QUANTIZATION USING PERMUTATON CODES WITH A UNIFORM SOURCE

C. Wayne Martin

A Thesis Submitted to the
University of North Carolina at Wilmington in Partial Fulfillment
Of the Requirements for the Degree of
Master of Science

Department of Mathematics and Statistics

University of North Carolina at Wilmington

2003

This thesis has been prepared in the style and format

consistent with the journal

IEEE Transactions on Information Theory.

TABLE OF CONTENTS

ABSTRACT

Permutation coding is a block coding/quantization scheme where the codebook is comprised entirely of permutations of a single starting vector. Permutation codes for the uniform source are developed using a simple algorithm. The performance of these codes is compared against scalar codes and permutation codes developed by different methodologies. It is shown that the algorithm produces codes as good as other more complex methods. Theoretical predictions of code design parameters and code performance is verified by numerical simulations.

ACKNOWLEDGMENTS

We would like to thank Dr. Karlof, Dr. Lammers, and Dr. TenHuisen for their help with this thesis.

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Permutation coding is a block coding/quantization scheme where the codebook is comprised entirely of permutations of a single starting vector. The structure of the codebook allows optimal(nearest neighbor) encoding based on an ordering relationship between n output samples of a source.

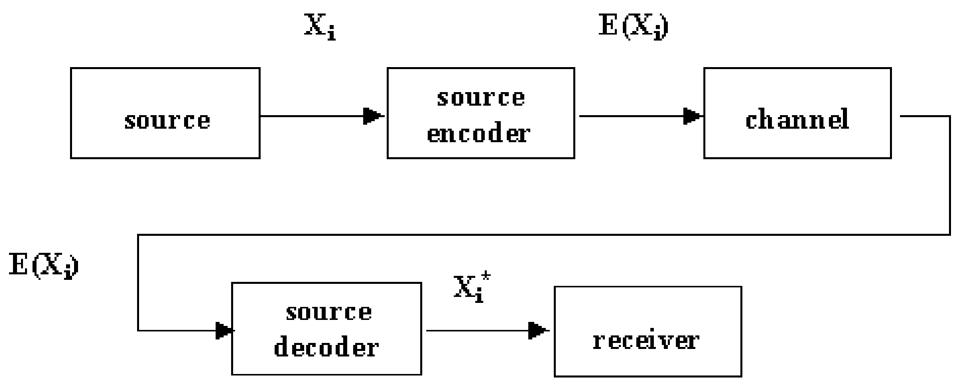
1.1 Motivation

The process of converting a stream of analog or high-rate discrete data into a data stream with lower rate for transmission is called data compression or source coding. A simple communication model using source coding is shown in figure 1. Here, a source generates a data sequence X_i that is compressed by the source encoder and then this compresssed data $E(X_i)$ is sent through the channel. Here we will assume a noiseless channel. Finally, the transmitted data is decoded by the source decoder D and the recovered data $X^* = D(E(X_i))$ is passed to the receiver. One channel property is that there is an upper bound to the number of bits per second that can be correctly transmitted. This bound is called the channel capacity. The source-coding block reduces the number of bits per second with which the input signal is represented, to a number low enough for transmission. The reduced rate signal is called the source-coded signal. Generally, however, source-coding changes the signal, which results in signal distortion.

There are two main types of source coding. The first is called lossless coding where the reconstructed data has to match the original. Many applications, however, do not require a perfect match but rather a match within some constraints. Telephone communication would be such an application. This second case is called lossy data compression. Our studies will only consider lossy data compression

Scalar quantization is the simplest method for the lossy coding of an information source with real-valued outputs. A scalar quantizer maps each output of the source(usually an infinite number) to one of finitely many quantizer output values. Quantizer outputs can

Figure 1: A Simple Model for Source Encoding



be futher encoded using variable length(entropy coding) to improve performance where implementation complexity is important. For this reason, entropy coded scalar quantization is used in moble communication, where simplicity is important.

The negative aspect to scalar quantization is that a variable-length code must be used to achieve a bit rate that only slightly exceeds the quantizer entropy. Additionally, very accurate reproduction requires the quantizer to have many different levels, some of which are more probable than others. This requires that certain words in the variable-length code have to be much longer than others which creates difficult instrumentation problems. Vector quantizers, having a fixed block length provide an alternative to these variable length scalar codes.

The two main performance characteristics of a quantizer Q are its distortion and rate. The distortion $D(Q)$ is the average difference between the source and the quantizer output. For fixed-rate quantization, the rate is defined to be the $\log_2$ of the number of possible outouts. For entropy coding the rate is defined as the entropy $H(Q)$ of the output of Q and is described as entropy-constrained quantization. The coding design probem becomes the minimization of both $H(Q)$ and $D(Q)$, realizing that they are inversely related.

1.2 Literature Overview

The concept of permutation codes was first studied by Slepian[1] in 1965 as a form of channel coding called permutation modulation. Development of permutation codes for more general sources and distortion measures was due to Berger *et al* [2, 3] in 1972. He developed on algorithm for generating good codes and applied the algorithm to a Gaussian source. He then showed how permutation codes could be made equalivant to entropy-constrained scalar quantizers (ECSQ) as the permutation codes approached an infinite length. A key result by Berger was the equivalence between ECSQ and permutation codes. Here he asserts, without proof that no permutation code can perform better than an optimal ECSQ. Townes[6] used Bergers' algorithm to study optimal codes for the Laplacian source in 1984. He studied their performance both theoretically and by simulation. In 2000, Gyorgy[7] studied optimal ECSQ for a wide class of distortion measures. One result was a parametric representation of the optimal distortion-rate function for the uniform source. Permutation codes for the uniform souce was further studied by Goyal[8] in 2001. His main result was to exhibit a set of permutation codes for the uniform source that cannot be equaled with ECSQ. This result contradicts the assertion by Berger[2].

1.3 Thesis Contribution and Overview

In section 2, we introduce the general concept of quantization[4, 7, 9]. A detailed communication model[7] is presented along with applicable source signals[9]. Both scalar and vector quantization is described with performance measures[4]. Fixed length and variable length binary codes are presented with performance measures[9]

Section 3 includes the detailed theory for optimal permutation code design[1, 2]. We start by describing the structure of permutation codes[2]. A theorem is presented and proved for optimum nearest neighbor encoding by use of a simple algorithm[1]. Finally, the theory of optimal code design is developed along with an algorithm for easily generating good permutation codes[2].

