On the Reduction of Entropy Coding Complexity via Symbol Grouping: I – Redundancy Analysis and Optimal Alphabet Partition

Amir Said
Imaging Systems Laboratory
HP Laboratories Palo Alto
HPL-2004-145
August 23, 2004*

E-mail: said@hpl.hp.com, said@ieee.org

data compression, symbol grouping, dynamic programming, Monge matrices

We analyze the technique for reducing the complexity of entropy coding that consists in the *a priori* grouping of the source alphabet symbols, and in the decomposition of the coding process in two stages: first coding the number of the symbol's group with a more complex method, followed by coding the symbol's rank inside its group using a less complex method, or simply using its binary representation. This technique proved to be quite effective, yielding great reductions in complexity with reasonably small losses in compression, even when the groups are designed with empiric methods. It is widely used in practice and it is an important part in standards like MPEG and JPEG. However, the theory to explain its effectiveness and optimization had not been sufficiently developed. In this work, we provide a theoretical analysis of the properties of these methods in general circumstances. Next, we study the problem of finding optimal source alphabet partitions. We demonstrate a necessary optimality condition that eliminates most of the possible solutions, and guarantees that a more constrained version of the problem, which can be solved via dynamic programming, provides the optimal solutions. In addition, we show that the data used by the dynamic programming optimization has properties similar to the Monge matrices, allowing the use of much more efficient solution methods.

* Internal Accession Date Only
Approved for External Publication

1 Introduction

1.1 Motivation

Techniques for reducing the computational complexity of data coding are commonly developed employing both theory and heuristics. On one hand, we have very general results from information theory, and a variety of coding methods of varying complexity that had been developed for any type of data source. On the other hand, we frequently have the coding methods improved by exploiting properties from a particular type of source (text, images, audio, etc.). In consequence, a large number of *ad hoc* cost-reduction methods had been developed, but the techniques created for one type of source and equipment may not be used directly for another type.

There is a need to find out and study techniques for reducing coding costs that have wide application, and are valid for many measures of computational complexity, and to clearly identify the range of situations in which they are effective. One such general technique to reduce the coding complexity, which we call *symbol grouping,* uses the following strategy:

- The source alphabet is partitioned, before coding, into a relatively small number of groups;
- Each data symbol is coded in two steps: first the group that it belongs (called *group number*) is coded; followed by the rank of that particular symbol inside that group (the *symbol index*);
- When coding the pair (group number, symbol index) the group number is entropy-coded with a powerful and complex method, while the symbol index is coded with a simple and fast method, which can be simply the binary representation of that index.

Fig. 1 shows a diagram of such scheme when the symbol indexes are coded using their binary representation. With this very general technique we commonly can trade small losses in compression with very significant reductions in complexity.

1.2 Practical Applications

The effectiveness and usefulness of symbol grouping are now well established because it has been used quite extensively, in a variety of practical applications. The technique may not be immediately recognized because in most cases its complexity reduction is used to create a more elaborate coding method, and consequently it is not implemented exactly as presented above.

For example, symbol grouping is employed by the JPEG and MPEG standards, where it is used in the VLI (variable length integer) representation [12, 13, 19], in which the magnitude

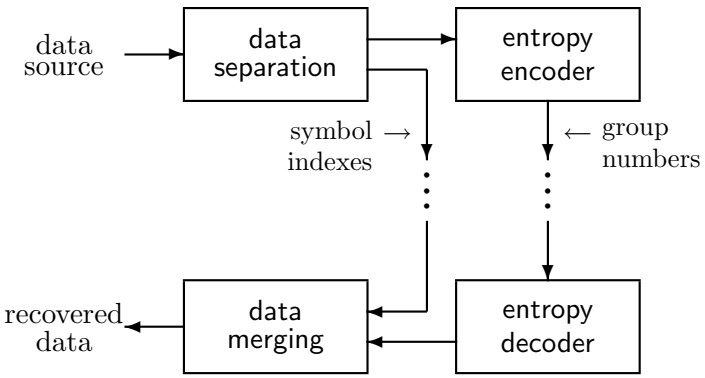


Figure 1: System for data compression using the symbol grouping method for complexity reduction.

category corresponds to the group number, and the information in the extra bits corresponds to the symbol index. Its complexity reduction enables efficiently coding the magnitude category (group number) together with the run-length with a single Huffman code, and exploiting the statistical dependence between these two types of data. The use of symbol grouping in these standards can be traced back to its earlier choice for facsimile coding [7]. The practical advantages of identifying the information that can be efficiently coded with its binary representation was also identified in applications that used Golomb, Golomb-Rice, and similar codes [1, 2, 5].

The same approach is used in the embedded image coding methods like EZW, SPIHT, and similar [14, 17, 24, 34], where the wavelet coefficient significance data corresponds to a group number, which is coded with set-partitioning (optionally followed by arithmetic coding). The sign and refinement data, which define the symbol index, can be coded using simply one sign bit and one bit per coefficient refinement. Methods that use this approach, but without producing embedded bit streams, have similar compression efficiency [18, 23].

TThe advantages of symbol grouping come from the combination of two factors. First, using simply the binary representation to represent the symbol index is significantly faster than any form of entropy coding [33]. The second factor is defined by how the complexity of coding algorithms depends on alphabet size [21, 33]. There are many practical difficulties when the data alphabet is large, ranging from the time required to design and implement the code, to the amount of memory necessary to store the codebooks. When the coding method is adaptive there is also a possible compression loss due to the long time required to gather sufficiently accurate statistics. These problems get much worse when exploiting a statistical dependence between the source samples by, for example, coding several symbols together, or designing context-based codes.

The price to pay for this complexity reduction is that there is possibly some loss in compression, which can be minimized by exploiting properties of the symbol probabilities

and using the proper alphabet partition. For example, the large alphabet problem can be alleviated using an extra "overflow" symbol to indicate that less frequent symbols are to be coded with a less complex method. Golomb-Rice codes use the fact that if the symbols have the geometric probability distribution they can be grouped and coded with the same number of bits with a small loss [3]. Symbol grouping is significantly more efficient than each of these techniques used separately because it combines both in a synergistic manner, greatly increasing their effectiveness (this is explained in the mathematical analysis in Section 2). Consequently, it commonly yields a net gain in compression because the loss is typically smaller than the coding gains obtained from the more complex source modeling that it enables.

It is worth mentioning that another important practical advantage of symbol grouping (which is not analyzed in this work) is the increased immunity to errors in data transmission. An error in a variable-length code can lead to unrecoverable error propagation, but with the symbol grouping technique the errors in the symbol index bits can only affect one reconstructed source sample [25, 29], and this can be effectively exploited in sophisticated unequal-error protection schemes [26].

1.3 Previous Work

Despite the widespread adoption of symbol grouping among coding practitioners, it is not mentioned in coding textbooks, except when in a standard's context, and its theoretical analysis received comparatively little attention. The factors that enable its remarkably small compression loss are identified in the preliminary theoretical analysis by Said and Pearlman [18]. In the same work a heuristic algorithm for finding good alphabet partitions in real time is proposed, but numerical experiments demonstrate another property of symbol grouping, which is the ability to use the same partition for sources with some similar features, but very different parameters. This is shown in [18] by developing a new image coding method using symbol grouping exactly as defined above, and obtaining excellent compression with the same alphabet partition in applications ranging from high compression ratios to lossless compression (0.25–5 bits/symbol). Subsequent work confirmed these results [22, 23].

A more recent work covers the same technique and some variations. Chen *et al.* [30] analyze the application of symbol grouping for semi-adaptive coding, and develop dynamic programming and heuristic solutions to the problem of finding optimal groups. While there are similarities with this work, a very important difference is that they reduce complexity by using Golomb codes for the symbol indexes. This complicates the analysis significantly because in this case many more probability patterns are acceptable inside a group. Furthermore, Golomb codes are defined for infinite alphabets, and it is necessary to consider early termination strategies for coding the finite number of symbol indexes in a group.

The terminology used in this document is meant to avoid confusion between two very different coding techniques that we worked on. In [18] the complexity reduction technique

studied here is called *alphabet partitioning*, and later called *amplitude partitioning* in [23]. However, it was used together with set-partitioning coding [17], in which sets of source *samples* are sequentially divided, and we found that our terminology sometimes lead to the incorrect assumption that the two technique are equivalent. Consequently, in this work we decided to call it *symbol grouping*.

1.4 New Contributions and Organization

The main contributions of this work are the following:

- A theoretical analysis of the compression loss (coding redundancy) caused by symbol grouping, extending the work in [18] to include
  - A very precise approximation of the redundancy function, which enables a more intuitive interpretation of its properties.
  - An analysis of the structure and convexity of the grouping redundancy function.
- An study of the optimal alphabet partition problem, considering how it fits in the general class of optimal set-partitioning problems, the type of combinatorial problem, and the number of solutions.
- A theorem that states a necessary optimality condition for the optimal alphabet partitioning problem, which enables us to state that only one type of solution, which can be much more easily obtained via dynamic programming, can be optimal.
- A general dynamic programming solution to the optimal alphabet partitioning problem, plus a set of mathematical properties of the quantities needed by the algorithm, which enable much faster solution of the problem.
- A study of the computational complexity of the solution algorithms, demonstrating the advantages of exploiting the particular properties of our design problem.

This document is organized as follows. In Section 2 we introduce the notation and present the initial analysis of symbol grouping. In Section 3 we study the problem of finding optimal partitions, by first presenting a theorem with a necessary optimality condition, by the development of a dynamic programming problem formulation, and the study of its mathematical properties. Section 4 discusses the computational implementation and complexity of the alphabet partition algorithms. Section 5 presents the conclusions.

2 Analysis of Symbol Grouping

2.1 Analysis using Information Theory

We can use some well-known facts from information theory to understand, in very general terms, the properties exploited by symbol grouping for complexity reduction. It uses the fact that we can split the coding process in two or more stages without necessarily adding redundancy. Let S be a random data source, which is to be coded using side information Z. The optimal coding rate is the conditional entropy $H(S|Z)$. If we have a one-to-one transformation between S and two other random sources, G and X, we have [11]

$$H(S|Z) = H(G, X|Z) = H(G|Z) + H(X|G, Z), \tag{1}$$

where $H(\cdot|\cdot)$ is the conditional entropy of the corresponding discrete random variables.

In the symbol grouping case we have G corresponding to the group numbers, and X corresponding to the symbol indexes. We aim to decompose the data samples in a way that we can use a code $C = \Phi(G)$, and a low-complexity coding method with rate $R(X|C)$ such that

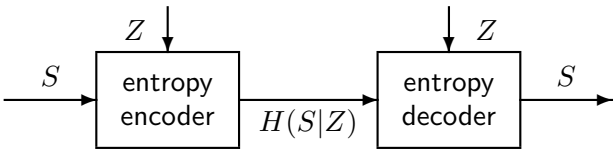
$$R(X|C) - H(X|G, Z) \leq \varepsilon\, H(S|Z), \quad 0 \leq \varepsilon \ll 1, \tag{2}$$

i.e., the relative loss in compression (relative coding redundancy) is very small.

Fig. 2 shows two alternative ways of coding data. While the first system seems to be simpler and more elegant, for complex data sources it may require an unreasonable amount of computational resources for achieving rates near $H(S|Z)$. The second system requires some effort to separate the data components, but can provide a great overall reduction in complexity. While this solution may be clearly more attractive, the real practical problem is in the identification of good decompositions of S into G and X that satisfies (2). It is important to note that the system in Fig. 2(b) works with data from a single source sample: it is not difficult to find different source samples that are nearly independent, but we seek the *component* of information in a single sample that can be separated without significant loss.

In this document we study this problem under the assumption that that all symbols in a group use the same (or nearly the same) number of bits, and thus the code C corresponds roughly to the size of the group to which a symbol belongs. For example, when the group size is a power of two (*dyadic groups*), C corresponds to the integer number of bits required to code the symbol index. Otherwise, it may correspond to a fractional number of bits that may be coded using arithmetic coding [31, 32], or an approximation using combination of two numbers of bits, as in Golomb codes [1] (cf. Section 3.6).

(a) General coding system using side information



(b) Coding system using symbol grouping for complexity reduction

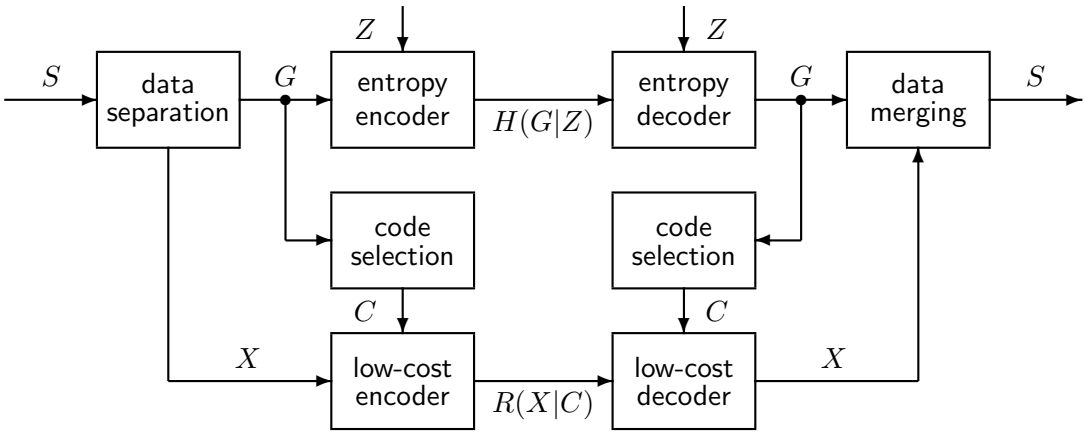


Figure 2: Two alternative systems for data coding: the low-complexity version identifies and separates the data that can be coded with lower computational cost.

2.2 Basic Definitions, Notation, and Analysis

Most of the results in this section are presented in ref. [18], but they are repeated here because they help introduce the notation and the basic assumptions used throughout the document. In Section 2.3 we start introducing new results.

We consider a random data source that generates independent and identically distributed (i.i.d.) symbols belonging to an alphabet $\mathcal{A} = \{1, 2, \ldots, N_s\}$. We use a single random variable S, with a probability mass function $p(s)$, to represent samples from this source. A vector called $\mathbf{p}$ is used to represent the set of symbol probabilities, and $p_s = p(s)$ denotes the probability of data symbol $s \in \mathcal{A}$.

