L. Mangasarian (1969), *Nonlinear Programming*, McGraw-Hill, New York.

[34] John G. Proakis (1983), *Digital Communications*, McGraw-Hill, ISBN 0-07-100269-3.

[35] R. Tyrell Rockafellar (1970), *Convex Analysis*, Princeton University Press, ISBN 0-691-01586-4.

Curriculum Vitae

Jossy Sayir
Steinmüristr. 20
CH-8123 Ebmatingen, Switzerland

8 June 1968 Born in Zürich, Switzerland.

1974-86 Attended primary and secondary school at the Ecole Française de Zürich, obtained Baccalauréat Français Type C (Mathématiques/ Physique), mention "Bien".

1986-91 Attended and graduated from the ETH Zürich, received diploma in Electrical Engineering as dipl. El.-Ing. ETH.

1991-93 Employed by Motorola Communications Israel as a development engineer, worked on the first digital mobile radio system developed by Motorola.

1993-95 Employed as a Teaching Assistant at the Signal and Information Processing Lab, ETH Zürich.

December 1993 Co-founded the Jazz-Band "Who Cares?" with Markus Schenkel at the Signal and Information Processing Laboratory.

1995-98 Employed as a Research Assistant at the Signal and Information Processing Lab. and enlisted as a PhD student.

1994-98 Served as Secretary, then Chairperson of the IEEE Student Branch at the ETH Zürich.

1999 ... Temporarily employed by Supercomputing Systems in Zürich while seeking employment in or around London, England.