Efficient Alphabet Partitioning Algorithms for Low-complexity Entropy Coding

Amir Said (said@ieee.org)
Hewlett Packard Labs, Palo Alto, CA, USA

Abstract

We analyze the technique for reducing the complexity of entropy coding consisting in the *a priori* grouping of the source alphabet symbols, and in dividing 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. Because this method proved to be quite effective it is widely used in practice, and is an important part in standards like MPEG and JPEG. However, a theory to fully exploit its effectiveness had not been sufficiently developed. In this work, we study methods for optimizing the alphabet decomposition, and prove that a necessary optimality condition eliminates most of the possible solutions, and guarantees that dynamic programming solutions are optimal. In addition, we show that the data used for optimization have useful mathematical properties, which greatly reduce the complexity of finding optimal partitions. Finally, we extend the analysis, and propose efficient algorithms, for finding min-max optimal partitions for multiple data sources. Numerical results show the difference in redundancy for single and multiple sources.

1 Introduction

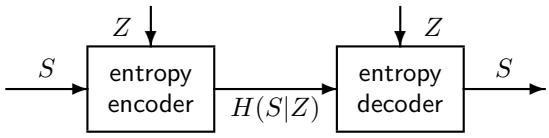
There is a need to find out and study techniques for reducing entropy coding that have wide application, and are valid for many measures of computational complexity. One such general technique, which we call *symbol grouping,* uses the following strategy:

- The source alphabet is partitioned, before coding, into a small number of groups;
- Each data symbol is coded in two steps:
  1. The group that it belongs (*group number*) is coded with a complex method;
  2. The rank of that symbol inside that group (*symbol index*) is coded with a simple method (e.g., its binary representation).

The effectiveness and usefulness of symbol grouping are now well established. For example, it is employed for facsimile coding, and in the JPEG and MPEG standards [6, 8], where the magnitude category corresponds to the group number, and the information in the extra bits corresponds to the symbol index. The same approach is used in image coding methods like EZW and SPIHT (see [12] and its references), where the wavelet coefficient significance corresponds to the group number, which is coded with set-partitioning. The sign and refinement data, which define the symbol index, can be coded using simply one bit per coefficient sign and refinement. Methods

Proceedings of the 2005 Data Compression Conference (DCC'05)
1068-0314/05 $ 20.00 **IEEE**

(a) General coding system using side information.



(b) Coding system using symbol grouping for complexity reduction.

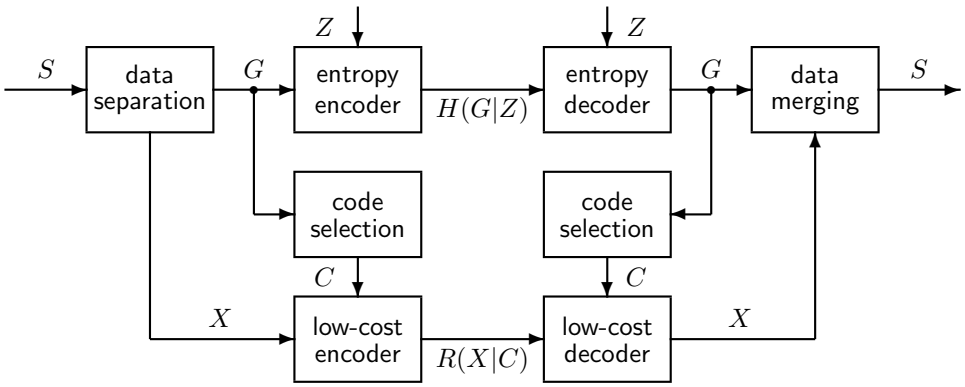


Figure 1: Two alternative systems for data coding: the low-complexity version identifies and separates the data that can be coded with lower computational cost.

that use this approach, but without producing embedded bit streams, have similar compression efficiency [7, 10].

Fig. 1 shows how symbol grouping can be used for context-based entropy coding. 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 [5]

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

Note that, even with the help of side information Z, to improve compression and efficiency we may want to aggregate data, and the alphabet of S can have a very large number of symbols (e.g., millions).

In the symbol grouping case we have G and X corresponding to the group numbers and symbol indexes. We aim to decompose the data samples in a way that we can use a code C, 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.