The entropy of this source is

$$H(\mathbf{p}) = \sum_{s \in \mathcal{A}} p_s \log_2 \left(\frac{1}{p_s} \right) \quad \text{bits/symbol.} \tag{3}$$

In the computation of entropies we use the definition $p \log(1/p) = 0$ when $p = 0$.

We are assuming memoryless sources only to simplify the notation: the analysis below can be easily extended to context-based coding by simply replacing the probabilities with

conditional probabilities.

We create a partition $\mathcal{P} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{N_g}\}$ of the source alphabet $\mathcal{A}$ by defining N_g nonempty sets of data symbols $\mathcal{G}_n$, such that

$$\bigcup_{n=1}^{N_g} \mathcal{G}_n = \mathcal{A}, \quad \mathcal{G}_m \cap \mathcal{G}_n = \varnothing \quad \text{if} \quad m \neq n. \tag{4}$$

Each set $\mathcal{G}_n$ is called the *group* of data symbols with number n. We represent the *group number* of symbol s by the function

$$g(s) = \{n : s \in \mathcal{G}_n\}, \quad s = 1, 2, \ldots, N_s. \tag{5}$$

Representing the number of elements in the set $\mathcal{G}_n$ by $|\mathcal{G}_n|$, we identify each data symbol inside a group by defining the *index* of symbol s using a function $x(s)$ that assigns new numbers for each symbol in a group, such that

$$\bigcup_{s \in \mathcal{G}_n} x(s) = \{1, 2, \ldots, |\mathcal{G}_n|\}, \quad n = 1, 2, \ldots, N_g. \tag{6}$$

With these two definitions we have a one-to-one mapping between s and the ordered pair $(g(s), x(s))$.

The probability that a data symbol s belongs to group $\mathcal{G}_n$, $\text{Prob}(g(s) = n)$, is represented by

$$\rho_n = \sum_{s \in \mathcal{G}_n} p_s, \quad n = 1, 2, \ldots, N_g, \tag{7}$$

and the conditional probability of symbol s, given the symbol's group number n, is

$$p_{s|n} = \begin{cases} p_s/\rho_n, & g(s) = n, \\ 0, & g(s) \neq n. \end{cases} \tag{8}$$

With the notation defined above we can analyze an ideal entropy-coding method in which we code symbol $s \in \mathcal{A}$ by first coding its group number $g(s)$, and then coding the symbol index $x(s)$. The coding rate, here equal to the combined entropy, is

$$\begin{aligned} H(\mathbf{p}) &= \sum_{n=1}^{N_g} \rho_n \log_2\left(\frac{1}{\rho_n}\right) + \sum_{s \in \mathcal{A}} p_{s|n} \log_2\left(\frac{1}{p_{s|n}}\right) \\ &= \sum_{n=1}^{N_g} \rho_n \log_2\left(\frac{1}{\rho_n}\right) + \sum_{n=1}^{N_g} \sum_{s \in \mathcal{G}_n} p_s \log_2\left(\frac{\rho_n}{p_s}\right). \end{aligned} \tag{9}$$

The first term in the right-hand side of (9) is the rate to code the group number, and the second term is the rate to conditionally code the symbol index.

The combined rate is here clearly equal to the source entropy, since there is no loss in separating the coding process in two steps, as long as the optimal number of bits is used for each symbol [11]. Now, for the analysis of symbol grouping, we assume that all indexes of the group $\mathcal{G}_n$ are coded with the same number of bits, $\log_2 |\mathcal{G}_n|$, knowing that there is loss in compression whenever $\log_2 |\mathcal{G}_n|$ is different from $\log_2(1/p_{s|n})$.

For practical applications we are mainly interested in *dyadic partitions*, i.e., in the cases in which $|\mathcal{G}_n|$ is constrained to be a power of two, because the symbol indexes can be coded using their binary representation without further redundancy. However, we believe it is better to add this and other constraints only at the end of the analysis, since we can obtain more general results without complicating the notation. Furthermore, with arithmetic coding we can have some complexity reduction even if $|\mathcal{G}_n|$ is not a power of two, because some operations can be eliminated when the probability distribution is set to be uniform [31, 32]. There is also the possibility of using one or two number of bits for coding the symbol indexes in a group (cf. Section 3.6).

The bit rate of the simplified coding method, which is the entropy of the group numbers plus an average of the fixed rates used to code the symbol indexes, is obtained with a simple modification of (9):

$$R(\mathcal{P}, \mathbf{p}) = \sum_{n=1}^{N_g} \rho_n \left[\log_2 \left(\frac{1}{\rho_n} \right) + \log_2 |\mathcal{G}_n| \right] \tag{10}$$

$$= \sum_{n=1}^{N_g} \rho_n \log_2 \left(\frac{1}{\rho_n} \right) + \sum_{n=1}^{N_g} \sum_{s \in \mathcal{G}_n} p_s \log_2 |\mathcal{G}_n|. \tag{11}$$

The loss due to symbol grouping, called *grouping redundancy* and represented by $\ell(\mathcal{P}, \mathbf{p})$, is the difference between the new coding rate (11) and the source entropy (9), which can be computed as

$$\ell(\mathcal{P}, \mathbf{p}) = R(\mathcal{P}, \mathbf{p}) - H(\mathbf{p}) = \sum_{n=1}^{N_g} \sum_{s \in \mathcal{G}_n} p_s \log_2 \left(\frac{p_s |\mathcal{G}_n|}{\rho_n} \right). \tag{12}$$

Denoting the average symbol probability in each group by

$$\bar{p}_n = \frac{\rho_n}{|\mathcal{G}_n|}, \quad n = 1, 2, \ldots, N_g, \tag{13}$$

we can rewrite (12) as

$$\ell(\mathcal{P}, \mathbf{p}) = \sum_{n=1}^{N_g} \sum_{s \in \mathcal{G}_n} p_s \log_2 \left(\frac{p_s}{\bar{p}_n} \right) = \sum_{s=1}^{N_s} p_s \log_2 \left(\frac{p_s}{\bar{p}_{g(s)}} \right). \tag{14}$$

Equation (14) shows that the grouping redundancy is in the form of a relative entropy, or, equivalently, the Kullback-Leibler distance [11] between the original source probability

distribution, and a distribution in which the probabilities of all the symbols inside each group $\mathcal{G}_n$ are equal to $\bar{p}_n$. The Kullback-Leibler distance is always non-negative, and can be infinite when $p_s \neq 0$ and $\bar{p}_{g(s)} = 0$, but from (7) and (13) we conclude that in our case the distance is always finite.

The analysis of equations (10) and (14) shows that the relative redundancy $\ell(\mathcal{P}, \mathbf{p})/H(\mathbf{p})$ can be made small if we partition the alphabet in such a way that

1. $(\rho_n \log_2 |\mathcal{G}_n|)/H(\mathbf{p})$ is small, i.e., the relative contribution of the symbols in the group $\mathcal{G}_n$ to $R(\mathcal{P}, \mathbf{p})$ is small.

2. $p_s \approx \bar{p}_n$, for all $s \in \mathcal{G}_n$, i.e., the distribution inside group $\mathcal{G}_n$ is approximately uniform.

The first condition exploits the fact that there may be little loss if we code sub-optimally symbols that occur with very low probability. The second, on the other hand, shows that small losses are possible even when grouping the most probable symbols. What makes the symbol grouping technique so effective, in such a wide range of probability distributions, is that $\ell(\mathcal{P}, \mathbf{p})$ can be small even if not all the terms in (14) are small, since

1. In (14) the probabilities p_s are multiplied by $\log_2(p_s/\bar{p}_n)$, which have magnitudes typically much smaller than $\log_2(1/\bar{p}_n)$ or $\log_2(1/p_s)$. Consequently, the product is typically much smaller than each of these terms.

2. There is a cancellation of positive and negative terms $\log(p_s/\bar{p}_n)$, making the sum in (14) significantly smaller than the sum of the magnitudes.

The second property is not easily recognized in (14), but can be made clear with the help of an approximation.

2.3 Approximation of Kullback-Leibler Distances

Some properties of the Kullback-Leibler distance are commonly identified using simple approximations. For instance, if we apply the approximation derived in [15, p. 239] to (14) we obtain

$$\ell(\mathcal{P}, \mathbf{p}) \approx \frac{1}{\ln(2)} \sum_{s=1}^{N_s} \frac{\left[p_s - \bar{p}_{g(s)}\right]^2}{p_s}, \tag{15}$$

This approximation shows that when we have $p_s \approx \bar{p}_{g(s)}$ in (14) the distance grows slowly, i.e., with the square of $p_s - \bar{p}_{g(s)}$. However, because the squared difference is divided by p_s, it is an approximation that is good only in a very narrow range. In fact, because its error grows quite fast and goes to infinity when $p_s \to 0$, we may get a grossly incorrect intuition about the redundancy.

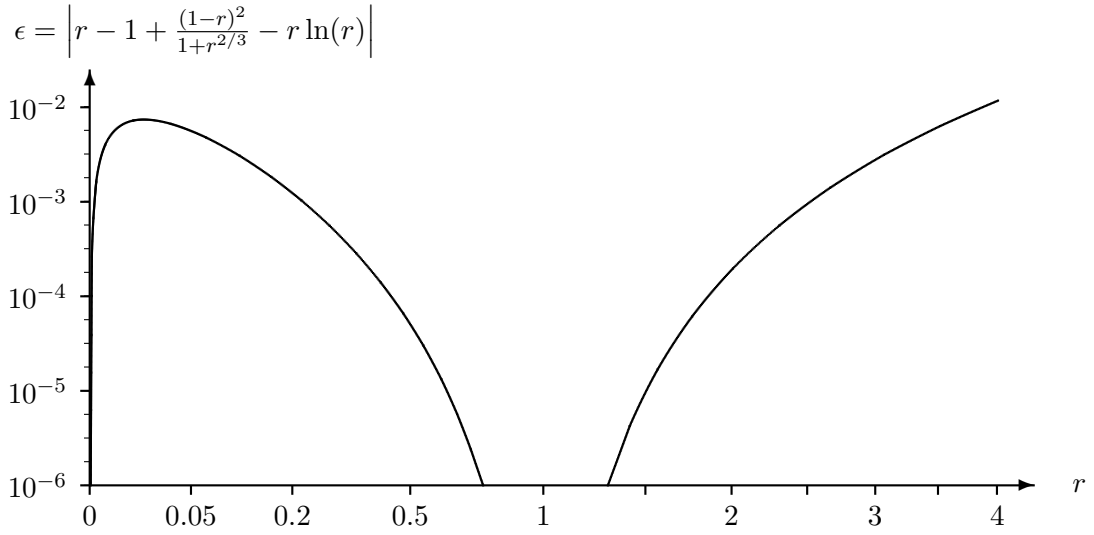


Figure 3: Error of the approximation used for computing the Kullback-Leibler distances.

We found that the approximation

$$r\ \ln(r) \approx r - 1 + \frac{(1-r)^2}{1+r^{2/3}} \tag{16}$$

is also simple enough, and is much better for approximating Kullback-Leibler distances. Similar to other approximations, (16) is more precise when $r \approx 1$. The magnitude of the approximation's error, ϵ, is shown in Fig. 3. Note that we have $\epsilon < 10^{-5}$ in the interval $0.6 \leq r \leq 1.5$, and also note that (16) is a very good approximation in a much wider interval, being exact at $r = 0$, and having $\epsilon < 10^{-2}$ for all $r \leq 3.8$.

The series expansion of both sides of (16) yields

$$r\ \ln(r) = r - 1 + \frac{(1-r)^2}{1+r^{2/3}} + \frac{(1-r)^5}{1620} + \frac{(1-r)^6}{1080} + \cdots \tag{17}$$

which shows that the approximation is exceptionally precise because the exponent 2/3 guarantees that in the point $r = 1$ the first *four* derivatives of the two functions are identical.

Using (16) we can exploit the fact that $\sum_{s\in\mathcal{G}_n}(p_s - \bar{p}_n) = 0$ and obtain the following approximation for the grouping redundancy

$$\ell(\mathcal{P}, \mathbf{p}) \approx \frac{1}{\ln(2)} \sum_{n=1}^{N_g} \bar{p}_n \sum_{s\in\mathcal{G}_n} \frac{(1 - p_s/\bar{p}_n)^2}{1 + (p_s/\bar{p}_n)^{2/3}}, \tag{18}$$

Since groups with zero probability have no redundancy, we can ignore the terms with $\bar{p}_n = 0$. Furthermore, since $\bar{p}_n$ is the average probability of the symbols in the group, normally the

values of $p_s/\bar{p}_n$ are not large, which means that the approximation is quite precise, except in the very unusual cases when we have $p_s \gg \bar{p}_n$.

Note that (18) is a sum with only non-negative terms, enabling a more intuitive interpretation of the cancellation of positive and negative terms that occur in (14). The consequence is that the redundancy grows very slowly with the difference $p_s - \bar{p}_n$ because it is approximately proportional to the sum of the squares of the normalized differences $1 - p_s/\bar{p}_n$, which is divided by number that is never smaller than one. Furthermore, it also shows that the overall sum can be very small because these sums are in turn multiplied by $\bar{p}_n$. In conclusion, the approximation (18) let us see all the different conditions that can make the overall grouping redundancy small, and that with the proper alphabet partition the redundancy can be relatively very small in a wide range of probability distributions.

2.4 Convexity of the Grouping Redundancy Function

An interesting formula for the grouping redundancy is obtained when we normalize the probabilities of the symbols in a group. Let us define, for each group $\mathcal{G}_n$

$$q_{x(s),n} = p_{s|n} = p_s/\rho_n, \quad s \in \mathcal{G}_n \quad \Rightarrow \quad \sum_{i=1}^{|\mathcal{G}_n|} q_{i,n} = 1. \tag{19}$$

The substitution into (12) yields

$$\begin{aligned} \ell(\mathcal{P}, \mathbf{p}) &= \sum_{n=1}^{N_g} \left[\rho_n \log_2 |\mathcal{G}_n| - \sum_{s \in \mathcal{G}_n} p_s \log_2 (\rho_n/p_s) \right] \\ &= \sum_{n=1}^{N_g} \left[\rho_n \log_2 |\mathcal{G}_n| + \rho_n \sum_{i=1}^{|\mathcal{G}_n|} q_{i,n} \log_2 (q_{i,n}) \right] \\ &= \sum_{n=1}^{N_g} \rho_n \left[\log_2 |\mathcal{G}_n| - H(\mathbf{q}_n) \right] \end{aligned} \tag{20}$$

where $\mathbf{q}_n$ is the probability vector containing all probabilities $q_{i,n}$, and $H(\mathbf{q}_n)$ is its entropy.