Section 4 provides the detailed results of our studies. First we expand on previous studies by developing optimal permutation codes using the Berger algorithm[2] for the uniform source. We compare these our results to codes generated by Goyal[8] and to the optimal distortion-rate function for ECSQ derived by Gyorgy[7]. Using the Berger algorithm[2], we identify codes that perform better than ECSQ, which also condradicts the Berger[2] assertion. Finally, we study the theoretical performance of permutation codes and verify their performance through numerical simulation.

A more detailed view of our communication model is shown in figure 2. Here the source encoder $\mathbf{E}$ and decoder $\mathbf{D}$ are futher subdivided into two parts. First $\mathbf{E}_1 : \mathbf{S} \longrightarrow \{1, \ldots, N\}$ maps the quantizer input X to a unique index value, $\mathbf{I}$. Then the generated index $\mathbf{I}$ is encoded by the uniquely decodable code $\mathbf{E}_2 : \{1, \ldots, N\} \longrightarrow \{0, 1\}^*$ to a sequence of bits $\{b_i\}$, where $\{0, 1\}^*$ denotes the set of all binary sequences of finite length. Let $\mathbf{B} = \{\mathbf{E}_2(\mathbf{I}) : \mathbf{I} \in \{0, \ldots, N\}\}$ be the range of $\mathbf{E}_2$. After transmission through the channel, $\mathbf{D}_2 : \mathbf{B} \longrightarrow \{1, \ldots, N\}$ decodes the orginal index value, and finally $\mathbf{D}_1 : \{1, \ldots, N\} \longrightarrow \{c_1, \ldots, c_N\}$ calculates the quantizer output

$$Q(\mathbf{X}) = \mathbf{D}_1(\mathbf{D}_2(\mathbf{E}_2(\mathbf{E}_1(\mathbf{X})))).$$

2.1 Source Signals

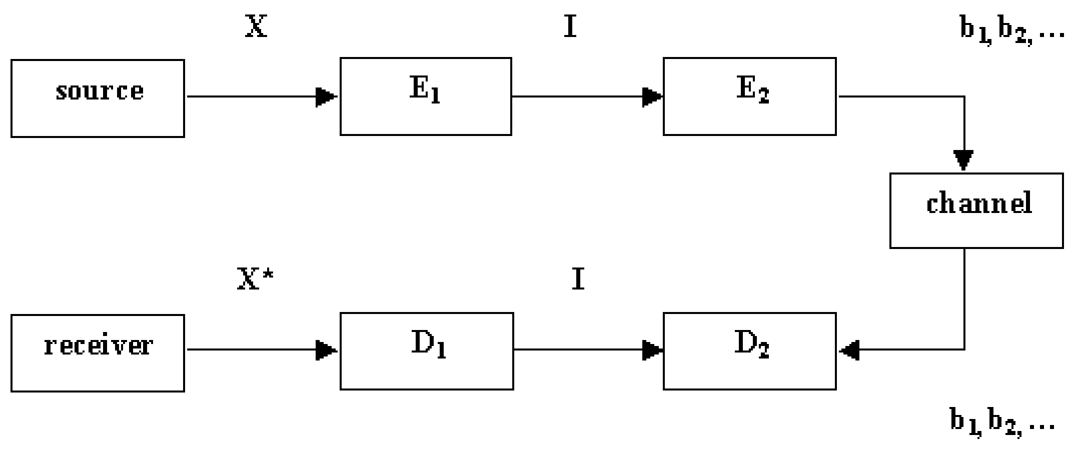
Our studies consider real-valued continuous-time. Mathematically, these signal are functions of time and are denoted as $x(t)$ where $t \in \Re$ and $x(t) \in \Re$. Another name for these signals is analog signals. Sampling is then used to convert the real-valued continuous-time signals into real-valued discrete-time signals, denoted as $x[n]$. Here $n \in \Re$ and $x[n] \in \Re$. The parameter n represents time and corresponds to an instant nT_s where T_s represents the sampling period.

The input signal for source-coding is generally not know in advance, however, many times there is knowledge of the source statistics such as probability density functions and spectral densities.

2.2 Quantization

Block $\mathbf{E}_1$ in figure 2 represents the quantizer. The dictionary(*Random House*) defines quantization as the division of a quantity into a discrete number of small parts, often assumed to be integral multiples of a common quantity. One of the simplest examples of

Figure 2: Detailed Communication Model



quantization is rounding off. For example, any real number x can be rounded off to the nearest integer.

Definition 1 *[7]A k-dimensional N-point quantizer Q is a mapping of a subset S of the k-dimensional Euclidean space R^k into a finite or countably infinite set of distinct k-dimensional real vectors $\{c_1, \ldots, c_N\} \subset \Re^k$ called the codebook of Q. The elements of the codebook, c_i are called the codepoints(codewords) of Q and the associated sets $S_i = \{x : Q(x) = c_i\}, i = 1, \ldots, N$ are called the quantization cells of Q.*

The quality of a quantizer is measured by the goodness of the resulting reproduction in comparison to the original. One way of accomplishing this is to define a distortion measure $d(x, \hat{x})$ where $d(\cdot, \cdot)$ is a nonnegative measurable function of two real variables. Thus, the smaller average distortion means a higher quality quantizer.

Definition 2 *[7]The distortion D of a quantizer is the expectation*

$$\begin{aligned} D(Q) &= E[d(X, Q(X))] \\ &= \int_S d(x, Q(x))\mu_x\ (dx) \qquad (1) \end{aligned}$$

where μ_x is the probability distribution(pdf) corresponding to the random variable X.

The most common distortion measure is the square error $d(x, \hat{x}) = |x - \hat{x}|^2$. This measure will be used exclusively in our studies.

We wish to have the average distortion as small as possible. Negligible distortion can be achieved by letting the cells become numerous and small. There is, however, a cost in terms of the number of bits required to describe the quantizer output to an encoder. Arbitrarily reliable reproduction is not generally possible for digitial storage and communication systems.

A simple method for quantifying this cost is to assume the quantizer 'codes' an input x into a binary representation or channel codeword with an index i. If there are N possible levels and all of the binary codewords have equal length(temporary assumption), the binary vectors will need $\log N$ (or the next larger interger) bits.