The advantages of symbol grouping come from the combination of two factors. First, using the binary representation to represent the symbol index is significantly faster than any form of entropy coding. The second factor is defined by how the complexity of coding grows with alphabet size [9]. Large data alphabets create several practical problems, ranging from the effort to design and implement the code, to the

amount of memory necessary to store the codebooks, and the long time to gather statistics sufficiently accurate for adaptive coding.

We minimize complexity by finding the optimal decomposition of S into G and X that satisfies (2). We study this problem under the assumption 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, in bits (cf. Section 3.3 for extensions).

Despite its widespread use, the theoretical analysis of symbol grouping did not receive enough attention. The factors that enable remarkably small compression loss are identified by Said and Pearlman [7], which proposed an heuristic algorithm for finding good alphabet partitions in real time.

Chen *et al.* [11] 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 many more probability patterns are acceptable inside a group.

The main contributions of this work are the following: (a) A study of the optimal alphabet partition problem, and a theorem that states a necessary optimality condition, greatly reducing the number of solutions to be tested; (b) A general dynamic programming solution, plus a set of mathematical properties of the data and results of the algorithm; (c) A study of the computational complexity of the solution algorithms, demonstrating how mathematical properties can reduce the worst-case complexity, and enable fast solutions for large alphabets (e.g., with millions of symbols); (d) Analysis and algorithms for finding optimal alphabet partitions that minimize the worst-case redundancy for multiple source probabilities, and numerical results showing that we can indeed obtain good partitions for many distributions.

Reference [13]—available for downloading—contains more details and all the theorem proofs. This document is organized as follows. In Section 2 we introduce the notation and the analysis of symbol grouping redundancy. In Section 3 we study the problem of finding optimal partitions, present a theorem with a necessary optimality condition, followed 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 extension to multiple data sources, and includes some numerical results. Section 6 presents the conclusions.

2 Analysis of Symbol Grouping

We consider a random data source that generates independent and identically distributed symbols belonging to an alphabet $\mathcal{A} = \{1, 2, \ldots, N_s\}$, represented by a single random variable S, with a probability mass function $p(s)$. The vector $\mathbf{p}$ represents the set of symbol probabilities, and $p_s = p(s)$ denotes the probability of data symbol $s \in \mathcal{A}$. The analysis can be extended to context-based coding by simply replacing the probabilities with conditional probabilities.

Using the definition $p\log(1/p) = 0$ when $p = 0$, the source entropy is

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

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

$$g(s) = \{n : s \in \mathcal{G}_n\}, \quad s = 1, 2, \ldots, N_s. \tag{4}$$

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

$$\bigcup_{s\in\mathcal{G}_n} x(s) = \{1, 2, \ldots, |\mathcal{G}_n|\}, \quad n = 1, 2, \ldots, N_g, \tag{5}$$

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

The probability of a group, $\rho_n = \text{Prob}(g(s) = n)$, and the average probability of the symbols in that group, $\bar{p}_n$, are

$$\rho_n = \sum_{s\in\mathcal{G}_n} p_s, \quad \bar{p}_n = \frac{\rho_n}{|\mathcal{G}_n|}, \quad n = 1, 2, \ldots, N_g. \tag{6}$$

We assume that all indexes of the group $\mathcal{G}_n$ are coded with the same number of bits, $\log_2|\mathcal{G}_n|$. 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. However, we believe it is better to add this and other constraints later (cf. Section 3.3).

The bit rate of the simplified coding method is the entropy of the group numbers plus an average of the fixed rates used to code the symbol indexes, while the loss due to symbol grouping, called *grouping redundancy* and represented by $\ell(\mathcal{P}, \mathbf{p})$, is the difference between the coding rate and the source entropy [13]

$$\ell(\mathcal{P}, \mathbf{p}) = \sum_{n=1}^{N_g} \rho_n \left[\log_2\left(\frac{1}{\rho_n}\right) + \log_2|\mathcal{G}_n|\right] - H(\mathbf{p}) = \sum_{n=1}^{N_g}\sum_{s\in\mathcal{G}_n} p_s \log_2\left(\frac{p_s}{\bar{p}_n}\right). \tag{7}$$

Equation (7) shows that the grouping redundancy is in the form of a relative entropy, or, equivalently, the Kullback-Leibler distance [5] 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 $\bar{p}_n$.