We can see in (20) that the grouping redundancy can be computed with terms that depend only on the logarithm of the number of elements in the group and the entropy of the normalized probabilities, which is multiplied by the group's probability to produce its average redundancy. Using this formula we can use our knowledge of the entropy function [11] to reach some interesting conclusions. For example, since $0 \leq H(\mathbf{q}_n) \leq \log_2 |\mathcal{G}_n|$, we conclude that, if the group's probability is fixed, the largest redundancy occurs when $|\mathcal{G}_n| - 1$ symbols in the group have zero probability, and $H(\mathbf{q}_n) = 0$.

Some other interesting properties concerning the convexity of $\ell(\mathcal{P}, \mathbf{p})$ can also be derived from (20), but we prefer to start with a more conventional approach, calculating the partial derivatives of $\ell(\mathcal{P}, \mathbf{p})$, which are

$$\frac{\partial \ell(\mathcal{P}, \mathbf{p})}{\partial p_s} = \log_2 \left(\frac{p_s}{\bar{p}_{g(s)}} \right), \quad s = 1, 2, \ldots, N_s, \tag{21}$$

where $\bar{p}_{g(s)}$ is the average probability of the symbols in the group of symbol s. Note that we have undefined derivatives if $p_s = 0$, so we assume in this section that all probabilities are positive. The extension to cases with zero probabilities can be based on the fact that the redundancy function is continuous.

The second derivatives of $\ell(\mathcal{P}, \mathbf{p})$ are given by

$$\frac{\partial^2 \ell(\mathcal{P}, \mathbf{p})}{\partial p_r \partial p_s} = \frac{1}{\ln 2} \times \begin{cases} 0, & g(r) \neq g(s), \\ -1/\rho_{g(s)}, & r \neq s,\ g(r) = g(s), \\ 1/p_s - 1/\rho_{g(s)}, & r = s. \end{cases} \tag{22}$$

For example, if $\mathcal{P} = \{\{1, 2, 3\}, \{4\}, \{5, 6\}\}$ then the Hessian matrix (which contains the second partial derivatives) is

$$\mathbf{H}_\ell(\mathcal{P}, \mathbf{p}) = \frac{1}{\ln 2} \begin{bmatrix} \frac{1}{p_1} - \frac{1}{\rho_1} & -\frac{1}{\rho_1} & -\frac{1}{\rho_1} & 0 & 0 & 0 \\ -\frac{1}{\rho_1} & \frac{1}{p_2} - \frac{1}{\rho_1} & -\frac{1}{\rho_1} & 0 & 0 & 0 \\ -\frac{1}{\rho_1} & -\frac{1}{\rho_1} & \frac{1}{p_3} - \frac{1}{\rho_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{p_5} - \frac{1}{\rho_3} & -\frac{1}{\rho_3} \\ 0 & 0 & 0 & 0 & -\frac{1}{\rho_3} & \frac{1}{p_6} - \frac{1}{\rho_3} \end{bmatrix}$$

Note how this matrix is composed of independent sub-blocks, each corresponding to a different group. From the equations above we can derive a more general result.

Theorem 2.1 *For every alphabet partition $\mathcal{P}$, the grouping redundancy $\ell(\mathcal{P}, \mathbf{p})$ is a convex function of $\mathbf{p}$.*

Proof: We show that the function is convex because the Hessian matrix is semi-definite positive for all probabilities $\mathbf{p}$. Since each block of the matrix corresponding to a group is independent of all the others, the properties of the full matrix are defined by the properties of each sub-matrix. Thus, we can show that they are semi-definite positive by considering the Hessian corresponding to a single group.

For that purpose we assume that $N_g = 1$, and consequently vector $\mathbf{p}$ contains the symbol probabilities in the group $\mathcal{G}_1 = \mathcal{A}$. We also define the vector $\mathbf{v} = [\sqrt{p_1}\ \sqrt{p_2}\ \cdots\ \sqrt{p_{N_s}}]'$. Note that $|\mathbf{v}|^2 = 1$ and $\rho_1 = 1$.

The Hessian matrix of the redundancy of this single group is

$$\mathbf{H}_\ell(\{\mathcal{A}\}, \mathbf{p}) = \frac{1}{\ln 2} \left\{ [\mathrm{diag}(\mathbf{p})]^{-1} - \mathbf{1}\mathbf{1}' \right\}, \tag{23}$$

where $\mathbf{1}$ is the vector with N_s ones, and $\mathrm{diag}(\cdot)$ is the $N_s \times N_s$ diagonal matrix with diagonal elements equal to the vector elements.

We can define $\mathbf{D} = \sqrt{\ln 2}\, \mathrm{diag}(\mathbf{v})$ and

$$\tilde{\mathbf{H}} = \mathbf{D}\, \mathbf{H}_\ell(\{\mathcal{A}\}, \mathbf{p})\, \mathbf{D} = \mathbf{I} - \mathbf{v}\mathbf{v}',$$

such that for any vector $\mathbf{x}$ we have

$$\mathbf{x}'\tilde{\mathbf{H}}\mathbf{x} = \mathbf{x}'\mathbf{x} - (\mathbf{v}'\mathbf{x})^2 = |\mathbf{x}|^2(1 - \cos^2\theta_{vx}),$$

where θ_{vx} is the angle between vectors $\mathbf{v}$ and $\mathbf{x}$.

Since this quadratic form cannot be negative, and is zero only if $\theta_{vx} = 0$ or $\theta_{vx} = \pi$, the conclusion is that matrix $\tilde{\mathbf{H}}$ is semi-definite positive, has one zero eigenvalue, and its corresponding eigenvector is $\mathbf{v}$. This means that $\mathbf{H}_\ell(\{\mathcal{A}\}, \mathbf{p})$ is also semi-definite positive, with one zero eigenvalue, and corresponding eigenvector $\mathbf{p}$.

In the general case the sub-matrix corresponding to each group is semi-definite positive, and the Hessian matrix contains N_g zero eigenvalues (one for each group), and $N_s - N_g$ positive eigenvalues. ∎

We can now interpret the last results using equation (20). The zero eigenvalue for each group, and the fact that the corresponding eigenvector is in the direction of the probabilities, means that the function is linear in radial directions. We can reach the same conclusion from (20) observing that, if the conditional probabilities are fixed, then the redundancy of a group is a linear function of the group's probability. In the orthogonal directions we have the redundancy defined by a constant minus the entropy function. Since the entropy function is strictly concave, the projection of the redundancy function to subspaces orthogonal to radial directions is strictly convex.

For example, Fig. 4 shows the three-dimensional plot of the redundancy of a group with two symbols. The two horizontal axes contain the symbol probabilities, and the vertical axis the redundancy. Note that in the radial direction, i.e., when the normalized probabilities are constant, the redundancy grows linearly, and we have straight lines. In the direction orthogonal to the line $p_1 = p_2$ the redundancy is defined by scaled versions of one minus the binary entropy function. We can also see in this example how the redundancy function defines a convex two-dimensional surface.

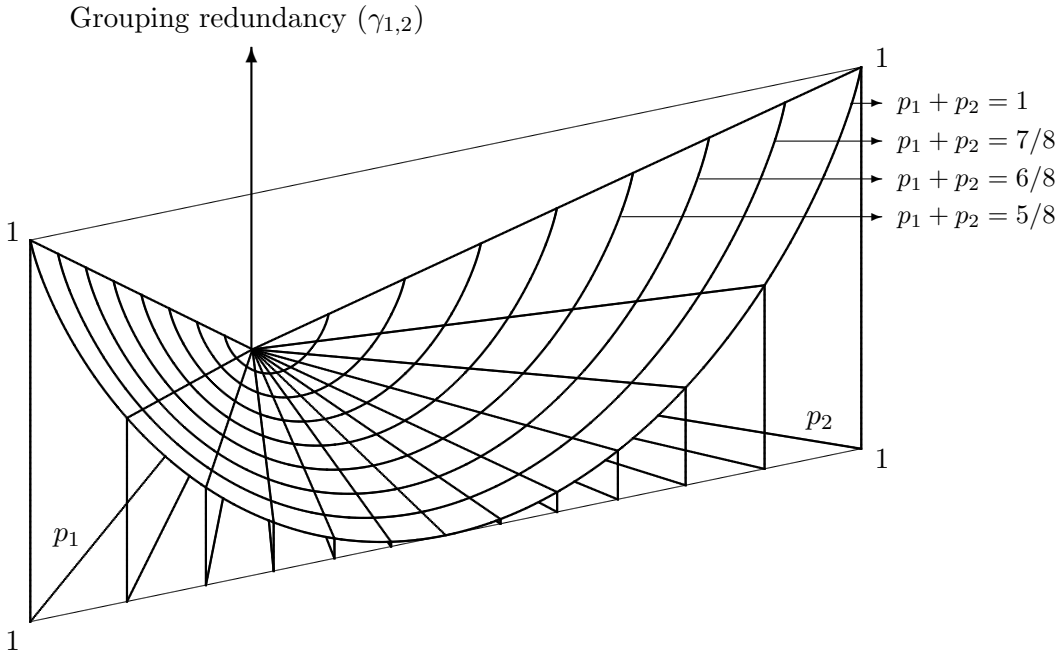


Figure 4: The redundancy of a group with two symbols: it is a convex function, linear in the radial directions, and strictly convex in subspaces orthogonal to the radial directions.

3 Optimal Alphabet Partition

3.1 Problem Definition

The optimal alphabet partition problem can be defined as the following: given a random data source with alphabet $\mathcal{A}$ and symbol probabilities $\mathbf{p}$, we want to find the partition $\mathcal{P}^*$ of $\mathcal{A}$ with the smallest number of groups (minimization of complexity), such that the relative redundancy is not larger than a factor ε. Formally, we have the optimization problem

$$\begin{array}{ll} \text{Minimize}_{\mathcal{P}} & |\mathcal{P}| \\ \text{subject to} & \ell(\mathcal{P}, \mathbf{p}) \leq \varepsilon H(\mathbf{p}). \end{array} \tag{24}$$

Alternatively, we can set the number of groups and minimize the grouping redundancy

$$\begin{array}{ll} \text{Minimize}_{\mathcal{P}} & \ell(\mathcal{P}, \mathbf{p}) \\ \text{subject to} & |\mathcal{P}| = N_g. \end{array} \tag{25}$$

It is interesting to note that, since the entropy of the source is the same for all partitions, i.e., $\ell(\mathcal{P}, \mathbf{p}) = R(\mathcal{P}, \mathbf{p}) - H(\mathbf{p})$, we can use the bit rate $R(\mathcal{P}, \mathbf{p})$ instead of the redundancy $\ell(\mathcal{P}, \mathbf{p})$ in the objective function of this optimization problem. However, in our analysis it

is more convenient to use the redundancy because it has properties that are similar to those found in other optimization problems (cf. Section 3.4).

This is a combinatorial problem which has a solution space—all the different partitions of a set—that had been extensively studied [8]. For instance, the total number partitions of an N_s-symbol alphabet (set) into N_g groups (nonempty subsets), $\Xi_g(N_s, N_g)$, is equal to the Stirling Number of the Second Kind

$$\Xi_g(N_s, N_g) = \mathfrak{S}(N_s, N_g) = \sum_{k=0}^{N_g} \frac{k^{N_s}(-1)^{N_g-k}}{k!(N_g-k)!}, \tag{26}$$

which is a number that grows very fast with N_s and N_g.

Finding the optimal partition in the most general cases can be an extremely demanding task due to its combinatorial nature. There are many other important practical problems that can be formulated as optimal set-partitioning, and there is extensive research on efficient methods to solve them [4, 28]. On one hand, our alphabet partitioning problem is somewhat more complicated because its objective function is nonlinear, and can only be computed when the group sizes and probabilities are fully defined. For example, it is not possible to know the cost of assigning a symbol to a group without knowing all the other symbols in the group. On the other hand, the alphabet partitioning has many useful mathematical properties that allow more efficient solution methods, and which we are going to explore next.

Given the original order of the data symbols, we can constrain the partitions to include only adjacent symbols. For example, we allow partitioning $\{1, 2, 3, 4\}$ as $\{\{1, 2\}, \{3, 4\}\}$, but not as $\{\{1, 3\}, \{2, 4\}\}$. This way the N_g symbol groups can be defined by the strictly increasing sequence of $N_g + 1$ thresholds $\mathcal{T} = (t_0 = 1, t_1, t_2, \ldots, t_{N_g} = N_s + 1)$, such that

$$\mathcal{G}_n = \{s : \ t_{n-1} \leq s < t_n\}, \quad n = 1, 2, \ldots, N_g. \tag{27}$$

In this case, the redundancy resulting from grouping (14) can be computed as

$$\ell(\mathcal{T}, \mathbf{p}) = \sum_{n=1}^{N_g} \sum_{s=t_{n-1}}^{t_n-1} p_s \log_2 \left(\frac{p_s}{\bar{p}_n} \right), \tag{28}$$

where

$$\bar{p}_n = \frac{1}{t_n - t_{n-1}} \sum_{s=t_{n-1}}^{t_n-1} p_s, \quad n = 1, 2, \ldots, N_g. \tag{29}$$

The new set partitioning problem, including these constraints, corresponds to finding the optimal *linear partition*, which is also a well known optimization problem [20]. The number of possible linear partitions is

$$\Xi_l(N_s, N_g) = C(N_s - 1, N_g - 1) = \frac{(N_s - 1)!}{(N_g - 1)!(N_s - N_g)!}, \tag{30}$$

which also grows fast, but not as fast as $\mathfrak{S}(N_s, N_g)$. For instance, we have $\Xi_g(50, 6) \approx 10^{36}$, while $\Xi_l(50, 6) \approx 2 \cdot 10^6$.

The determination of optimal linear partitions is a computational problem that is relatively much easier to solve then the general set partitioning problem. In fact, it has long been demonstrated that many of these partition problems can be solved with polynomial complexity via dynamic programming [10, 20].