The case of $k = 1$ is called scalar quantization and we assume that $\boldsymbol{S} = (\sigma, \tau)$ is an interval of the real line $\Re$. The codepoints c_i and the associated cells $S_i = \{x : Q(x) = c_i\}, i = 1, \ldots, N$ completely characterize Q since $\{S_i\}$ forms a partition of $\boldsymbol{S}$ and

$$Q(x) = c_i \longleftrightarrow x \in S_i.$$

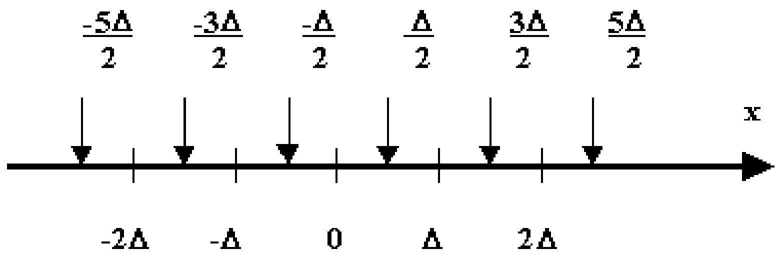
Thus the cells are disjoint and $\bigcup S_i = S$. The cells might take the form $S_i = (a_{i-1}, a_i)$ where the a_i's are called the thresholds and form an increasing sequence. The width of the cell S_i is its length, $a_i - a_{i-1}$. A quantizer is called uniform[4], if all the levels c_i, are equispaced apart(say Δ) and the thresholds a_i, are midway between the adjacent cell walls. An example of a uniform scalar quantizer with cell width Δ and $N = 4$ levels is given in figure 3.

The centroid is defined as the average value of all the inputs which fall within the cell. If the walls of the cells and the probability distribution of the input values are known, the centroid can be determined exactly. For the special case of a uniform probability distribution, the centroid will be located at the Euclidean center of the cell. Ideally, the output codeword associated with a given cell is optimal for that cell. A cell's optimal value is its centroid.

Scalar quantization is simply the mapping of an infinite number of input values x into a finite number of output values, y:

$$Q : x \longrightarrow y$$

Figure 3: A Uniform Quantizer



Scalar quantization is often referred to simply as quantization, even though it is in fact a special case of quantization.

Vector quantization is simply a quantizer with $k > 1$. Vector quantization is just like scalar quantization except all the components of a vector, of say k successive source samples, are quantized simultaneously. They are thus characterized by a k-dimensional partition containing codevectors(codewords).

Vector quantization is a generalization of scalar quantization to higher dimensions. This generalization opens up a wide range of possibilities and techniques not present in the scalar case. Unlike scalar quantization, vector quantization is usually applied to signal that have already been digitized.

2.3 Binary Codes

A list of codewords $A_1, \ldots, A_N$, consisting of vectors of the the binary symbols 0 and 1 is called a binary code with N codewords.

The number of zeros and ones in a codeword is the codeword length. A concatenation of codewords is called a message. Useful transmission systems require that every message can be uniquely decoded. Binary codes may be either fixed or variable length. Fixed length codes are always uniquely decodable, while variable length codes may or may not be. For a uniquely decodable code, there cannot exist a binary sequence that can be written as the concatenation of two different sequences of codewords.

Example 1 *The list* $1111, 1110, 110, 10, 00, 010, 0110$ *is a binary code with alphabet* $\{0, 1\}$.

Example 2 *Consider the code* $\{A_1, A_2, A_3\} = \{0, 01, 10\}$. *The message* 010 *can be decoded as* A_1A_3 *or as* A_2A_1. *Thus the code* $\{A_1, A_2, A_3\}$ *is not uniquely decodable.*

Uniquely decodable variable length codes correspond to the terminal nodes of a binary tree. Figure 4 shows this for the code $\{00, 01, 10, 110, 111\}$. To begin decoding we first start with the root Λ of the tree. If the first symbol is a zero we follow the upper branch to the node 0, otherwise we follow the lower branch to node 1. If, for example, the first symbol was a one and next the symbol is a zero, we proceed from node 1 to node 10 and so on. Finally we arrive at a terminal node which represents the received codeword. The procedure for the next codeword begins again at the root.

2.4 Binary Coding

The purpose of the coding block($\mathbf{E}_2$ in figure 2) is to map the quantizer-output symbols on to binary codewords and to reduce the bit rate to one below $[\log_2(N)]$ bits per symbol. Here, N is the number of distinct quantizer-output symbols.

If the output of the quantizer is encoded with fixed length codewords, the resulting mapping is called a fixed rate quantizer. Coding of the N different samples then requires at least $\log_2 N$ binary bits. These symbols represent k random variables and the rate is given by

$$R_f = \frac{1}{k} \log_2 N. \tag{2}$$

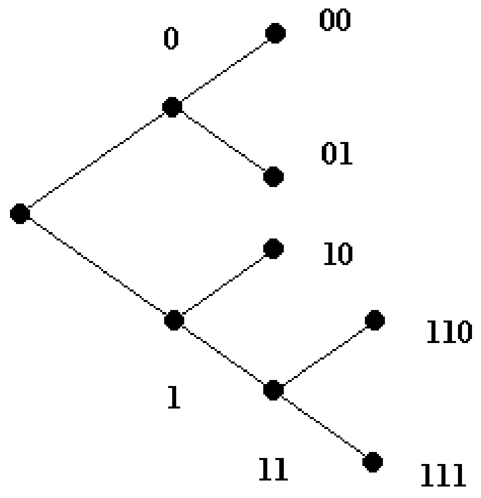
An N-point fixed-rate quantizer is called optimal if it has the lowest possible distortion amoung all N-point quantizers.

Definition 3 *[7]A fixed-rate quantizer Q^* is an optimal N-point quantizer if Q^* has N codepoints and*

$$D(Q^*) = \inf\{D(Q) : Q \textit{ is an N-point quantizer}\}.$$

Quantizers using variable length encoding(entropy coding) can achieve futher compression. Assigning equal numbers of bits to all quantization cells is wasteful if the cells have unequal probabilities. The number of bits produced by entropy coding will, on the average, be smaller if shorter binary codewords are assigned to higher probability cells. Assume that the codeword A_i with length s_i is assigned to a_i. The probability of a_i is p_i and the average

Figure 4: A Code Tree for {00, 01, 10, 110, 111}



codeword length is given by

$$\bar{s} = \sum_{i=1}^{N} p_i s_i. \quad (3)$$

The lowest average bit rate is achieved with the code that gives the smallest $\bar{s}$ for the given source. The following example compares the performance of the variable length code from figure 4 and a comparable fixed length code.