Several factors contribute to make the redundancy small, including the fact that there is a cancellation of positive and negative terms in (7). It is easier to see this fact with the help of the approximation [13]

$$\ell(\mathcal{P}, \mathbf{p}) \approx \frac{1}{\ln(2)} \sum_{n=1}^{N_g} \bar{p}_n \sum_{s\in\mathcal{G}_n} \frac{(1 - p_s/\bar{p}_n)^2}{1 + (p_s/\bar{p}_n)^{2/3}}, \tag{8}$$

A good alphabet partition can yield very small redundancy in a wide range of distributions by exploiting all these properties [13].

3 Optimal Alphabet Partition

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 $\mathcal{A}$ into N_g groups that minimizes the coding redundancy.

$$\begin{array}{ll} \text{Minimize}_{\mathcal{P}} & \ell(\mathcal{P}, \mathbf{p}) \\ \text{subject to} & |\mathcal{P}| = N_g. \end{array} \tag{9}$$

There are many important practical problems that can be formulated as optimal set-partitioning [1]. In general, finding the optimal solution can be an extremely demanding computational task due to its combinatorial nature. Our problem is particularly complicated because its objective function is nonlinear, and can only be computed when the group sizes and probabilities are fully defined. On the other hand, it has many useful mathematical properties, like the following necessary optimality condition [13].

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*

$$\min_{i \in \mathcal{G}_n} \{p_i\} < p_s < \max_{i \in \mathcal{G}_n} \{p_i\}. \tag{10}$$

This means that we can pre-sort the symbols according to probability—which can be done with $\mathrm{O}(N_s \log N_s)$ complexity [4]—and then constrain the partitions to include only adjacent symbols (*linear partitions*). In this case 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 $\mathcal{G}_n = \{s : \, t_{n-1} \leq s < t_n\}$ and the redundancy resulting from grouping (7) can be computed with

$$\ell(\mathcal{T}, \mathbf{p}) = \sum_{n=1}^{N_g} \sum_{s=t_{n-1}}^{t_n - 1} p_s \log_2 \left(\frac{p_s}{\bar{p}_n} \right), \tag{11}$$

$$\bar{p}_n = \frac{1}{t_n - t_{n-1}} \sum_{s=t_{n-1}}^{t_n - 1} p_s, \quad n = 1, 2, \ldots, N_g. \tag{12}$$

3.1 Optimal Partitions via Dynamic Programming

Using Theorem 3.1, we assume that $\mathbf{p}$ is monotonic, 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

$$\gamma_{i,j} = \sum_{s=i}^{j} p_s \log_2 \left(\frac{[j - i + 1] p_s}{\rho_{i,j}} \right), \quad 1 \leq i \leq j \leq N_s, \tag{13}$$

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

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 dynamic programming recursion*

$$\ell^*_{1,j} = \gamma_{1,j}, \qquad 1 \le j \le N_s, \tag{14}$$

$$\ell^*_{i,j} = \min_{i \le k \le j} \left\{ \ell^*_{i-1,k-1} + \gamma_{k,j} \right\}, \quad 2 \le i \le j \le N_s. \tag{15}$$

The recursion (15) 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 (15) when the value of $\ell^*_{i,j}$ is computed

$$\kappa^*(i,j) = \begin{cases} 1, & 1 = i \le j \le N_s, \\ \min\left\{ k : \ell^*_{i,j} = \ell^*_{i-1,k-1} + \gamma_{k,j},\ i \le k \le j \right\}, & 2 \le i \le j \le N_s. \end{cases} \tag{16}$$

Note that with this definition the value of $\kappa^*(i,j)$ is unique. The optimal group sizes are computed by the standard method of backtracking the optimal decisions.

3.2 Mathematical Properties

The results in Section 3.1 already show that we can find *all* optimal alphabet partitions using much less computational effort, but we delay the analysis of the computational implementation and its complexity to Section 4. In this section we show that $\gamma_{i,j}$ and $\ell^*_{i,j}$ have a set of mathematical properties (related to the entropy function) that are useful for further reduction of computational complexity (proofs in [13]).