The number of possible linear partitions is further greatly reduced when we consider only dyadic linear partitions, i.e., we add the constraint that the number of symbols in each group must be a power of two. In this case, the total number of solutions, which we denote by $\Xi_d(N_s, N_g)$, is not as well known as the other cases. However, for our coding problem it is important to study some of its properties, given that it corresponds to the most useful practical problems, and because it has some unusual characteristics that can affect the usefulness of the solutions.

For instance, the fact that $\Xi_d(50, 6) = 630$ shows that the number of possible dyadic partitions is much smaller, and in fact, we may even consider enumerating and testing all possibilities. However, when we try other values, we find cases like $\Xi_d(63, 5) = 0$, while $\Xi_d(64, 5) = 75$. This shows that with dyadic partitions we may actually have too few options, and may need to extend the alphabet, adding symbols with zero probability, in order to find out better alphabet partitions for coding.

Appendix A contains some analysis of the total number of dyadic partitions, and a C++ code showing that functions for enumerating of all possible linear partitions (including the option for dyadic constraints) can be quite short and simple. This code proved to be very convenient, since it is good to perform a few exhaustive searches to be sure that the implementations of the faster techniques are correct.

3.2 A Necessary Optimality Condition

From the complexity analysis above, it is clearly advantageous to consider only linear partitions of source alphabets. In addition, an intuitive understanding of the symbol grouping problem enabled us to see the necessity of sorting symbols according to their probability [18]. However, we could only determine if an alphabet partition is sufficiently good, but not know if it is optimal. In this section we present the theory that allows us to ascertain optimality.

The following theorem proves that among all possible alphabet partitions, only those that correspond to linear partitions on symbols sorted according to probability can be optimal. Thus, this necessary optimality condition guarantees that nothing is lost if we solve this easier partitioning problem. In Section 3.3 we show how to use this result to find optimal solutions, in polynomial time, using dynamic programming.

Theorem 3.1 *A partition $\mathcal{P}^*$ is optimal only if for each group $\mathcal{G}_n$, $n = 1, 2, \ldots, N_g$ there is*

no symbol $s \in \mathcal{A} - \mathcal{G}_n$ *such that*

$$\min_{i \in \mathcal{G}_n} \{p_i\} < p_s < \max_{i \in \mathcal{G}_n} \{p_i\} . \tag{31}$$

Proof: Assume $\mathcal{P}^*$ is the optimal partition. We show that, unless $\mathcal{P}^*$ satisfies the condition of Theorem 3.1, we can define another partition with smaller loss, contradicting the assumption that $\mathcal{P}^*$ is optimal.

Without loss of generality we consider only the first two groups in the partition $\mathcal{P}^*$, namely, $\mathcal{G}_1$ and $\mathcal{G}_2$. We define two other partitions $\mathcal{P}'$ and $\mathcal{P}''$ which are identical to $\mathcal{P}^*$ except for the first two groups. These first two groups of $\mathcal{P}'$ and $\mathcal{P}''$ are represented respectively, as $\mathcal{G}'_1$ and $\mathcal{G}'_2$, and as $\mathcal{G}''_1$ and $\mathcal{G}''_2$, and they have the following properties

$$\begin{array}{l} \mathcal{G}_1 \cup \mathcal{G}_2 = \mathcal{G}'_1 \cup \mathcal{G}'_2 = \mathcal{G}''_1 \cup \mathcal{G}''_2, \\ |\mathcal{G}_1| = |\mathcal{G}'_1| = |\mathcal{G}''_1|. \end{array} \tag{32}$$

In addition, the groups are defined such that $\mathcal{G}'_1$ contains the symbols with smallest probabilities, and the group $\mathcal{G}''_1$ contains the symbols with largest probabilities, i.e.,

$$\begin{array}{l} \max_{s \in \mathcal{G}'_1} \{p_s\} \leq \min_{s \in \mathcal{G}'_2} \{p_s\} , \\ \min_{s \in \mathcal{G}''_1} \{p_s\} \geq \max_{s \in \mathcal{G}''_2} \{p_s\} . \end{array} \tag{33}$$

The analysis of these partitions is simplified if we normalize probabilities. First, we define

$$\sigma = \sum_{s \in \mathcal{G}_1 \cup \mathcal{G}_2} p_s = \rho_1 + \rho_2 > 0.$$

We do not have to analyze the case $\sigma = 0$, since it implies that *all* symbols in $\mathcal{G}_1 \cup \mathcal{G}_2$ have zero probability, and that consequently these two groups have no redundancy. Assuming that $\sigma > 0$ we can define

$$f_1 = \rho_1/\sigma, \quad f_2 = \rho_2/\sigma, \quad f'_1 = \rho'_1/\sigma, \quad f'_2 = \rho'_2/\sigma, \quad f''_1 = \rho''_1/\sigma, \quad f''_2 = \rho''_2/\sigma.$$

Note that (32) implies that $f_1 + f_2 = f'_1 + f'_2 = f''_1 + f''_2 = 1$.

The difference between the coding losses of the different partitions is computed using (10)

$$\begin{aligned} \varphi' &= \ell(\mathcal{P}^*, \mathbf{p}) - \ell(\mathcal{P}', \mathbf{p}) = R(\mathcal{P}^*, \mathbf{p}) - R(\mathcal{P}', \mathbf{p}) \\ &= \rho_1 \log_2 \left(\frac{|\mathcal{G}_1|}{\rho_1} \right) + \rho_2 \log_2 \left(\frac{|\mathcal{G}_2|}{\rho_2} \right) - \rho'_1 \log_2 \left(\frac{|\mathcal{G}_1|}{\rho'_1} \right) - \rho'_2 \log_2 \left(\frac{|\mathcal{G}_2|}{\rho'_2} \right) \\ &= \sigma \left[f_1 \log_2 \left(\frac{|\mathcal{G}_1|}{\sigma f_1} \right) + f_2 \log_2 \left(\frac{|\mathcal{G}_2|}{\sigma f_2} \right) - f'_1 \log_2 \left(\frac{|\mathcal{G}_1|}{\sigma f'_1} \right) - f'_2 \log_2 \left(\frac{|\mathcal{G}_2|}{\sigma f'_2} \right) \right] \end{aligned}$$

Using the binary entropy function

$$H_2(p) = -p\log_2(p) - (1-p)\log_2(1-p), \tag{34}$$

and the fact that $f_1 - f_1' = f_2' - f_2$, we obtain

$$\varphi' = \sigma\left[H_2(f_1) - H_2(f_1') + (f_1 - f_1')\log_2\left(\frac{|\,\mathcal{G}_1|}{|\,\mathcal{G}_2|}\right)\right], \tag{35}$$

and, in a similar manner, obtain

$$\begin{aligned}\varphi'' &= \ell(\mathcal{P}^*, \mathbf{p}) - \ell(\mathcal{P}'', \mathbf{p}) \\ &= \sigma\left[H_2(f_1) - H_2(f_1'') + (f_1'' - f_1)\log_2\left(\frac{|\,\mathcal{G}_2|}{|\,\mathcal{G}_1|}\right)\right].\end{aligned} \tag{36}$$

Next we show that, because sorting the probabilities as defined by (33) guarantees that $f_1' \leq f_1 \leq f_1''$, we always have either $\varphi' \geq 0$ or $\varphi'' \geq 0$, with equality only if either $f_1 = f_1'$ or $f_1 = f_1''$.

We use the fact that the binary entropy $H_2(p)$ is a concave function, which means that

$$H_2(x) \leq H_2(f_1) + (x - f_1)\left.\frac{dH_2(p)}{dp}\right|_{p=f_1} = H_2(f_1) + (x - f_1)\log_2\left(\frac{1-f_1}{f_1}\right), \tag{37}$$

as shown in Fig. 5.

The substitution of (37) into (35) and into (36), with $x = f_1'$ and $x = f_1''$, respectively, results in

$$\begin{aligned}\varphi' &\geq \sigma\left[(f_1 - f_1')\log_2\left(\frac{|\,\mathcal{G}_1|(1-f_1)}{|\,\mathcal{G}_2|f_1}\right)\right], \\ \varphi'' &\geq \sigma\left[(f_1'' - f_1)\log_2\left(\frac{|\,\mathcal{G}_2|f_1}{|\,\mathcal{G}_1|(1-f_1)}\right)\right],\end{aligned}$$

which means that $\varphi' \geq 0$ whenever $f_1 \leq |\,\mathcal{G}_1|/(|\,\mathcal{G}_1| + |\,\mathcal{G}_2|)$, and $\varphi'' \geq 0$ whenever $f_1 \geq |\,\mathcal{G}_1|/(|\,\mathcal{G}_1| + |\,\mathcal{G}_2|)$.

Since having $\varphi' > 0$ or $\varphi'' > 0$ means that $\mathcal{P}^*$ is not optimal, the conclusion is that $\mathcal{P}^*$ can only be optimal if $f_1 = f_1'$ or if $f_1 = f_1''$, i.e, the first two groups of $\mathcal{P}^*$ must be defined in one of the two ways that satisfy the ordering of probabilities, as defined by (33). Since this fact was proved without any constraints on the groups $\mathcal{G}_1$ and $\mathcal{G}_2$, it is thus valid for any pair of groups $\mathcal{G}_m$ and $\mathcal{G}_n$, and we reach the conclusion that only the partitions that satisfy the condition of the theorem can be optimal ∎

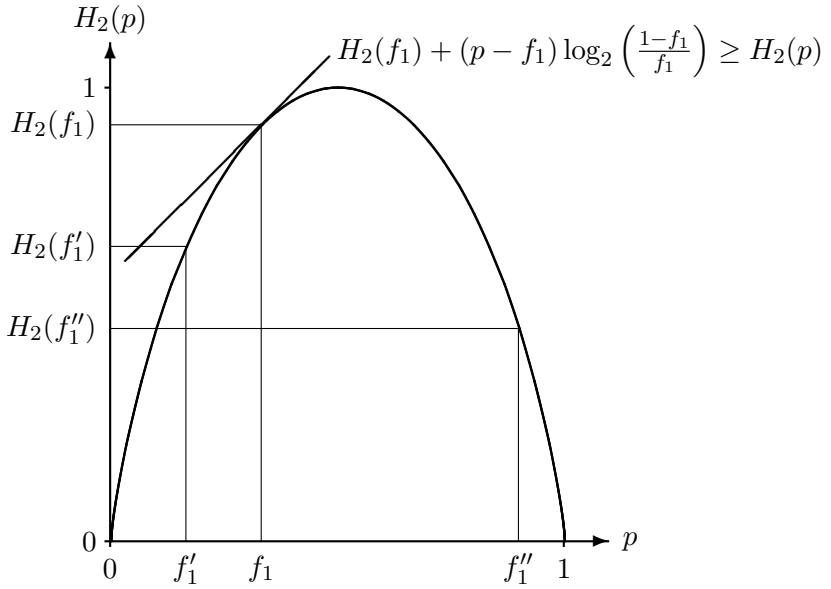


Figure 5: Bound for the grouping redundancy exploiting the fact that the binary entropy is a concave function.

3.3 Optimal Partitions via Dynamic Programming

Since Theorem 3.1 let us solve the alphabet partition problem (25) as a simpler linear partition problem, we first need to sort the N_s data symbols according to their probability, which can be done with $\mathrm{O}(N_s \log N_s)$ complexity [10]. Many of the results that follow depend only on the probabilities being monotonic, i.e., they are valid for both non-increasing and non-decreasing sequences. To simplify notation we assume that all the symbols had been renumbered after sorting. For instance, if we have non-increasing symbol probabilities, then $p_1 \geq p_2 \geq p_3 \geq \cdots \geq p_{N_s}$.

Next, we need to define some new notation. We assume that the *optimal* linear partition of the reduced alphabet with the first j symbols into i groups has redundancy equal to $\ell^*_{i,j}$, and represent the redundancy resulting from having the symbols $i, i+1, \ldots, j$ in a single group as

$$\gamma_{i,j} = \sum_{s=i}^{j} p_s \log_2\left(\frac{[j-i+1]p_s}{\rho_{i,j}}\right), \quad 1 \leq i \leq j \leq N_s, \tag{38}$$

where

$$\rho_{i,j} = \sum_{s=i}^{j} p_s. \tag{39}$$

Note that we always have $\gamma_{i,i} = 0$.

The dynamic programming solution is based on the following theorem.

Theorem 3.2 *Given a source alphabet $\mathcal{A}$ with N_s symbols and monotonic symbol probabilities* $\mathbf{p}$*, the set of all minimal grouping redundancies $\ell^*_{i,j}$, can be recursively computed using the initial values defined by*

$$\ell^*_{1,j} = \gamma_{1,j}, \quad 1 \leq j \leq N_s. \tag{40}$$

followed by

$$\ell^*_{i,j} = \min_{i \leq k \leq j} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,j} \right\}, \quad 2 \leq i \leq j \leq N_s, \tag{41}$$

Proof: The justification of the initial values (40) is quite simple: there is only one way to partition into one group, and the optimal solution value is the resulting redundancy. The rest of the proof is done by induction.

Given a sub-alphabet with the first j symbols, and a number of groups i, let us assume that all the values of $\ell^*_{i,k}$, $i \leq k \leq j$, are known. If we add the symbol $j+1$, and allow an extra group to include it, we know from Theorem 3.1 that, if the symbol probabilities are monotonic, then the last group of the new optimal partition (including symbol $j+1$) is in the form $\mathcal{G}_{i+1} = \{k+1, k+2, \ldots, j+1\}$. Our problem now is to find the optimal value of k, which can be done by finding the minimum redundancy among all possible choices, i.e., computing the minimum redundancy corresponding to all values $k = i, i+1, \ldots, j$. Using the redundancy equation (28), defined for linear partitions, we have

$$\begin{aligned}
\ell^*_{i+1,j+1} &= \min_{t_0=1<t_1<\cdots<t_{i+1}=j+2} \left\{ \sum_{n=1}^{i+1} \sum_{s=t_{n-1}}^{t_n-1} p_s \log_2 \left(\frac{p_s}{\bar{p}_n} \right) \right\} \\
&= \min_{i \leq k \leq j} \left\{ \gamma_{k+1,j+1} + \min_{t_0=1<t_1<\cdots<t_i=k+1} \left\{ \sum_{n=1}^{i} \sum_{s=t_{n-1}}^{t_n-1} p_s \log_2 \left(\frac{p_s}{\bar{p}_n} \right) \right\} \right\} \\
&= \min_{i \leq k \leq j} \left\{ \gamma_{k+1,j+1} + \ell^*_{i,k} \right\}.
\end{aligned}$$

The desired result (41) is obtained by simply changing the indexes.