Example 3 *Consider a source producing symbols* $\{a_1, a_2, a_3, a_4, a_5\}$*, with corresponding probabilities of* $\{0.30, 0.25, 0.25, 0.10, 0.10\}$*. The probabilities and codes are shown in table 1. Here we see that the average codeword length of 2.2 for the variable length code is a significant improvement over the fixed length code of 3.0.*

Variable rate coding requires a new definition of rate. For variable-rate coding, the rate is no longer defined as the logarithm of the codebook size. Rather, the instantaneous rate for a given input is the number of binary symbols in the binary codeword(the length of the binary codeword). Then the rate is the average length of the binary codewords, where the average is taken over the probability distributation of the source samples.

Now consider a source producing symbols $a_1, \cdots a_M$ with probabilities $p_1, \cdots p_M$. For example, these symbols could be the output of a quantizer Q. Next consider a sequence of N symbols $q[i], i = 0, \cdots, N-1$, with $N >> 1$ where each $q[i]$ equals an a_j. In this sequence a symbol a_j occurs approximately Np_j times. If it is assumed that each symbol a_j occurs precisely Np_j times, then the number of possible sequences is given by

$$K = \frac{N!}{(Np_1)! \ldots (Np_M)!}. \quad (4)$$

If all K possible sequences are coded with binary codewords, the codewords must consist

Table 1: Results from Example 3

a_i	p_i	variable code	fixed code
a_1	.30	00	000
a_2	.25	01	001
a_3	.25	10	010
a_4	.10	110	011
a_5	.10	111	100
		$\bar{s} = 2.2$	$\bar{s} = 3.0$

of B bits such that

$$2^B = \frac{N!}{(Np_1)! \ldots (Np_M)!} \tag{5}$$

or

$$B = \log_2 \left(\frac{N!}{(Np_1)! \ldots (Np_M)!} \right). \tag{6}$$

By applying Stirling's approximation

$$\log(n!) \simeq n \ \ \log(n), \tag{7}$$

$$B \simeq \frac{\log(N!)}{\log(2)} - \left(\frac{\log((Np_1)!)}{\log(2)} + \ldots + \frac{\log((Np_M)!)}{\log(2)} \right) \tag{8}$$

$$\log(2)B \simeq \log(N!) - [(\log((Np_1)!) + \ldots + \log((Np_M)!)] \tag{9}$$

$$\log(2)B \simeq N\log(N) - [(Np_1 \log(Np_1)) + \ldots + (Np_M \log(Np_M))] \tag{10}$$

$$\frac{\log(2)B}{N} \simeq \log(N) - (p_1 \log(Np_1)) + \ldots + (p_M \log(Np_M)) \tag{11}$$

$$\begin{aligned} \frac{\log(2)B}{N} &\simeq \log(N) - (p_1 \log(N)) - (p_1 \log(p_1)) - \ldots \\ &-(p_M \log(N) - (p_M \log(p_M)). \end{aligned} \tag{12}$$

Noting that

$$\sum_{j=1}^{M} p_M = 1, \qquad \sum_{j=1}^{M} p_M \log(N) = \log(N),$$

we conclude with the following definition.

Definition 4 *[9]The expression*

$$\frac{B}{N} \simeq -\sum_{j=1}^{M} p_j \log_2(p_j) \equiv H(Q[i]) \tag{13}$$

defines the entropy per symbol $H(Q[i])$ of the source producing the sequence $Q[i], i = 0 \ldots N-1$.

Entropy is a useful measure that quantifies the amount of information a signal contains. The entropy is the lower bound for the number of bits per second needed to code a discrete souce. A quantizer designed to provide the smallest average distortion subject to an entropy constraint is called an entropy-constrained scalar quantizer(ECSQ).

One problem with variable rate quantization is the necessity of dealing with the variable numbers of bits that it produces. This means that when communicating over a fixed rate channel, buffer overflows and underflows will have to be addressed by additional instrumentation.

In summary, the goal is to quantize and code the source data with a specific probablity distribution function, into a few binary bits as possible with the least distortion.

3 Encoding With Permutation Codes

Permutation coding is a vector block coding/quantization technique based on the ordering relationship between n output samples of the source. The structure of the codebook allows efficient optimal(nearest neighbor) encoding. Our studies consider real-valued continuous-time signals.

Consider a discrete time information source emitting a sequence of real random variables $X_k, k = 1, 2, \cdots$. These source outputs do not have to be identically distributed or independent. We are concerned with block coding(block quantizing) of an n dimensional random vector from this sequence. That is

$$\mathbf{z}_i = (X_{(i-1)n+1} \cdots X_{in}) = (x_{(i-1)n+1} \cdots x_{in}) \tag{14}$$

where

$$X_j = x_i. \tag{15}$$

A fixed-rate vector quantizer assigns the random vector $\mathbf{z} = (x_1, x_2, \cdots x_n)$ in $\Re_n$ with an element of the codebook $\mathbf{y} = (y_1, y_2, \cdots y_n)$, from the set of M n-vectors in our codebook B.

Thus, the infinite set of n-vectors produced by the source is mapped into the finite set B. When vector $\mathbf{z}$ is emmited by the source, the codeword $\mathbf{y} \in B$, which minimizes some distortion measure $d(\mathbf{z}, \mathbf{y})$ is chosen to represent $\mathbf{z}$. The per-letter average distortion of the code B is

$$D = n^{-1} E \left[\min_{y \in B} d(\mathbf{z}, \mathbf{y}) \right] \tag{16}$$

where the expected value(E) is taken with respect to to the distribution of $\mathbf{z}$.

There are many different distortion measures possible. The one most common the one used here is the mean square error(MSE) measure. The per-output MSE distortion measure is then

$$d(\mathbf{z}, \mathbf{y}) = n^{-1} \sum_{i=1}^{n} (z_i - y_i)^2. \tag{17}$$

The complexity of optimal encoding can grow very quickly with the dimension n. This encoding is generally implemented with an exhaustive search for the nearest neighbor codeword.