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, for all integer values of i, j, k and l satisfying $1 \le i \le k \le l \le j \le N_s$, we have*

$$\gamma_{k,l} \le \gamma_{i,j}. \tag{17}$$

Theorem 3.4 *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 $1 \le i \le k \le l \le j \le N_s$, we have*

$$\gamma_{i,j} + \gamma_{k,l} \ge \gamma_{i,l} + \gamma_{k,j}. \tag{18}$$

Theorem 3.3 shows that the matrix $\gamma_{i,j}$, assuming $\gamma_{i,j} = 0$ if $j < i$, is a *monotonic matrix*, and Theorem 3.4 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 3.5 *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 $1 \le i \le k \le l \le j \le N_s$, we have*

$$\ell^*_{k,l} \le \ell^*_{i,j}. \tag{19}$$

Theorem 3.6 *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*

$$\ell^*_{i,j} + \ell^*_{k,l} \geq \ell^*_{i,l} + \ell^*_{k,j}. \tag{20}$$

Theorem 3.7 *For any set of monotonic symbol probabilities* $\mathbf{p}$ *with minimum grouping redundancy* $\ell^*_{i,j}$*, the values of* $\kappa^*(i,j)$ *(defined by (16)) 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*

$$\kappa^*(i,l) \leq \kappa^*(i,j) \leq \kappa^*(k,j). \tag{21}$$

3.3 Additional Constraints

As explained in Section 2, practical applications may require all the groups to have a number of symbols that is a power of two (dyadic partitions). Theorem 3.1 (necessary optimality condition) and Theorem 3.2 (dynamic programming solution) are valid for dyadic partitions too, but we need to penalize partitions with invalid number of elements, which can be done replacing $\gamma_{i,j}$ in (14) and (15) with

$$\tilde{\gamma}_{i,j} = \begin{cases} \gamma_{i,j}, & \text{if } j-i+1 \text{ is a power of 2,} \\ \infty, & \text{otherwise.} \end{cases} \tag{22}$$

Only a subset of the properties in Section 3.2 are also valid for dyadic partitions.

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. If $|\mathcal{G}_n| = m$ we can assign the following number of bits for the symbol with index x

$$\Lambda(x,m) = \begin{cases} \lfloor \log_2 m \rfloor, & 1 \leq x \leq 2^{\lceil \log_2 m \rceil} - m, \\ \lceil \log_2 m \rceil, & 2^{\lceil \log_2 m \rceil} - m < x \leq m. \end{cases} \tag{23}$$

and apply dynamic programming, changing the value of the redundancy of single groups from (13) to

$$\bar{\gamma}_{i,j} = \sum_{s=i}^{j} p_s \left[\Lambda(s-i+1, j-i+1) + \log_2 \left(\frac{p_s}{\rho_{i,j}} \right) \right], \quad 1 \leq i \leq j \leq N_s. \tag{24}$$

4 Analysis of the Computational Complexity

The direct use of dynamic programming equation (15) 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 $\kappa^*(i,j)$, which needs $\mathrm{O}({N_s}^2)$ memory. If only the values ℓ^*_{i,N_s} are desired, we can use the fact that recursion (15) 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.

Note that the calculation of all values of $\gamma_{i,j}$ does not add to the complexity because they can be quickly computed, as stated in the following lemma.

Lemma 4.1 *All values of $\gamma_{i,j}$ needed for the dynamic programming recursions can be computed from two arrays using $O(N_s)$ memory and $O(N_s)$ operations.*

Using the mathematical properties of Section 3.2 we can reduce the computational complexity significantly, using a technique similar to that proposed by Yao [2].

Theorem 4.2 *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.*

If we need to know the optimal solution of only a few group sizes we need to avoid spending time computing solutions that are not needed. Using the SMAWK algorithm [3] we can reduce the complexity by computing one row at a time.

Theorem 4.3 *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.*

5 Optimal Partitions for Multiple Sources

Numerical experiments have shown [7] that the same alphabet partitions can be used for several sources, with small relative loss. This shows that it may be possible to find out partitions that are in a sense optimal for multiple data sources.

We define the data sources by the set of probability vectors $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{N_p}\}$, and we consider the partition $\mathcal{Q}^*$ that minimizes the maximum *relative* redundancy for all the groups, i.e.,

$$\begin{array}{ll} \text{Minimize}_{\mathcal{Q}} & \max\limits_{d=1,\ldots,N_p} \{\ell(\mathcal{Q}, \mathbf{p}_d)/H(\mathbf{p}_d)\} \\ \text{subject to} & |\mathcal{Q}| = N_g. \end{array} \tag{25}$$

We can use an extension of Theorem 3.1 if it is possible to find an order in which all the probability vectors are monotonic (in practice, this is not uncommon). However, we cannot use dynamic programming because the conditions for Theorem 3.2 are not satisfied by the minimum of the maxima. Nevertheless, we can use the previous fast algorithms for efficiently obtaining tight bounds for a branch-and-bound search, or solving sub-problems for a cutting-plane Lagrangian optimization.