Note that we used the fact the redundancy can be divided in the sum of two factors that are independent, since the total redundancy of the i groups containing the first k symbols does not depend on the redundancy of the last group. The conclusion is that the computed value of $\ell^*_{i,j}$ is indeed optimal because it is found by implicitly testing all possible linear partitions.

The last fact to consider is that, if the values of $\ell^*_{i,j}$ are computed following the sequence $i = 1, 2, \ldots, N_g$, and for each i the sequence $j = i, i+1, \ldots, N_s$, then all the values required for the computation of $\ell^*_{i,j}$ using (41) are already known. ∎

Fig. 6 shows an example of how the values of $\ell^*_{i,j}$ can be arranged in a two-dimensional matrix. In the figure we show the values needed for the computation

$$\ell^*_{3,6} = \min \left\{ \ell^*_{2,2} + \gamma_{3,6},\ \ell^*_{2,3} + \gamma_{4,6},\ \ell^*_{2,4} + \gamma_{5,6},\ \ell^*_{2,5} + \gamma_{6,6} \right\}.$$

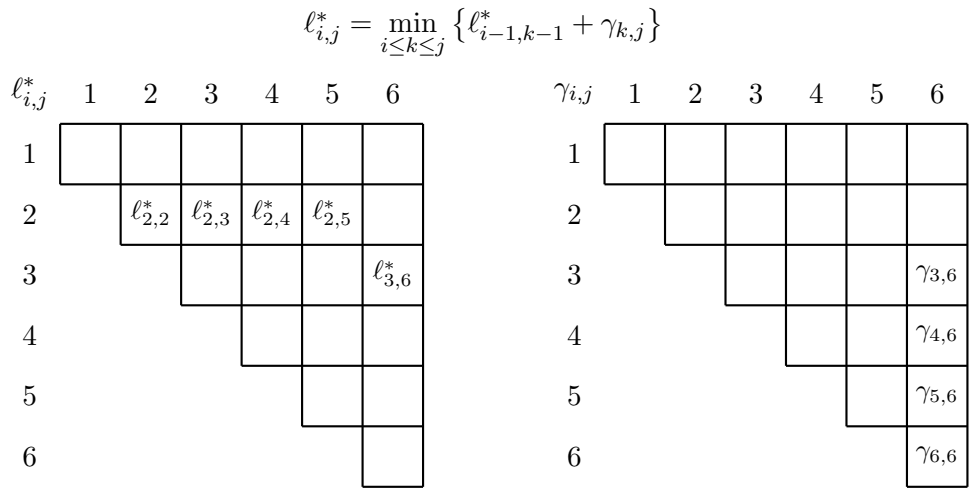


Figure 6: Diagram of the dynamic programming recursion to find optimal linear partitions: $\ell^*_{i,j}$ is the minimum redundancy due to grouping the first j symbols into i groups; $\gamma_{i,j}$ is the redundancy occured by having symbols i to j in a single group. The figure shows the data required for computing $\ell^*_{3,6} = \min\left\{\ell^*_{2,2} + \gamma_{3,6},\ \ell^*_{2,3} + \gamma_{4,6},\ \ell^*_{2,4} + \gamma_{5,6},\ \ell^*_{2,5} + \gamma_{6,6}\right\}$

On the right side we show the matrix with values of $\ell^*_{i,j}$. Note that the elements in a row only need the values in the previous row for their computation. In the left side of Fig. 6 we show the matrix with values of $\gamma_{i,j}$, and how the minimum redundancy is computed using a sequence of values in the same column.

The recursion (41) provides only the value of the optimal solutions, not the optimal group sizes. They can be recovered if we store all the values of index k that corresponds to the minimum in (41) when the value of $\ell^*_{i,j}$ is computed, and which we represent as

$$\kappa^*(i,j) = \begin{cases} 1, & 1 = i \le j \le N_s, \\ \min\left\{k : \ell^*_{i,j} = \ell^*_{i-1,k-1} + \gamma_{k,j},\ i \le k \le j\right\}, & 2 \le i \le j \le N_s. \end{cases} \tag{42}$$

We choose this specific definition because it makes the value of $\kappa^*(i,j)$ unique, simplifying the presentation of some of its properties.

The optimal group sizes are computed by backtracking the optimal decisions. For example, the optimal partition into N_g groups is defined by

$$\begin{array}{rcll} t_{N_g} & = & N_s + 1, & \\ t_{n-1} & = & \kappa^*(n, t_n - 1), & n = N_g, N_g - 1, \ldots, 1, \\ |\,\mathcal{G}_n| & = & t_n - t_{n-1}, & n = N_g, N_g - 1, \ldots, 1. \end{array} \tag{43}$$

3.4 Mathematical Properties

The results in Section 3.3 already show that the optimal alphabet partitions can be computed using much less computational effort than exhaustive search. In fact, in the end we find *all* the optimal partitions of the alphabet. In Section 4 we analyze the computational implementation and its complexity. In this section we show that $\gamma_{i,j}$ and $\ell^*_{i,j}$ have a set of mathematical properties that are useful for further reduction of computational complexity, enabling more efficient methods for computing (41). In addition, they are useful for identifying features of optimal alphabet partitions.

Theorem 3.3 *For any set of symbol probabilities* $\mathbf{p}$*, the redundancy* $\gamma_{i,j}$ *resulting from creating a group with symbols* $i, i+1, \ldots, j$*, is such that*

$$0 \leq \gamma_{i,j} \leq \rho_{i,j} \log_2(j-i+1). \tag{44}$$

Proof: We can rewrite (38) as

$$\gamma_{i,j} = \rho_{i,j} \log_2 \left(\frac{j-i+1}{\rho_{i,j}} \right) + \sum_{s=i}^{j} p_s \log_2(p_s). \tag{45}$$

Using (20) and (45) we obtain

$$\begin{aligned} \gamma_{i,j} &= \rho_{i,j} \left[\log_2(j-i+1) + \sum_{s=i}^{j} \frac{p_s}{\rho_{i,j}} \log_2 \left(\frac{p_s}{\rho_{i,j}} \right) \right] \\ &= \rho_{i,j} \left[\log_2(j-i+1) - H(\mathbf{q}) \right] \end{aligned}$$

where $q_{k+1} = p_{i+k}/\rho_{i,j}$, $k = 0, 1, \ldots, j-i$, and $H(\mathbf{q})$ is its entropy. The desired inequalities follow from the fact that $0 \leq H(\mathbf{q}) \leq \log_2(j-i+1)$. ∎

Theorem 3.4 *For any set of symbol probabilities* $\mathbf{p}$*, the redundancy* $\gamma_{i,j}$ *resulting from creating a group with symbols* $i, i+1, \ldots, j$*, is such that, for all integer values of* i, j *and* k *satisfying* $1 \leq i \leq k < j \leq N_s$*, we have*

$$\gamma_{i,j} \geq \gamma_{i,k} + \gamma_{k+1,j}. \tag{46}$$

Proof: Let us define

$$n_a = k - i + 1, \quad n_c = j - k,$$

and

$$f_a = \rho_{i,k}/n_a, \quad f_b = \rho_{i,j}/(n_a + n_c), \quad f_c = \rho_{k+1,j}/n_c.$$

From definition (39) we can see that $\rho_{i,j} = \rho_{i,k} + \rho_{k+1,j}$, and consequently we can write f_b as the following linear combination of f_a and f_c

$$f_b = \alpha f_a + (1-\alpha) f_c \quad \Rightarrow \quad \alpha = \frac{n_a}{n_a + n_c}, \quad 1 - \alpha = \frac{n_c}{n_a + n_c}.$$

Note that this result shows that the average probabilities are such that f_b is between f_a and f_c, i.e.,

$$\min\{f_a, f_c\} \leq f_b \leq \max\{f_a, f_c\}.$$

From (45) we obtain

$$\begin{aligned}
\gamma_{i,j} - \gamma_{i,k} - \gamma_{k+1,j} &= \rho_{i,j} \log_2\left(\frac{j-i+1}{\rho_{i,j}}\right) - \rho_{i,k} \log_2\left(\frac{k-i+1}{\rho_{i,k}}\right) - \rho_{k+1,j} \log_2\left(\frac{j-k}{\rho_{k+1,j}}\right) \\
&= \rho_{i,j} \log_2\left(\frac{1}{f_b}\right) - \rho_{i,k} \log_2\left(\frac{1}{f_a}\right) - \rho_{k+1,j} \log_2\left(\frac{1}{f_c}\right) \\
&= (n_a + n_c) f_b \log_2\left(\frac{1}{f_b}\right) - n_a f_a \log_2\left(\frac{1}{f_a}\right) - n_c f_c \log_2\left(\frac{1}{f_c}\right).
\end{aligned}$$

Since the function $p \log_2(1/p)$ is concave, and since $0 \leq \alpha \leq 1$, we can obtain a lower bound on $f_b \log_2(1/f_b)$ as shown in Fig. 7, which results in the inequality

$$(n_a + n_c) f_b \log_2\left(\frac{1}{f_b}\right) \geq (n_a + n_c)\left[\alpha f_a \log_2\left(\frac{1}{f_a}\right) + (1-\alpha) f_c \log_2\left(\frac{1}{f_c}\right)\right].$$

The substitution of the value of α yields the inequality

$$(n_a + n_c) f_b \log_2\left(\frac{1}{f_b}\right) \geq n_a f_a \log_2\left(\frac{1}{f_a}\right) + n_c f_c \log_2\left(\frac{1}{f_c}\right),$$

which produces the desired result

$$\gamma_{i,j} - \gamma_{i,k} - \gamma_{k+1,j} \geq 0.$$

■

Theorem 3.5 *For any set of symbol probabilities* $\mathbf{p}$*, the redundancy* $\gamma_{i,j}$ *resulting from creating a group with symbols* $i, i+1, \ldots, j$*, is such that, for all integer values of* i, j, k *and* l *satisfying* $1 \leq i \leq k \leq l \leq j \leq N_s$*, we have*

$$\gamma_{k,l} \leq \gamma_{i,j}. \tag{47}$$

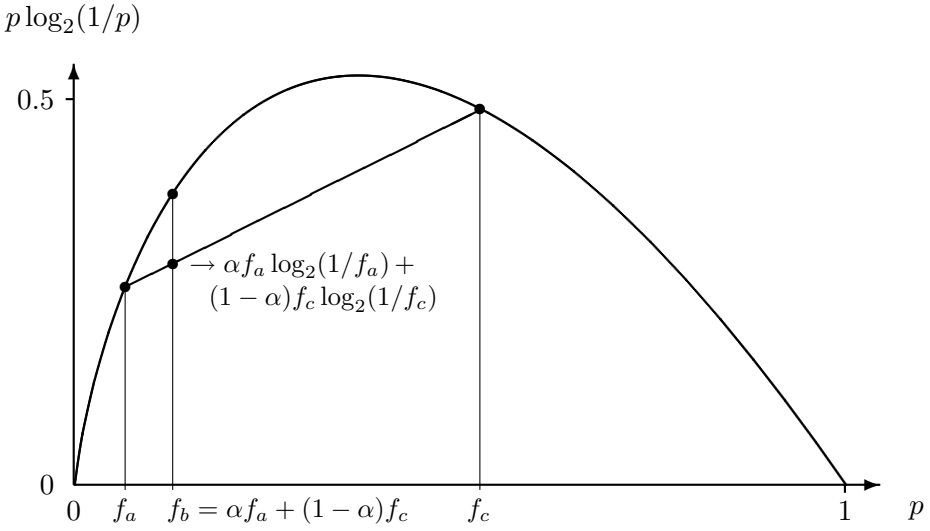


Figure 7: Use of the concavity of function $p\log_2(1/p)$ for obtaining lower bounds on redundancy.

Proof: From Theorem 3.4, and from the fact that we always have $\gamma_{j,j} = 0$, we conclude that for all values of $1 \leq i < j \leq N_s$ we have

$$\gamma_{i,j-1} = \gamma_{i,j-1} + \gamma_{j,j} \leq \gamma_{i,j}, \tag{48}$$

and, similarly, for all $1 < i \leq j \leq N_s$

$$\gamma_{i+1,j} = \gamma_{i,i} + \gamma_{i+1,j} \leq \gamma_{i,j}. \tag{49}$$

The general result (47) is obtained by induction, with the sequential use of (48) followed by (49)

$$\gamma_{k,l} \leq \gamma_{k,l+1} \leq \cdots \leq \gamma_{k,j-1} \leq \gamma_{k,j} \leq \gamma_{k-1,j} \leq \cdots \leq \gamma_{i+1,j} \leq \gamma_{i,j},$$

or (49) followed by (48)

$$\gamma_{k,l} \leq \gamma_{k-1,l} \leq \cdots \leq \gamma_{i+1,l} \leq \gamma_{i,l} \leq \gamma_{i,l+1} \leq \cdots \leq \gamma_{i,j-1} \leq \gamma_{i,j}.$$

■

Theorem 3.6 *For any* monotonic *set of symbol probabilities $p_1, p_2, \ldots, p_{N_s}$, the redundancy $\gamma_{i,j}$ resulting from creating a group with symbols $i, i+1, \ldots, j$, is such that, for all integer values of i, j, k and l satisfying $1 \leq i \leq k \leq l \leq j \leq N_s$, we have*

$$\gamma_{i,j} + \gamma_{k,l} \geq \gamma_{i,l} + \gamma_{k,j}. \tag{50}$$

Proof: Let us define

$$n_a = j - i + 1, \quad n_b = l - k + 1, \quad n_c = l - i + 1, \quad n_d = j - k + 1,$$

and

$$f_a = \rho_{i,j}/n_a, \quad f_b = \rho_{k,l}/n_b, \quad f_c = \rho_{i,l}/n_c, \quad f_d = \rho_{k,j}/n_d.$$

We can write f_a and f_b as linear combinations of f_c and f_d, as

$$\begin{aligned} f_a = \alpha f_c + (1-\alpha) f_d \quad &\Rightarrow \quad \alpha = \frac{f_a - f_d}{f_c - f_d}, \quad 1 - \alpha = \frac{f_c - f_a}{f_c - f_d}, \\ f_b = \beta f_c + (1-\beta) f_d \quad &\Rightarrow \quad \beta = \frac{f_b - f_d}{f_c - f_d}, \quad 1 - \beta = \frac{f_c - f_b}{f_c - f_d}. \end{aligned}$$

Because the sequence of probabilities is monotonic we have

$$\min\{f_c, f_d\} \leq \min\{f_a, f_b\} \leq \max\{f_a, f_b\} \leq \max\{f_c, f_d\},$$

which means that $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$.