3.1 Code Structures

With permutation codes, the codebook is either all unique permutations of the initial vector $\mathbf{y}_1$ (Variant I) or the unique permutations combined with all unique sign choices(Variant II). Here, we will only consider Variant I codes. The first codeword in a Variant I permutation code is an n-vector of the form

$$\mathbf{y}_1 = (\overbrace{u_1\,,\, u_1 \cdots ,\, u_1}^{\leftarrow n_1 \rightarrow},\; \overbrace{u_2\,,\, u_2\,,\, \cdots ,\, u_2}^{\leftarrow n_2 \rightarrow},\; \overbrace{u_k\,,\, \cdots ,\, u_k}^{\leftarrow n_k \rightarrow}) \tag{18}$$

where the u_i and k are real numbers with $u_1 < u_2 \cdots < u_k$ and the n_i are positive integers such that

$$n_1 + n_2 + \cdots , n_k = n. \tag{19}$$

The remaining $M - 1$ codewords in B are all of the unique words obtained by permuting the components of $\mathbf{y}_1$. Thus there are a total of

$$M = \frac{n!}{n_1!\ n_2! \cdots n_k!} \tag{20}$$

codewords. The rate of the code(in bits/sample) is then given by

$$R = n^{-1} \log_2 M. \tag{21}$$

3.2 Nearest Neighbor Encoding

Suppose we wish to encode a vector $\mathbf{Z}$ with a codebook $B = \{\mathbf{Y}_j, j = 1, 2, \cdots, M\}$. The ideal(nearest neighbor) encoder asserts that $\mathbf{Y}_j$ is the optimal codeword that minimizes

$$d_j^2 = |\mathbf{Z} - \mathbf{Y}_j|^2 = |\mathbf{Z}|^2 + |\mathbf{Y}_j|^2 - 2\mathbf{Z} \cdot \mathbf{Y}_j.$$

The encoder seeks to find the vector $\mathbf{Y}$ of the codebook, B, which maximizes the quantity

$$\mathbf{Z} \cdot \mathbf{Y} = \sum_{l=1}^{n} z_l y_l. \tag{22}$$

Theorem 1 *[1]Consider a block distortion measure with the form*

$$d(\boldsymbol{x} \ , \boldsymbol{y}) = \sum_{t=1}^{n} (x_t - y_t)^2$$

where $\boldsymbol{x} \ = (x_1, \cdots, x_n)$, $\boldsymbol{y} = (y_1, \cdots, y_n)$. *Then optimum encoding of Variant I permutation codes with respect to* $d(\boldsymbol{x} \ , \boldsymbol{y})$ *is accomplished by the simple algorithm as follows.*

1. *Replace the* n_1 *largest componets of* $\mathbf{x}$ *by* u_1

2. *Replace the next* n_2 *largest componets of* $\mathbf{x}$ *by* u_2

   $\vdots$

3. *Replace the* n_k *smallest componets of* $\mathbf{x}$ *by* u_k

Use the permutation of y_1 *that results from these replacements to represent* $\mathbf{x}$.

Proof Let $x_1, \cdots, x_n, z_1, \cdots, z_n$ be $2n$-given real numbers. We wish to show that the sum

$$(x_i)_1 z_1 + (x_i)_2 z_2 + \cdots + (x_i)_n z_n \tag{23}$$

is maximized under all permutations of $i_1, \cdots, i_n$ of the integers $1, 2, \cdots, n$ by pairing the largest x with the largest z, the second largest x with the second largest z, etc.

We proceed by induction. The statement is trivially true for $n = 1$. For arbitrary integral $n > 1$, let $\bar{x}$ and $\bar{z}$ denote respectively a largest x_i and z_i. If $\bar{x}$ is not paired with $\bar{z}$ in the sum (23), then the sum contains the two terms $\bar{x}z' + x'\bar{z}$ with $x' \leq \bar{x}$ and $z' \leq \bar{z}$. If x'

and $\bar{x}$ are interchanged, the new sum will not be less than the old sum since the difference

$$(\bar{x}\bar{z} + x'z') - (\bar{x}z' + x'\bar{z}) = (\bar{x} - x')(\bar{z} - z') \geq 0.$$

There is, therefore, a pairing of the x's and z's for which (23) achieves its maximum value and $\bar{x}$ is paired with $\bar{z}$. Delete $\bar{x}\bar{z}$ from this sum. By the induction hypothesis, the remaining $(n-1)$ x's and $(n-1)$ z's can be paired according to size without decreasing the $(n-1)$ term sum. QED

Example 4 *Consider the permutation code based on the codeword*

$$y_1 = (-1,\ -1,\ 0,\ 0,\ 0,\ 1,\ 1).$$

We then have $n = 7,\ k = 3,\ n_1 = 2,\ n_2 = 3,\ n_3 = 2,\ u_1 = -1,\ u_2 = 0,\ u_3 = -1.$ *There are*

$$M = \frac{7!}{(2!)(2!)(3!)} = 210$$

codewords. The rate is

$$R = \frac{\log_2(210)}{7} = 1.10$$

bits per sample. Then, if a source vector were $x = (2.1,\ -1.2,\ 0.8,\ 0.2,\ 0.0,\ 0.2,\ 0.9)$*, the subsequent codeword to represent* $\boldsymbol{x}$ *would be* $\mathbf{c}_j = (1,\ -1,\ 0,\ 0,\ -1,\ 0,\ 1)$ *since the two smallest values of* x *were mapped into* -1*'s, the three middle values were mapped to* 0*'s,* *and the two largest values were mapped to* 1*'s The resulting distortion from (17) would be*

$$\begin{aligned} D &= [(2.1-1)^2 + (-1.2-(-1))^2 + (0.8-0)^2 + (0.2-0)^2 \\ &\quad +(0.0-(-1))^2 + (0.2-0)^2 + (0.9-1)^2]/7 \\ &= 0.4257. \end{aligned}$$

3.3 Optimal Code Design

The permutation code design parameters are the block length n, the number of unique codeword components k, the number of repetitions $\{n_i\}_{i=1}^k$, and the codebook entries themselves $\{u_i\}_{i=1}^k$. We will determine optimal u_i's in terms of order statistics means assuming the other parameters are fixed. The distortion can then be expressed as a function of the other parameters. Our design goal is to find permutation codes that minimize distortion for a given n and a rate of at most R.

Definition 5 *[5]If the random variables $X_1, X_2, \ldots, X_n$ are arranged in assending order of magnitude and then written as*

$$X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)},$$

we call $X_{(i)}$ the ith order statistic $(i = 1, 2, \ldots n)$.