We tested the solutions for sources with truncated geometric distributions

$$p_{s,d} = \frac{(1-\rho_d)\,\rho_d^s}{1-\rho_d^{N_s}}, \quad s = 0, 1, \ldots, N_s, \tag{26}$$

where each parameter ρ_d, $d = 1, 2, \ldots, N_p$, was computed to obtain a given source entropy. Table 1 shows the results. We use $N_s = 250$, and source entropies from 1 to

$H(\mathbf{p}_d)$	$\lvert\mathcal{Q}^*\rvert = \lvert\mathcal{P}^*\rvert = 8$		$\lvert\mathcal{Q}^*\rvert = \lvert\mathcal{P}^*\rvert = 10$		$\lvert\mathcal{Q}^*\rvert = \lvert\mathcal{P}^*\rvert = 12$	
(bits/symbol)	$\ell(\mathcal{Q}^*, \mathbf{p}_d)$	$\ell(\mathcal{P}^*, \mathbf{p}_d)$	$\ell(\mathcal{Q}^*, \mathbf{p}_d)$	$\ell(\mathcal{P}^*, \mathbf{p}_d)$	$\ell(\mathcal{Q}^*, \mathbf{p}_d)$	$\ell(\mathcal{P}^*, \mathbf{p}_d)$
1.0	5.295	5.295	1.603	0.092	0.092	0.004
1.5	5.713	5.713	1.524	0.278	0.276	0.033
2.0	4.578	4.383	1.272	0.553	0.506	0.158
2.5	3.335	2.390	1.098	0.977	0.709	0.427
3.0	2.722	2.067	0.996	0.996	0.819	0.463
3.5	2.491	1.471	0.914	0.698	0.797	0.416
4.0	2.251	1.138	0.920	0.672	0.695	0.400
4.5	1.952	1.113	1.031	0.516	0.592	0.323
5.0	1.647	0.769	1.122	0.470	0.565	0.287
5.5	1.390	0.720	1.132	0.380	0.645	0.245
6.0	1.184	0.527	1.071	0.334	0.747	0.216

Table 1: Relative redundancy (% of source entropy) resulting from a partition optimized for multiple sources, $\mathcal{Q}^*$, compared with the redundancy of a partition optimized for a single source, $\mathcal{P}^*$. Data sources have 250 symbols, truncated geometric distribution.

6 bits/symbol. The optimal dyadic partitions for multiple sources are represented by $\mathcal{Q}^*$, while the optimal dyadic partition for a single source is represented by $\mathcal{P}^*$. All the redundancy values in the table are given as a percentage of the source entropy.

These results show that the most significant changes in redundancy happen when the single source redundancy is very small. In all cases we find alphabet partitions such that all the relative redundancies have values that are quite reasonable for practical applications.

6 Conclusions

We analyzed the symbol group technique for reducing the computational complexity of entropy coding, and the problem of finding the partition of the source alphabet (symbol groups) that maximizes the complexity reduction.

From particular properties of our optimization problem we show that necessary optimality condition let 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.

The analysis of the complexity of computating optimal alphabet partitions shows that 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 $O(N_g N_s)$ operations and $O(N_g N_s)$ memory.

Finally, we consider the generalized problem of finding alphabet partitions that are optimal for multiple sources, defining optimality as the minimization of the worst-case relative redundancy, and propose algorithms for solving the problem. Numerical results confirm the robustness of symbol grouping: the optimal solutions can yield—with a small number of groups—very low redundancy for a set of sources with different parameters.

References

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

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

[3] 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.

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

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

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

[7] 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.

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

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

[10] X. Zou and W.A. Pearlman, "Lapped orthogonal transform coding by amplitude and group partitioning," *Proc. SPIE,* vol. 3808, pp. 293–304, 1999.

[11] 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.

[12] 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,* Nov. 2004.

[13] A. Said, *On the reduction of entropy coding complexity via symbol grouping: Part I – redundancy analysis and optimal alphabet partition,* Hewlett-Packard Laboratories Report, HPL–2004–145, Palo Alto, CA, Aug. 2004 (available from http://www.hpl.hp.com/techreports/2004/HPL-2004-145.pdf).