From (45) we have

$$\gamma_{i,j} + \gamma_{k,l} - \gamma_{i,l} - \gamma_{k,j} = n_a f_a \log_2\left(\frac{1}{f_a}\right) + n_b f_b \log_2\left(\frac{1}{f_b}\right) - n_c f_c \log_2\left(\frac{1}{f_c}\right) - n_d f_d \log_2\left(\frac{1}{f_d}\right).$$

Using again the lower bound shown in Fig. 7, we obtain

$$\gamma_{i,j} + \gamma_{k,l} - \gamma_{i,l} - \gamma_{k,j} \geq [\alpha n_a + \beta n_b - n_c] f_c \log_2\left(\frac{1}{f_c}\right) + [(1-\alpha) n_a + (1-\beta) n_b - n_d] f_d \log_2\left(\frac{1}{f_d}\right).$$

From the fact that $n_a + n_b = n_c + n_d$, and the substitution of the values of α and β

$$\begin{aligned} \alpha n_a + \beta n_b - n_c &= \frac{(f_a - f_d) n_a + (f_b - f_d) n_b + (f_d - f_c) n_c}{f_c - f_d} \\ &= \frac{\rho_{i,j} + \rho_{k,l} - \rho_{i,l} - \rho_{k,j}}{f_c - f_d} = 0. \end{aligned}$$

A similar calculation shows that $(1-\alpha) n_a + (1-\beta) n_b - n_d = 0$, and consequently

$$\gamma_{i,j} + \gamma_{k,l} - \gamma_{i,l} - \gamma_{k,j} \geq 0.$$

■

Note that Theorems 3.4 and 3.5 are valid for any set of probabilities, while Theorem 3.6 is valid only for monotonic probabilities. In fact, Theorem 3.6 is enough to guarantee all the desired properties when the probabilities are sorted.

Theorem 3.5 shows that the matrix containing the values of $\gamma_{i,j}$, assuming $\gamma_{i,j} = 0$ if $j < i$, is a *monotonic matrix*, and Theorem 3.6 shows that its elements satisfy the *quadrangle inequalities*, i.e., it is a *Monge matrix*. Both these properties had been shown to be useful for designing faster algorithms for sorting and for dynamic programming (cf. Section 4). The minimum redundancy values $\ell^*_{i,j}$ have similar properties, as shown below.

Theorem 3.7 *For any set of monotonic symbol probabilities* $\mathbf{p}$*, the minimum redundancy* $\ell^*_{i,j}$*, resulting from grouping the first* j *symbols in* i *groups, is such that, for all integer values of* i *and* j *we have, when* $1 \leq i < j \leq N_s$

$$\ell^*_{i+1,j} \leq \ell^*_{i,j}, \tag{51}$$

when $1 \leq i \leq j < N_s$

$$\ell^*_{i,j} \leq \ell^*_{i,j+1}, \tag{52}$$

and when $1 < i < j < N_s$

$$\ell^*_{i+1,j} \leq \ell^*_{i+1,j+1} \leq \ell^*_{i,j} \leq \ell^*_{i,j+1}. \tag{53}$$

Proof: Let us first prove (51). Since $i < j$, the optimal partition corresponding to $\ell^*_{i,j}$ has at least one group such that $|\mathcal{G}_n| > 1$. From Theorem 3.4 we know that the redundancy cannot increase when we divide a group in two, increasing the number of groups by one. Thus, since $\ell^*_{i+1,j}$ cannot be larger than the redundancy of this new partition, we conclude that $\ell^*_{i,j} \geq \ell^*_{i+1,j}$. Note that this is also valid for dyadic partitions, since any dyadic group with more than one element can always be divided in two dyadic groups.

Inequality (52) can be proved by induction. It is valid in the first row because, from (40) and Theorem 3.5, we have

$$\ell^*_{1,j+1} = \gamma_{1,j+1} \geq \gamma_{1,j} = \ell^*_{1,j}.$$

The cases when $i > 1$ follow from (41), the fact that $\gamma_{j,j} = 0$, $\gamma_{k,j+1} \geq \gamma_{k,j}$, and (51):

$$\begin{aligned}
\ell^*_{i,j+1} &= \min\left\{\ell^*_{i-1,j},\ \min_{i\leq k\leq j}\left\{\ell^*_{i-1,k-1} + \gamma_{k,j+1}\right\}\right\} \\
&\geq \min\left\{\ell^*_{i-1,j},\ \min_{i\leq k\leq j}\left\{\ell^*_{i-1,k-1} + \gamma_{k,j}\right\}\right\} \\
&\geq \min\left\{\ell^*_{i-1,j},\ \ell^*_{i,j}\right\} = \ell^*_{i,j}.
\end{aligned}$$

The next inequality follows from (41)

$$\ell^*_{i+1,j+1} = \min_{i\leq k\leq j}\left\{\ell^*_{i,k} + \gamma_{k+1,j+1}\right\} \leq \ell^*_{i,j} + \gamma_{j+1,j+1} = \ell^*_{i,j}.$$

∎

Corollary 3.8 *For any set of monotonic symbol probabilities* $\mathbf{p}$*, the minimum redundancy resulting from grouping the first* j *symbols in* i *groups,* $\ell^*_{i,j}$*, is such that, for all integer values of* i, j, k *and* l *satisfying* $1 \leq i \leq k \leq l \leq j \leq N_s$*, we have*

$$\ell^*_{k,l} \leq \ell^*_{i,j}. \tag{54}$$

Proof: This result follows from Theorem 3.7, and two chains of inequalities

$$\ell^*_{k,l} \leq \ell^*_{k,l+1} \leq \cdots \leq \ell^*_{k,j-1} \leq \ell^*_{k,j} \leq \ell^*_{k-1,j} \leq \cdots \leq \ell^*_{i+1,j} \leq \ell^*_{i,j},$$

$$\ell^*_{k,l} \leq \ell^*_{k-1,l} \leq \cdots \leq \ell^*_{i+1,l} \leq \ell^*_{i,l} \leq \ell^*_{i,l+1} \leq \cdots \leq \ell^*_{i,j-1} \leq \ell^*_{i,j}.$$

∎

Theorem 3.9 *For any set of monotonic symbol probabilities* $\mathbf{p}$*, the minimum redundancy* $\ell^*_{i,j}$*, resulting from grouping the first* j *symbols in* i *groups, is such that, for all integer values of* i, j, k *and* l *satisfying* $1 \leq i \leq k \leq l \leq j \leq N_s$*, we have*

$$\ell^*_{i,j} + \ell^*_{k,l} \geq \ell^*_{i,l} + \ell^*_{k,j}. \tag{55}$$

Proof: Our proof uses two stages of induction. First, we prove the result for pairs of consecutive rows, and then generalize to any combination of rows.

Let us begin proving the theorem for the first pair of rows. From Theorem 3.2 we can compute, for all $2 \leq l < j \leq N_s$, the difference

$$\begin{aligned} \ell^*_{1,j} - \ell^*_{1,l} + \ell^*_{2,l} - \ell^*_{2,j} &= \ell^*_{1,j} - \ell^*_{1,l} + \min_{2\leq k\leq l} \left\{\ell^*_{1,k-1} + \gamma_{k,l}\right\} - \min_{2\leq k\leq j} \left\{\ell^*_{1,k-1} + \gamma_{k,j}\right\} \\ &= \gamma_{1,j} - \gamma_{1,l} + \min_{2\leq k\leq l} \left\{\gamma_{1,k-1} + \gamma_{k,l}\right\} - \min_{2\leq k\leq j} \left\{\gamma_{1,k-1} + \gamma_{k,j}\right\} \end{aligned}$$

Let $u = \kappa^*(2, l)$, defined in (42), be the value of k corresponding to the minimum in the computation of $\ell^*_{2,l}$. Since $2 \leq u \leq l < j$, we have the inequality

$$\ell^*_{2,j} = \min_{2\leq k\leq j} \left\{\gamma_{1,k-1} + \gamma_{k,j}\right\} \leq \gamma_{1,u-1} + \gamma_{u,j},$$

and hence

$$\ell^*_{1,j} - \ell^*_{1,l} + \ell^*_{2,l} - \ell^*_{2,j} \geq \gamma_{1,j} - \gamma_{1,l} + \gamma_{1,u-1} + \gamma_{u,l} - \gamma_{1,u-1} - \gamma_{u,j}.$$

From Theorem 3.6 we know that

$$\gamma_{1,j} + \gamma_{u,l} - \gamma_{1,l} - \gamma_{u,j} \geq 0,$$

and consequently, we obtain our first result: if $1 \leq l \leq j \leq N_s$ then

$$\ell^*_{1,j} + \ell^*_{2,l} \geq \ell^*_{1,l} + \ell^*_{2,j}. \tag{56}$$

Now, assume (55) is valid for all values $\ell^*_{i-1,j}$, and $\ell^*_{i,j}$, $j = i, i+1, \ldots, N_s$, i.e., rows $i-1$ and i. We use Theorem 3.6 again to establish that the inequality is valid for values of row $i+1$ ($\ell^*_{i+1,j}$). Given i, for all l and j such that $i < l < j \leq N_s$, we have

$$\begin{aligned} \ell^*_{i,j} + \ell^*_{i+1,l} - \ell^*_{i,l} - \ell^*_{i+1,j} &= \min_{i \leq k \leq j} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,j} \right\} + \min_{i+1 \leq k \leq l} \left\{ \ell^*_{i,k-1} + \gamma_{k,l} \right\} - \\ &- \min_{i \leq k \leq l} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,l} \right\} - \min_{i+1 \leq k \leq j} \left\{ \ell^*_{i,k-1} + \gamma_{k,j} \right\}. \end{aligned} \tag{57}$$

Let us define $u = \kappa^*(i,j)$ and $v = \kappa^*(i+1,l)$. Note that this implies that $i \leq u \leq j$ and $i < v \leq l$. First, we consider the cases in which $u \leq v$. Since $i \leq u \leq v$, and $v \geq i+1$ we have the inequalities

$$\begin{aligned} \ell^*_{i+1,j} &= \min_{i+1 \leq k \leq j} \left\{ \ell^*_{i,k-1} + \gamma_{k,j} \right\} \leq \ell^*_{i,u-1} + \gamma_{u,j}, \\ \ell^*_{i,l} &= \min_{i \leq k \leq l} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,j} \right\} \leq \ell^*_{i-1,v-1} + \gamma_{v,l}. \end{aligned}$$

Therefore, the substitution into (57) produces

$$\begin{aligned} \ell^*_{i,j} + \ell^*_{i+1,l} - \ell^*_{i,l} - \ell^*_{i+1,j} &= \ell^*_{i-1,u-1} + \gamma_{u,j} + \ell^*_{i,v-1} + \gamma_{v,l} - \ell^*_{i,l} - \ell^*_{i+1,j} \\ &\geq \gamma_{u,j} + \gamma_{v,l} - \gamma_{u,l} - \gamma_{v,j}. \end{aligned}$$

Using Theorem 3.6 again we conclude that

$$u \leq v \quad \Rightarrow \quad \ell^*_{i,j} + \ell^*_{i+1,l} \geq \ell^*_{i,l} + \ell^*_{i+1,j}.$$

In the cases in which $u \geq v$, we can use the fact that $i < v \leq u \leq j$, and $v \leq l$, and interchange the use of the indexes for computing upper bounds to obtain

$$\begin{aligned} \ell^*_{i+1,j} &= \min_{i+1 \leq k \leq j} \left\{ \ell^*_{i,k-1} + \gamma_{k,j} \right\} \leq \ell^*_{i,v-1} + \gamma_{v,j}, \\ \ell^*_{i,l} &= \min_{i \leq k \leq l} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,j} \right\} \leq \ell^*_{i-1,u-1} + \gamma_{u,l}. \end{aligned}$$

The substitution into (57) now yields

$$\ell^*_{i,j} + \ell^*_{i+1,l} - \ell^*_{i,l} - \ell^*_{i+1,j} \geq \ell^*_{i-1,u-1} + \ell^*_{i,v-1} - \ell^*_{i,u-1} - \ell^*_{i-1,v-1}.$$

note that the right-hand-side of this inequality contains only elements in the rows $i-1$ and i, which we assume already satisfy (55). Thus, we cover all possible cases, since

$$v \leq u \quad \Rightarrow \quad \ell^*_{i,j} + \ell^*_{i+1,l} \geq \ell^*_{i,l} + \ell^*_{i+1,j}.$$

The conclusion is that the theorem is valid for all pairs of rows (1,2), (2,3), $\ldots, (i, i+1), \ldots (N_s-2, N_s-1)$. The second part is of the proof simply extends the inequalities using consecutive pairs of rows. Thus, for all integer values of i, j, k and l satisfying $1 \leq i \leq k \leq l \leq j \leq N_s$, we have

$$\ell^*_{i,j} - \ell^*_{i,l} \geq \ell^*_{i+1,j} - \ell^*_{i+1,l} \geq \cdots \geq \ell^*_{k-1,j} - \ell^*_{k-1,l} \geq \ell^*_{k,j} - \ell^*_{k,l}.$$

and consequently

$$\ell^*_{i,j} - \ell^*_{i,l} \geq \ell^*_{k,j} - \ell^*_{k,l}.$$

■

Theorem 3.10 *For any set of monotonic symbol probabilities* $\mathbf{p}$ *with minimum grouping redundancy* $\ell^*_{i,j}$*, the values of* $\kappa^*(i,j)$ *(defined by (42)) are such that, for all integer values of* i, j, k *and* l *satisfying* $1 \leq i \leq k \leq l \leq j \leq N_s$*, we have*

$$\kappa^*(i,l) \leq \kappa^*(i,j) \leq \kappa^*(k,j). \tag{58}$$

Proof: Let us consider the values used in the minimization in (41), which are

$$\lambda_{i,n,j} = \ell^*_{i-1,k-1} + \gamma_{k,j}, \quad 2 \leq i \leq n \leq j \leq N_s.$$

From Theorem 3.9 we have, if $i \leq k \leq m \leq n$ then

$$\begin{aligned} \lambda_{i,m,j} - \lambda_{i,n,j} &= \ell^*_{i-1,m-1} + \gamma_{m,j} - \ell^*_{i-1,n-1} - \gamma_{n,j} \\ &\leq \ell^*_{k-1,m-1} + \gamma_{m,j} - \ell^*_{k-1,n-1} - \gamma_{n,j} = \lambda_{k,m,j} - \lambda_{k,n,j} \end{aligned}$$

This results means that

$$\lambda_{i,m,j} \geq \lambda_{i,n,j} \quad \Rightarrow \quad \lambda_{k,m,j} \geq \lambda_{k,n,j}.$$

In addition, for a given value of j we have

$$\ell^*_{i,j} = \min_{i \leq n \leq j} \{\lambda_{i,n,j}\}, \quad \ell^*_{k,j} = \min_{k \leq n \leq j} \{\lambda_{k,n,j}\}.$$

Consequently, if we know $\kappa^*(i,j)$, then we know that

$$\lambda_{i,m,j} \geq \lambda_{i,\kappa^*(i,j),j} \quad \Rightarrow \quad \lambda_{k,m,j} \geq \lambda_{k,\kappa^*(i,j),j}, \quad m = k, \ldots, \kappa^*(i,j),$$

and the conclusion is that $\kappa^*(i,j) \leq \kappa^*(k,j)$.