Let π be a permutation that puts the random vector $\mathbf{z}$(from (14,15)) in decreasing order and let $(\xi_1, \xi_2, \cdots, \xi_n) = \pi(\mathbf{z})$. The ξ's are then the order statistics. using the notation

$$S_0 = 0, \qquad S_i = \sum_{j=1}^{i} n_j, \qquad i \in \{1, 2, \ldots, k\} \tag{24}$$

the optimal encoder from theorem 1, replaces ξ_j with u_i for $j = S_{i-1}+1, S_{i-1}+2, \ldots, S_i$. Thus the distortion(from (17)) incurred by the optimal encoder is

$$D = n^{-1} E\left[\sum_{i=1}^{k} \sum_{j=s_{i-1}+1}^{s_i} (\xi_j - u_i)^2\right]. \tag{25}$$

Expanding (25) and noting that

$$\sum_{k=1}^{n} \xi_k^2 \;=\; \sum_{k=1}^{n} X_k^2 \tag{26}$$

results in

$$nD \;=\; E\left[\sum_{i=1}^{k}\sum_{j=S_{i-1}+1}^{S_i}(\xi_j)^2\right] - \sum_{i=1}^{k}\sum_{j=S_{i-1}+1}^{S_i} 2u_i E[\xi_j] + \sum_{i=1}^{k}\sum_{j=S_{i-1}+1}^{S_i}(u_i)^2$$

$$=\; E\left[\sum_{i=1}^{n}(X_i)^2\right] - 2\sum_{i=1}^{k} u_i \sum_{j=S_{i-1}+1}^{Si} E[\xi_j] + \sum_{i=1}^{k} n_i u_i^2. \tag{27}$$

Next we differentate D with respect to u_i set the result equal to zero and solve for u_i.

$$\frac{d(nD)}{d(u_i)} = -2\sum_{j=S_{i-1}+1}^{Si} E[\xi_j] + 2n_i u_i = 0$$

$$u_i \;=\; n_i^{-1}\sum_{j=s_{i-1}+1}^{s_i} E(\xi_j). \tag{28}$$

Substiting $u_i n_i = \sum_{j=S_{i-1}+1}^{Si} E[\xi_j]$ into (27) yields

$$nD \;=\; E\left[\sum_{i=1}^{n} X_i^2\right] - 2\sum_{i=1}^{k} u_i(u_i n_i) + \sum_{i=1}^{k} n_i u_i^2$$

$$D \;=\; n^{-1}\left[E\left(\sum_{j=1}^{n} X_j^2\right) - \sum_{i=1}^{k} n_i u_i^2\right]. \tag{29}$$

The derivation of (28) and (29) did not assume the X_k to be statistically independent or identically distributed. However, when the components of **z** are highly dependent, permutation codes perform relatively poorly even when optimum u_i are used. It is easy to encode permutation codes optimumally, even with large n. Therefore, it is desireable to have a method for generating optimum codes of a specified rate with $n \gg 1$. When n is large, however, there are many ways in which K and the group sizes(n_i) can be chosen so that the code rate R closely approximates a specified value.

3.4 Code Design Algorithm

Berger [2] developed an algorithm that searches for the values of k and $n_1, n_2, \cdots, n_k$ that minimizes D given in (29) for a specified rate and block length. This algorithm is derived as follows. Define

$$p_i \quad = \quad \frac{n_i}{n}, \qquad i = 1, 2, \cdots, k. \tag{30}$$

Then, if each n_i is large, we can use Stirling's formula($\log(n!) \simeq n \quad \log(n)$) to approximate the rate R by

$$R \quad \approx \quad \hat{R} \equiv -\sum_{i=1}^{k} p_i \ \log_2 p_i. \tag{31}$$

Futhermore, the MSE for the optimum u_i of (28) is given exactly by

$$D \quad = \quad n^{-1} E\left[\sum_{i=1}^{n} (X_i)^2\right] - \sum_{i=1}^{k} p_i u_i^2. \tag{32}$$

Treating (31) as an equality, we can minimize D with respect to $p_1, p_2, \cdots, p_K$, subject to the rate constraint. The resulting p_i are

$$p_i \quad = \quad \frac{2^{-\beta u_i^2}}{\sum_{j=1}^{k} 2^{-\beta u_j^2}} \tag{33}$$

where β is chosen so that $\hat{R}$ of (31) equals some specified rate value R^*.

We do not have an analytic solution for the best n_i for three reasons. First, although $n_i = np_i$ according to (30), np_i usually is not an integer. Second, each p_i depends on all the u_i each of which is a complex function of all the n_i per (19) and (28). Finally, the above procedure assumes that k is known. The procedure temporarily holds k fixed. Then we iterate (28), (33), and(30) in that order until the same $\{n_i\}$ set appears on two successive iterations. A detailed description of this algorithm is contained in appendix I.

3.5 An Algorithm For Generating Good Permutation Codes

1. Input n, R^*, and $E\xi_j$ for $j = 1, \cdots, n$
2. Set k equal to the smallest odd integer whose base 2 log exceeds R^*
3. Set $\beta = 1$ and make the integers n_i $i = 1, \cdots, k$, approximately equal
4. compute $u_1, u_2, \cdots, u_k$ from (28)
5. Evaluate the p_i by (33). Adjust β until (31) is satisfied for $\hat{R} = R^*$
6. Compute the new n_i as the closest integers to np_i such that $\sum_{i=1}^{k} n_i = n$
7. If $n_i = 0$ for any i, proceed to step 11
8. If the new and old n_i agree for all i, proceed to step 9, otherwise return to step 4
9. Store $n_1, \cdots, n_k$, D and the exact value of R calulated from (20) and (21)
10. Replace k with $k+2$, reduce the largest n_i by 2. Relabel n_i as n_{i+1} for $i = 1, \cdots, k-2$, and put $n_1 = n_k = 1$. Return to step 4.
11. Print $\{n_i\}, R$ and D stored in step 9. Go to step 13 unless k is odd
12. Set k equal to the smallest even integer whose base 2 log exceeds R^* and return to step 3
13. Stop

The purpose of our studies is first to apply the Berger[2] algorithm to generate permutation codes for a source with a uniform density and compare the results to other similar studies. Berger applied his algorithm to sources with the Gaussian distribution while Townes[6] applied the algorithm to sources with a Laplacian density. Goyal[8] studied permutation codes for the uniform source, but he used other methodologies for code generation. To our knowledge, no one has used the Berger algorithm to generate permutation codes with a uniform density source. We use Maple code to perform our alorithm calculations. Our second purpose is to provide additional counterexamples that contradict the Berger assertion that no permutation code can perform better than an optimal ECSQ. Goyal[8] was the first to identify such a counterexample, however, we produce similar results using the Berger algorithm to generate the codes. Our final purpose is to compare the theoretical performance of permutation codes with numerical simulations.