Similarly, from Theorem 3.6 we know that if $m \leq n \leq l \leq j$ then

$$\begin{aligned} \lambda_{i,n,j} - \lambda_{i,m,j} &= \ell^*_{i-1,n-1} + \gamma_{n,j} - \ell^*_{i-1,m-1} - \gamma_{m,j} \\ &\leq \ell^*_{i-1,n-1} + \gamma_{n,l} - \ell^*_{i-1,m-1} - \gamma_{m,l} = \lambda_{i,n,l} - \lambda_{i,m,l} \end{aligned}$$

$\tilde{\gamma}_{i,j}$	1	2	3	4	5	6
1	0	$\gamma_{1,2}$	∞	$\gamma_{1,4}$	∞	∞
2		0	$\gamma_{2,3}$	∞	$\gamma_{2,5}$	∞
3			0	$\gamma_{3,4}$	∞	$\gamma_{3,6}$
4				0	$\gamma_{4,5}$	∞
5					0	$\gamma_{5,6}$
6						0

Figure 8: Example of modified redundancy values $\tilde{\gamma}_{i,j}$, used for finding the optimal dyadic partitions (groups with a number of symbols equal to a power of two).

and therefore

$$\lambda_{i,n,j} \geq \lambda_{i,m,j} \quad \Rightarrow \quad \lambda_{i,n,l} \geq \lambda_{i,m,l}.$$

For this case, if we know $\kappa^*(i,j)$, then we know that

$$\lambda_{i,n,j} \geq \lambda_{i,\kappa^*(i,j),j} \quad \Rightarrow \quad \lambda_{i,n,l} \geq \lambda_{i,\kappa^*(i,j),l}, \quad n = \kappa^*(i,j), \ldots l,$$

and the conclusion now is that $\kappa^*(i,j) \geq \kappa^*(i,l)$. ∎

3.5 Dyadic Partitions

As explained in Section 2.2, practical applications may require all the groups to have a number of symbols that is a power of two (dyadic groups). In this section we consider which properties are conserved when we add these constraints on group size. We show that many of the strong properties identified for the general case are not valid for dyadic partitions because of the limitations in the number of possible partitions. In fact, in Appendix A we explain why, for pairs of values of N_s and N_g that are not uncommon, there are no possible dyadic partitions.

Below we explain the how the use of dyadic partitions affects the results of the theorems in the previous sections.

1. The result of Theorem 3.1—a necessary optimality condition for the alphabet partitioning problem—does not depend on the size of the symbol groups. Thus, it also applies to the optimal solutions that have limited choices of group sizes, which means that the dyadic groups also must contain symbols ordered by probability.

2. Theorem 3.2 (dynamic programming recursion) is based only on the fact that we look for the optimal *linear* partition, and on the fact that the grouping redundancy can be divided as the sum of two independent terms. Since these are not changed by constraints on group size, the same type of solution can be used. The simplest way to analyze the required modification is to define

$$\tilde{\gamma}_{i,j} = \begin{cases} \gamma_{i,j}, & \text{if } j-i+1 \text{ is a power of 2,} \\ \infty, & \text{otherwise.} \end{cases} \tag{59}$$

   The dynamic programming recursion is the same as in the general case, but using $\tilde{\gamma}_{i,j}$ instead of $\gamma_{i,j}$ in (41), noting that the value '∞' may be represented by a sufficiently large number. Fig. 8 shows an example of a matrix containing values of $\tilde{\gamma}_{i,j}$.

3. The properties of $\gamma_{i,j}$, i.e., theorems 3.4, 3.5, and 3.6 are not valid for all cases, but are valid when all the $\tilde{\gamma}$ values in the inequalities are equal to γ.

4. Only some of the cases covered by Theorem 3.8 are satisfied by optimal dyadic partitions. All results derived from (51) are valid, while those derived from (52) are not. Furthermore, since Theorem 3.2 is valid, we still can use the fact that $\ell^*_{i+1,j+1} \leq \ell^*_{i,j}$.

3.6 Two Numbers of Bits for Symbol Indexes

Using the same number of bits for all symbol indexes in a group has many practical advantages, but it is not the only manner to reduce complexity. For any group size we can use two numbers of bits to code all the possible symbol indexes in a group. In a group with size $|\,\mathcal{G}_n| = m$ we can assign the following number of bits for the symbol with index x

$$\Lambda(x,m) = \begin{cases} \lfloor \log_2 m \rfloor\,, & 1 \leq x \leq 2^{\lceil \log_2 m \rceil} - m, \\ \lceil \log_2 m \rceil\,, & 2^{\lceil \log_2 m \rceil} - m < x \leq m. \end{cases} \tag{60}$$

The main theoretical problem with this assignment of bits is that Theorem 3.1 (requirement for sorted probabilities) does not hold. Since the symbols with larger indexes use more bits, it is clear that the first modification that is required is to sort the symbols in a group in non-increasing order of probabilities. We do not have a proof that the optimal partition is a linear partition on non-increasing probabilities. However, if we want to search the optimal solution only among those partitions, we can use the same type of dynamic programming of Theorem 3.2, only changing the value of the redundancy of single groups from (38) to

$$\bar{\gamma}_{i,j} = \sum_{s=i}^{j} p_s \left[\Lambda(s-i+1, j-i+1) + \log_2 \left(\frac{p_s}{\rho_{i,j}} \right) \right], \quad 1 \leq i \leq j \leq N_s. \tag{61}$$

and replacing γ with $\bar{\gamma}$ in (40) and (41).

Since the number of bits used to code the symbol indexes are approximately equal to $\log_2 |\mathcal{G}_n|$, it is reasonable to assume that the partitions designed with this modification are not too different from those designed with the original values of γ, and in most cases have the same properties.

4 Analysis of the Computational Complexity

In this section we study how to use the theoretical results from the last sections for implementing algorithms for finding optimal alphabet partitions for symbol grouping. We start by demonstrating that we only need to pre-compute two arrays with dimension $N_s + 1$ for the calculation of all values of $\gamma_{i,j}$.

Using (45), if we pre-compute

$$c_j = \sum_{s=1}^{j} p_s, \quad j = 0, 1, \ldots, N_s, \tag{62}$$

$$h_j = \sum_{s=1}^{j} p_s \log_2(p_s), \quad j = 0, 1, \ldots, N_s, \tag{63}$$

then we can compute all values of $\gamma_{i,j}$ using

$$\gamma_{i,j} = (c_j - c_{i-1}) \log_2 \left(\frac{j - i + 1}{c_j - c_{i-1}} \right) + h_j - h_{i-1}. \tag{64}$$

The direct use of dynamic programming equation (refeqEllRecursion) for computing all the optimal alphabet partitions (i.e., from one to N_s groups) requires the calculation of $\mathrm{O}({N_s}^3)$ terms. To recover all the optimal group sizes we need to store all values of $\kappa^*(i, j)$ (cf. (42) and (43)), which needs $\mathrm{O}({N_s}^2)$ memory. On the other hand, if only the values of ℓ^*_{i,N_s} are desired, we can use the fact that recursion (refeqEllRecursion) can be computed using only two rows at a time, i.e., $\mathrm{O}(N_s)$ memory. When we need to compute the optimal solution for only one number of groups, N_g, we still need to compute the first $N_g - 1$ rows of $\ell^*_{i,j}$, so the complexity is $\mathrm{O}(N_g {N_s}^2)$ operations, and $\mathrm{O}(N_g {N_s}^2)$ memory.

Using the mathematical properties of Section 3.4 we can reduce the computational complexity significantly, using a technique similar to that proposed by Yao [6].

Theorem 4.1 *All optimal partitions of an alphabet with N_s symbols can be computed with $O({N_s}^2)$ operations and $O({N_s}^2)$ memory complexity.*

Proof: The memory complexity is defined by the need to store $N_s(N_s + 1)/2$ values of $\ell^*_{i,j}$ and $\kappa^*(i, j)$, $1 \leq i \leq j \leq N_s$.

From Theorem 3.10 we know that

$$\kappa^*(i, j-1) \leq \kappa^*(i, j) \leq \kappa^*(i+1, j),$$

and thus we can rewrite equation (41) as

$$\ell^*_{i,j} = \min_{\kappa^*(i,j-1) \leq k \leq \kappa^*(i+1,j)} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,j} \right\}, \quad 2 \leq i < j \leq N_s. \tag{65}$$

Note that if we compute the values of $\ell^*_{i,j}$ and $\kappa^*(i, j)$ in the order following diagonals, i.e., $(2, j), (3, j+1), (4, j+2), \ldots, (N_s - j + 2, N_s)$, then all the values required for using (65) are already known. Using (65) we need a much smaller number of computations and comparisons to find each $\ell^*_{i,j}$. The total number of comparisons is, for the diagonal starting at column j,

$$\sum_{i=2}^{N_s-j+2} [\kappa^*(i, j) - \kappa^*(i-1, j-1) + 1] = N_s - j + 1 + \kappa^*(N_s - j + 2, j) - \kappa^*(1, j-1),$$

which is proportional to N_s and smaller than $2N_s$. The computation of $N_s - 1$ diagonals results in the complexity of $\mathrm{O}({N_s}^2)$ operations. ∎

Note that this proof defines the improved algorithm, which consists simply of using recursion (65) following diagonals. Also observe that this faster technique is also very simple. The constant factor in $\mathrm{O}({N_s}^2)$ can be quite small because there is no need for complicated programming or data structures.

Since we normally have $N_s \gg N_g$, if we need to know the optimal solution to only a few group sizes we need to avoid spending time computing solutions that are not needed. A practical solution is to stop the algorithm described above in the row $N_g - 1$ and use full search only in the last row, but that is not asymptotically optimal.

Theorem 4.2 *The optimal partition of an alphabet with N_s symbols in N_g can be computed with $O(N_g N_s)$ operations and $O(N_g N_s)$ memory complexity.*

Proof: We can compute the values of $\ell^*_{i,j}$ and $\kappa^*(i, j)$ one row at a time. To do that, for each row i we can define the matrix $\mathbf{A}$ with elements

$$a_{m,n} = \begin{cases} \ell^*_{i-1,i+m-2} + \gamma_{i+m-1,i+n-1}, & 1 \leq m \leq n \leq N_s - i, \\ \infty, & 1 \leq n < m \leq N_s - i, \end{cases},$$

and note that the problem of finding all the minimum values in (41) for a given value of i (finding $\kappa^*(i, i+1), \kappa^*(i, i+2), \ldots$) is the same as finding the minimum values in each column of $\mathbf{A}$.

From the definition above and Theorem 3.6 we conclude that, given values $m_1 < m_2$ and $n_1 < n_2$,

$$\begin{aligned} a_{m_1,n_1} - a_{m_2,n_1} &= \ell^*_{i-1,i+m_1-2} + \gamma_{i+m_1-1,i+n_1-1} - \ell^*_{i-1,i+m_2-2} - \gamma_{i+m_2-1,i+n_1-1} \\ &\leq \ell^*_{i-1,i+m_1-2} + \gamma_{i+m_1-1,i+n_2-1} - \ell^*_{i-1,i+m_2-2} - \gamma_{i+m_2-1,i+n_2-1}, \end{aligned}$$

which means that

$$a_{m_1,n_1} - a_{m_2,n_1} \leq a_{m_1,n_2} - a_{m_2,n_2},$$

and thus matrix $\mathbf{A}$ is *totally monotone*, i.e., its elements satisfy the property

$$a_{m_1,n_1} \geq a_{m_2,n_1} \quad \Rightarrow \quad a_{m_1,n_2} \geq a_{m_2,n_2}.$$

An algorithm for the computation of all the minimum row values of such a totally monotone matrix with complexity with $\mathrm{O}(N_s - i)$ operations was proposed by Aggarval *et al.*, and it commonly knwon as SMAWK [9]. Using this algorithm for each row, the complexity of computing $N_g - 1$ rows is $\mathrm{O}(N_g N_s)$. ∎

Note that the SMAWK algorithm is somewhat more elaborate than that used in the proof of Theorem 4.1, so it may not be the best choice when $N_g = N_s$.

The faster algorithms described above cannot be used on the search for optimal dyadic partitions, since they are based on Theorems 3.6 and 3.9, which are are not valid for dyadic groups. However, the complexity of recursion (41) can be reduced simply by eliminating the terms in which $\tilde{\ell}_{i,j} = \infty$. Since these are defined by powers of two, the number of values of k tested is not larger than $\log_2 N_s$, and in consequence the complexity is $\mathrm{O}({N_s}^2 \log N_s)$ for finding all optimal partitions, and $\mathrm{O}(N_g N_s \log N_s)$ for finding one optimal partition.

The analysis above only covers the worst-case complexity. There are other ways of using the results in Section 3.4 for accelerating the search for optimal partitions. For instance, we can use the fact that the values of $\gamma_{i,j}$ (or finite values of $\tilde{\gamma}_{i,j}$) are monotonic, and we can stop the search in (41) when they exceed the value of the current best solution.