Definition 6 *[10]The uniform density on $[a, b]$ is defined by*

$$f(x) = \begin{cases} 1/(b-a) & \text{if } \ a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

The uniform density was chosen because of the simplicity of its order statistics which allows application of the theoretical models while still providing reasonable results with comparable data.

To begin our analysis, consider an information source emitting a sequence of randon variables $\{x_k, k = 1, 2, \ldots\}$, each uniformly distributed over the interval $[-\frac{1}{2}, \frac{1}{2}]$ The expected value of the jth order statistic for a sample block length of n is given by Goyal[8]

$$E[\xi_j] \quad = \quad \frac{n+1-2j}{2(n+1)}, \quad j = 1, 2, \ldots, n. \tag{34}$$

Order statistics are usually sorted smallest-to-largest, but we use the reverse order for consistency with other studies of permutation codes.

Example 5 *Consider a sample of block length 6 from our uniform source. The expected value of the order statistics are shown in table 2.*

4.1 Algorithm

The theoretical performance of the permutation code for this source is given by its rate from (21) and distortion from (25). The rate(R) is a function of the block sample length n and the $n_i s$ while the distortion(D) is a function of the $n_i s$ and the order statistics. The goal then, is to find the set $\{k, n_1 \ldots, n_k\}$ such that the distortion is minimized for a specified rate. One can find the best code for a specific rate by an exhaustive search of all possibilities, but as the block length grows, the practically of this method decreases. The method we use to search for good codes is the algorithm described in section 3 implemented using the Maple code.

Example 6 *Implementation of Algorithm*

Step 1:

Let n(variable nt) $= 20, R^*$(variable rs) $= 3.0$ *and calculate* $E\xi_j$(variable E) *from (34)*

```
rs:=3: nt:=20:
for t from 1 by 1 while t<=nt do
E[t]:=(nt+1-2*t)/(2*(nt+1));
od:
```

Step 2:

Set k(variable k) *equal to the smallest odd integer whose base 2 log exceeds R^**

```
k:=9:
```

Table 2: Results from Example 5

j	$E[\xi_j]$
1	.357
2	.214
3	.071
4	-.071
5	-.214
6	-.357

Step 3:

Set β(variable b) $= 1$ *and* $n_1 \cdots n_k$(variable n) *approximately equal*

```
b:=1:


n:=array(1..k,[2,2,2,2,3,3,2,2,2]):
```

step 4:

Compute $u_1 \cdots u_k$(variable u) *from (24,28)*

```
for t from 1 by 1 while t<=k do
s[t]:=sum('n[j]','j'=1..t);
od:


for t from 1 by 1 while t<=k do
u[t]:=(1/n[t])*sum('E[j]','j'=s[t-1]+1..s[t]):
od:
```

Results are shown on table 3

Step 5:

Evaluate p_i(variable p) *by (33). Adjust* β *until (31) is satisfied for* $\hat{R} - R^* < 0.1$ *or 100 iterations.*

```
for f from 1 by 1 while f<100 do
rdold:=rd:
b:=b+1:


for t from 1 by 1 while t<=k do
```

```
    p[t]:=evalf((2^(-b*u[t]^2)/sum('2^(-b*u[j]^2)',j=1..k)));
    od;


    r:=-sum('p[j]*log[2](p[j])',j=1..k);
    delta:=rs-r;
    if abs(delta)<.1 then f:=100 fi;
    od:
```

Step 6:

Compute the new n_i *as the closest intergers to* np_i*. The results are shown on table 4*

Step 7:

$n_i > 0$*, continue with step 8.*

Step 8:

New and old n_i *do not agree, repeat steps 4 thru 6. The results are shown on table 5.*

Step 7a:

$n_i > 0$*, continue with step 8.*

Step 8a:

New and old n_i *agree, proceed to step 9.*

Step 9:

$n_i . \cdots, n_k = [1, 2, 3, 3, 3, 3, 2, 2, 1]$*,* $d = .009013605442$*,* $r = 2.386876696$

Step 10:

$k = 11, n_i, \cdots, n_k = [1, 1, 2, 2, 3, 3, 2, 2, 2, 1, 1]$ *return to step 4.*

step 4a:

Compute $u_1 \cdots u_k$(variable u) *from (24,28)*

```
  for t from 1 by 1 while t<=k do
  s[t]:=sum('n[j]','j'=1..t);
  od:
```

```
for t from 1 by 1 while t<=k do
u[t]:=(1/n[t])*sum('E[j]','j'=s[t-1]+1..s[t]):
od:
```

The results are shown on table 6.

Step 5a:

Evaluate p_i(variable p) *by (33). Adjust* β *until (31) is satisfied for* $\hat{R} - R^* < 0.1$ *or 100 iterations.*

```
for f from 1 by 1 while f<100 do
rdold:=rd:
b:=b+1:


for t from 1 by 1 while t<=k do
p[t]:=evalf((2^(-b*u[t]^2)/sum('2^(-b*u[j]^2)',j=1..k)));
od;


r:=-sum('p[j]*log[2](p[j])',j=1..k);
delta:=rs-r;
if abs(delta)<.1 then f:=100 fi;
od:
```

Step 6a:

Compute the new n_i *as the closest intergers to* np_i*. The results are shown on table 7.*

Step 7b:

$n_1 = n_{11} = 0$*, stop.*

Table 8 provides the sequence of n_i *sets for the example(n = 20, r = 3.0, k = odd) .*

4.2 Results

The performance of permutation codes obtained from this algorithm and using a block length of $n = 20$ is compared against the perfomance of ECSQ[7] and displayed on figure 5. First we observe our counterexample to the Berger[2] assertion that no permutation code can perform better than an optimal ECSQ. We note that the algorithm produces codes similar to or better than ECSQ for rates less than about 1.3. Above this rate, the permutation code performance diverges from ECSQ and approaches an asymptotic distortion of about .01 for large rates. These same general results were obtained by Berger[2] for the Gaussian source and Townes[6] for the Laplacian source. Figure 6 compares our codes to those found by Goyal[8] where he performed an exhaustive search of all possible codes. Our codes are seen to be essentially the same, while using the much simpler algorithm search technique. Parameter values for selected codes are presented in table 9.