5 Conclusions

In this work we study *symbol grouping*: a technique for the reduction of the complexity of entropy coding that is quite effective, and that can be used in a wide range of situations. While this method had proved its effectiveness and usefulness in many practical coding methods, there were few studies of its properties.

The reduction in complexity provided by symbol grouping comes at the expense of loss in compression. Some simple calculations show that this loss is in the form of relative entropy, i.e., the Kullback-Leibler distance between two probability distributions. However, this form

does not provide an intuitive insight on the potential of symbol grouping. We propose a new approximation of the Kullback-Leibler distance that is very precise in the full range of practical applications, and that let us see that the compression loss (coding redundancy) due to symbol grouping can be very small in many types of source probability distributions.

We also show how the redundancy function can be decomposed in independent parts defined by the entropy of the normalized distribution inside each group. Each part is shaped as a scaled version of a constant minus the entropy. Thus, we conclude that the redundancy function is linear in certain directions, while its projection to certain subspaces is strictly convex.

While the analysis shows that in many situations the symbol grouping redundancy can be practically insignificant, the challenge is to find the partition of the source alphabet (symbol groups) that maximizes the complexity reduction. This basically corresponds to finding the partition with the smallest number of groups such that the relative redundancy is smaller than a constant, or equivalently, given a number of groups, finding the partition that minimizes the symbol grouping redundancy.

We explain how this type of problem is related to well-known set-partitioning problems, and present some information about the number of solutions of the combinatorial problems. We show that in our case it is important to consider these facts because when we constrain the group sizes to be powers of two (dyadic groups), the reduction in the number of possibilities can be so large that we may not be able to find good partitions, and the optimization problem has to be slightly modified.

From particular properties of our optimization problem we show a necessary optimality condition that enables us to solve the problem with much more efficient methods, based on dynamic programming. Next we present the dynamic programming recursive equations, and a rich set of mathematical properties of the problem data and of the optimal solutions. We show that these properties correspond to properties of Monge matrices, which can be used in the design of more efficient dynamic programming algorithms.

Finally, we analyze the complexity of algorithms for the computation of optimal alphabet partitions. The direct implementation of the dynamic programming recursion needs $\mathrm{O}({N_s}^3)$ operations and $\mathrm{O}({N_s}^2)$ memory to find all the optimal partitions of an alphabet with N_s data symbols. We show that, by using the special properties of our problem, it can be solved with $\mathrm{O}({N_s}^2)$ operations and $\mathrm{O}({N_s}^2)$ memory. In addition, methods to find only the minimum redundancy values, without the information about the optimal groups, require only $\mathrm{O}(N_s)$ memory. The complexity of finding only the optimal partition in N_g groups is shown to be $\mathrm{O}(N_g N_s)$ operations and $\mathrm{O}(N_g N_s)$ memory.

A Enumeration of Linear Partitions

The total number of linear partitions is defined by a well-known combinatorial function: $\Xi_l(N_s, N_g) = C(N_s - 1, N_g - 1)$. The total number of dyadic linear partitions $\Xi_d(N_s, N_g)$ is related to the number of binary paritions of a number [27].

Values of $\Xi_d(N_s, N_g)$ can be computed recursively using

$$\Xi_d(N_s, N_g) = \sum_{k=0}^{\lfloor \log_2(N_s-1) \rfloor} \Xi_d(N_s - 2^k, N_g - 1). \tag{66}$$

While this formula is quite simple, and is good for obtaining many values of $\Xi_d(N_s, N_g)$, it does not provide much intuition about the expected behavior of the function.

The easiest cases to evaluate analytically are those in which N_g is nearly equal to N_s, since most subsets have only one element. For example, when $N_g = N_s - 1$, we can only have groups with one or two elements, and the total number of partitions is the same as in the general case. When $N_g = N_s - 2$, we can have groups with one, two or three elements, and the number of dyadic partitions is the total number when we exclude all partitions that have groups with three elements. The formulas obtained for the first cases are

$$\begin{aligned}
\Xi_d(m, m) &= 1 \\
\Xi_d(m, m-1) &= m - 1 \\
\Xi_d(m, m-2) &= (m-2)(m-3)/2 \\
\Xi_d(m, m-3) &= (m-3)[6 + (m-4)(m-5)]/6 \\
\Xi_d(m, m-4) &= (m-4)(m-5)[24 + (m-6)(m-7)]/24 \\
\Xi_d(m, m-5) &= (m-5)(m-6)(m-7)[60 + (m-8)(m-9)]/120 \\
\Xi_d(m, m-6) &= (m-6)(m-7)\{360 + (m-8)(m-9)[120 + (m-10)(m-11)]\}/720.
\end{aligned}$$

We can compare these values with the linear partition case

$$\Xi_l(m, m-k) = (m-1)(m-2)\cdots(m-k)/k!,$$

and observe that, for these particular values, the growth of Ξ_d is indeed significantly slower than the growth of Ξ_l.

In our applications we have to consider a more unusual property of Ξ_d: when N_g is relatively small the value of Ξ_d can change abruptly with N_g. Let $W_H(n)$ be the Hamming weight of the integer number n, i.e., the number of nonzero coefficients in its binary representation. Consider the following properties of Ξ_d:

$$\begin{aligned}
N_g &< W_H(N_s) &&\Rightarrow \Xi_d(N_s, N_g) = 0, \\
N_g &= W_H(N_s) &&\Rightarrow \Xi_d(N_s, N_g) = N_g!, \\
N_g &= W_H(N_s) + 1 &&\Rightarrow \Xi_d(N_s, N_g) = c_{10} N_g!/2 + c_{11} N_g!/6.
\end{aligned}$$

The first case is a consequence of the fact that the binary representation is unique, and the Hamming weight is therefore the minimum number of dyadic components. When the number of groups is equal to the Hamming weight the size of the subsets is uniquely defined by the binary representation of N_s, and the number of partitions corresponds to the number of permutations of these subsets.

In the third case, c_{10} and c_{11} are, respectively, the total number of occurrences of the bit patterns 10 and 11 in the binary representation of N_s. The number of partitions is dependent on the number of groups with same size. The 11 bit pattern implies three groups with same size, and thus its number of possibilities is 1/3 the number of possibilities defined by the 10 bit pattern. The next cases are increasingly more complicated, but they show how Ξ_d can change abruptly from a very large number to zero. For example, $\Xi_d(2^n-1, n-1) = 0$ while $\Xi_d(2^n-1, n) = n!$, and $\Xi_d(2^n-1, n+1) = (n+1)!(n-1)/6$. The conclusion is that we have to be careful in the choice of N_s and N_g when searching for optimal dyadic partitions.

As explained in Section 3.1, a linear partition of a set of N_s elements (data symbols) in N_g nonempty subsets (symbol groups) can be defined with a sequence of $N_g + 1$ numbers $t_0 = 1 < t_1 < t_2 \cdots < t_{N_g} = N_s + 1$. The enumeration of all possible linear partitions is a simple programming task, which can be easily solved with recursive function calls. The enumeration of dyadic partitions can exploit some properties for faster execution, so we add here a C++ program that enumerates all linear partitions, including the option of enumerating only dyadic partitions.

Fig. 9 shows the required functions. The first function implements the enumeration using an inverted stack, since it is more efficient than recursive function calls. When its Boolean parameter '`dyadic`' is false it generates all possible linear partitions, which are represented in the array '`t[n]`.' We assume that a function called '`Test_Partition`' is used to test each partition, which is normally the computation of the resulting redundancy. Note that '`t[n]`' contains all the information required for controlling the stack, so no other arrays are necessary.

When parameter '`dyadic`' is true the function will only generated dyadic partitions. Note that it uses the same variables and stack structure. However, the generated partition sizes are always powers of two, and it uses the Hamming weight function to avoid generating partition size values that cannot produce a full dyadic partition. For completeness, the function to compute the Hamming weight of an integer is also in Fig. 9.

A simple improvement to the algorithm in Fig. 9 is to compute a group's contribution to the overall redundancy as soon as the group size is fixed, and create partial sums for each stack level. This way the function to evaluate partitions just has to add the contribution of the last group, greatly decreasing the number of computations.

```
void Enumerate_Partitions(bool dyadic, int groups, int symbols)
{
  int * t = new int[groups+1], n = groups - 1;

  t[0] = 1;
  t[groups] = t[n] = symbols + 1;

  while (true) {
    t[n] += (dyadic && (t[n] < t[n+1]) ?  t[n] - t[n+1] :  -1);

    if (t[n] <= n)
      if (++n < groups) continue; else break;

    if (dyadic && (Hamming_Weight(t[n] - 1) > n)) continue;

    if (n == 1)
      Test_Partition(groups, t);
    else {
      --n;
      t[n] = t[n+1];
    }
  }
  delete [] t;
}

int Hamming_Weight(unsigned n)
{
  const int HW[16] = { 0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4 };
  int w = HW[n&0xF];
  while (n >>= 4) w += HW[n&0xF];
  return w;
}
```

Figure 9: C++ functions for enumerating all linear partitions of N_s symbols in N_g nonempty groups.

References

[1] S.W. Golomb, "Run-length encodings," *IEEE Trans. Inform. Theory,* vol. 12, pp. 388–401, July 1966.

[2] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inform. Theory,* vol. 21, pp. 194–203, March 1975.

[3] R.G. Gallager and D.C. Van Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Trans. Inform. Theory,* vol. 21, pp. 228–230, March 1975.

[4] E. Balas and M. Padberg, "Set partitioning: a survey," *SIAM Review,* vol. 18, pp. 710–760, Oct. 1976.

[5] R. F. Rice, *Some Practical Universal Noiseless Coding Techniques,* Technical Report 79–22, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, 1979.

[6] F.F. Yao, "Efficient dynamic programming using quadrangle inequalities," *Proc. ACM Symp. on the Theory of Computing,* pp. 429–435, April 1980.

[7] International Communications Union, "Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus," ITU-T Recommendation T.6, Malaga-Torremolinos, 1984.

[8] J.L. Mott, A. Kandel, and T.P. Baker, *Discrete Mathematics for Computer Scientists & Mathematicians,* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[9] A. Aggarval, M.M. Klawe, S. Moran, P.Shor, and R. Wilber, "Geometric applications of a matrix searching algorithm," *Algorithmica,* vol. 2, pp. 195–208, 1987.

[10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, MA, 1990.

[11] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.

[12] G.K. Wallace, "The JPEG still picture compression standard," *Comm. ACM*, vol. 34, pp. 30–44, April 1991.

[13] W.B. Pennebaker and J.L. Mitchell, *JPEG: Still Image Data Compression Standard*, Von Nostrand Reinhold, New York, 1992.

[14] J.M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, vol. 41, pp. 3445–3462, Dec. 1993.

[15] D.L. Duttweiler and C. Chamzas, "Probability estimation in arithmetic and adaptive-Huffman entropy coders," *IEEE Trans. on Image Processing*, vol. 4, pp. 237–246, March 1995.

[16] A. Said and W.A. Pearlman, "Reduced-complexity waveform coding via alphabet partitioning," *IEEE Int. Symp. on Information Theory,* Sept. 1995.

[17] A. Said and W.A. Pearlman, "A new fast and efficient codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 6, pp. 243–250, June 1996.

[18] A. Said and W.A. Pearlman, "Low-complexity waveform coding via alphabet and sample-set partitioning," *Proc. SPIE: Visual Communications and Image Processing,* SPIE vol. 3024, pp. 25–37, Feb. 1997.

[19] B.G. Haskel, A. Puri, and A.N. Netravali, *Digital Video: an Introduction to MPEG-2*, Chapman & Hall, New York, 1997.

[20] S.S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, New York, 1997.

[21] A. Moffat, R.M. Neal, and I.H. Witten, "Arithmetic coding revisited," *ACM Trans. Inform. Systems,* vol. 16, pp. 256–294, July 1998.

[22] A. Islam, *Set-partitioned image coding,* Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, NY, Sept. 1999.

[23] X. Zou and W.A. Pearlman, "Lapped orthogonal transform coding by amplitude and group partitioning," *Proc. SPIE: Applications of Digital Image Processing XXII,* SPIE vol. 3808, pp. 293–304, 1999.

[24] C. Chrysafis, A. Said, A. Drukarev, A. Islam, and W.A. Pearlman, "SBHP – A Low Complexity Wavelet Coder," *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing,* vol. 4, pp. 2035–2038, June 2000.

[25] A.A. Alatan, M. Zhao, and A.N. Akansu, "Unequal error protection of SPIHT encoded image bit streams," *IEEE J. Select. Areas Commun.,* vol. 18, pp. 814–818, June 2000.

[26] D. Mukherjee and S.K. Mitra, "A vector set partitioning noisy channel image coder with unequal error protection," *IEEE J. Select. Areas Commun.,* vol. 18, pp. 829–840, June 2000.

[27] M. Latapy, "Partitions of an integer into powers," *Discrete Mathematics and Theoretical Computer Science Proc.,* pp. 215–228, 2001.

[28] W.E. Wilhelm, "A technical review of column generation in integer programming," *Optimization and Engineering,* vol. 2(2), pp. 159–200, 2001.

[29] M. Zhao, W.A. Pearlman, and A.N. Akansu, "Performance evaluation and optimization of embedded image sources over noisy channels," *IEEE Signal Processing Letters,* vol. 9, pp. 200–203, July 2002.

[30] D. Chen, Y.-J. Chiang, N. Memon, and X. Wu, "Optimal alphabet partitioning for semi-adaptive coding of sources of unknown sparse distributions," *Proc. IEEE Data Compression Conf.*, March 2003.

[31] A. Said, "Arithmetic Coding," Chapter 5 in *Lossless Compression Handbook* (K. Sayood, ed.), Academic Press, San Diego, CA, 2003.

[32] A. Said *Introduction to Arithmetic Coding Theory and Practice,* Hewlett-Packard Laboratories Report, HPL-2004-76, April 2004.

[33] A. Said *Comparative analysis of arithmetic coding computational complexity,* Hewlett-Packard Laboratories Report, HPL-2004-75, April 2004.

[34] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, Low-Complexity Image Coding with a Set-Partitioning Embedded Block Coder," *IEEE Trans. Circuits and Systems for Video Technology,* in press.