4.3 Numerical Simulation

We also performed a numerical simulation using the uniform source for a permutation code with the parameters of $n = 20,\quad k = 3,\quad n_1 = 6,\quad n_2 = 8,\quad n_3 = 6,\quad u_1 = .333,\quad u_2 = 0.0,\quad u_3 = -.333$. These results are shown on table 10. Here columns 1 and 2 are the initial vector and a sorting index. The next three columns represent the encoding process. First the initial vector is sorted in descending order. Then the $u_i s$ are applied per theorem 1. The middle three columns are then resorted on the index to give the encoded vector(last column). This single sample distortion was calculated to be .0198 using (17). This process was then repeated 5000 times and the average distortion form these simulations was .0142. These simulated distortions compare favorably to the theoretical distortion prediction of .0166 from (25)

4.4 Conclusions

We have applied the Berger algorithm to sources with a uniform density. The algorithm produces codes as good as those from Goyal by using a simpler method. An additional counterexample to the Berger assertion that no permutation code can perform better than an optimal ECSQ was found. Theoretical predictions of code design parameters and code performance was verified by numerical simulations.

Table 3: Results from Example 6 Step 4

i	s_i	u_i
1	2	0.4285714286
2	4	0.3333333333
3	6	0.2380952381
4	8	0.1428571429
5	11	0.2380952381
6	14	-0.1190476190
7	16	-0.2380952381
8	18	-0.3333333333
9	20	-0.4285714286

Table 4: Results from Example 6 step 6

i	np_i	new n_i
1	1.188505337	1
2	1.777250755	2
3	2.403307744	3
4	2.938888918	3
5	3.280693894	3
6	3.042289516	3
7	2.403307744	2
8	1.777250755	2
9	1.188505337	1

Table 5: Results from Example 6 step 8

i	np_i	new n_i
1	1.250649448	1
2	1.669416727	2
3	2.42008617	3
4	3.151439132	3
5	3.366498668	3
6	2.950118086	3
7	2.27172559	2
8	1.669416727	2
9	1.250649448	1

Table 6: Results from Example 6 step 4a

i	s_i	u_i
1	1	0.4523809524
2	2	0.4047619048
3	4	0.3333333333
4	6	0.2380952381
5	9	0.1190476190
6	12	-0.02380952381
7	14	-0.1428571429
8	16	-0.2380952381
9	18	-0.3333333333
10	19	-0.4047619048
11	20	-0.4523809524

Table 7: Results from Example 6 step 6a

i	np_i	new n_i
1	.4281929554	0
2	.6733339860	1
3	1.208260562	1
4	2.209438702	2
5	3.540497298	4
6	4.117130248	4
7	3.303920040	4
8	2.209438702	2
9	1.208260562	1
10	.6733339860	1
11	.4281929554	0

Table 8: Sequence of n_i sets from Example 6.

k	n_i										
k = 9		2	2	2	2	3	3	2	2	2	
		1	2	3	3	3	3	2	2	1	
		1	2	3	3	3	3	2	2	1	
k = 11	1	1	2	2	3	3	2	2	2	1	1
	0	1	1	2	4	4	4	2	1	1	0
$n_1 = n_9 = 1, n_2 = n_7 = n_8 = 2$											
$n_3 = n_4 = n_5 = n_6 = 3$											
$r = 2.386876696, D = 0.009013605442$											

Figure 5: Performance of Permutation Codes with $n = 20$ Compared to ECSQ

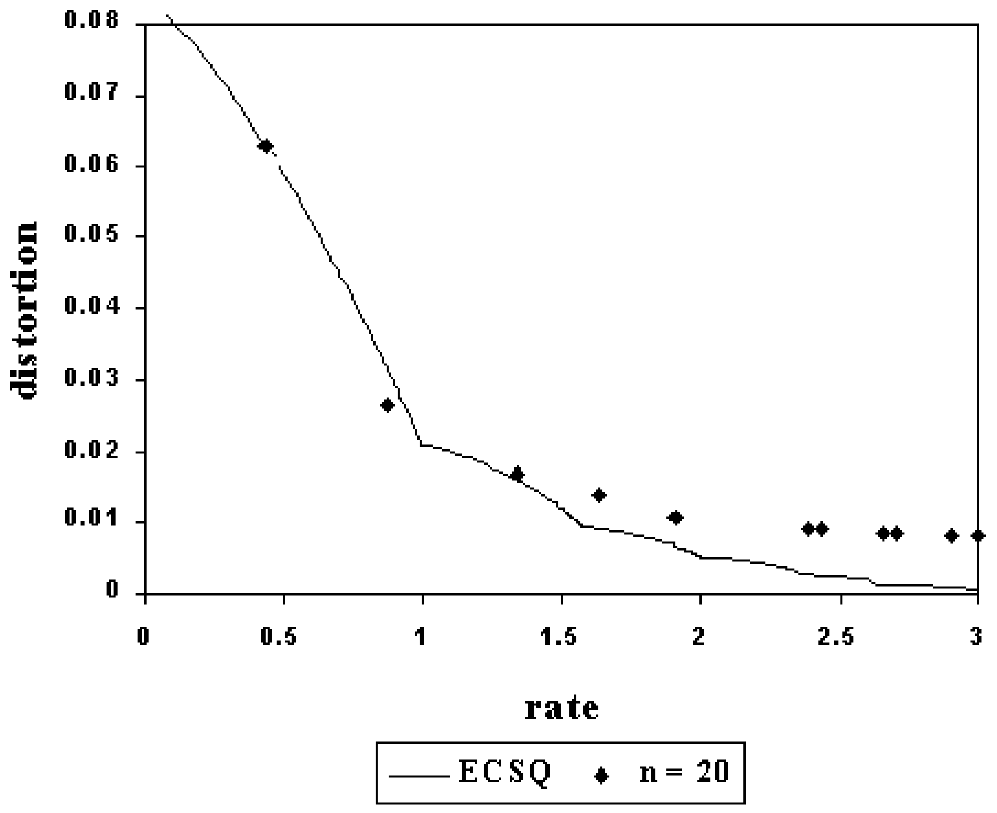


Figure 6: Performance of Permutation Codes with $n = 20$ Compared to Goyal codes

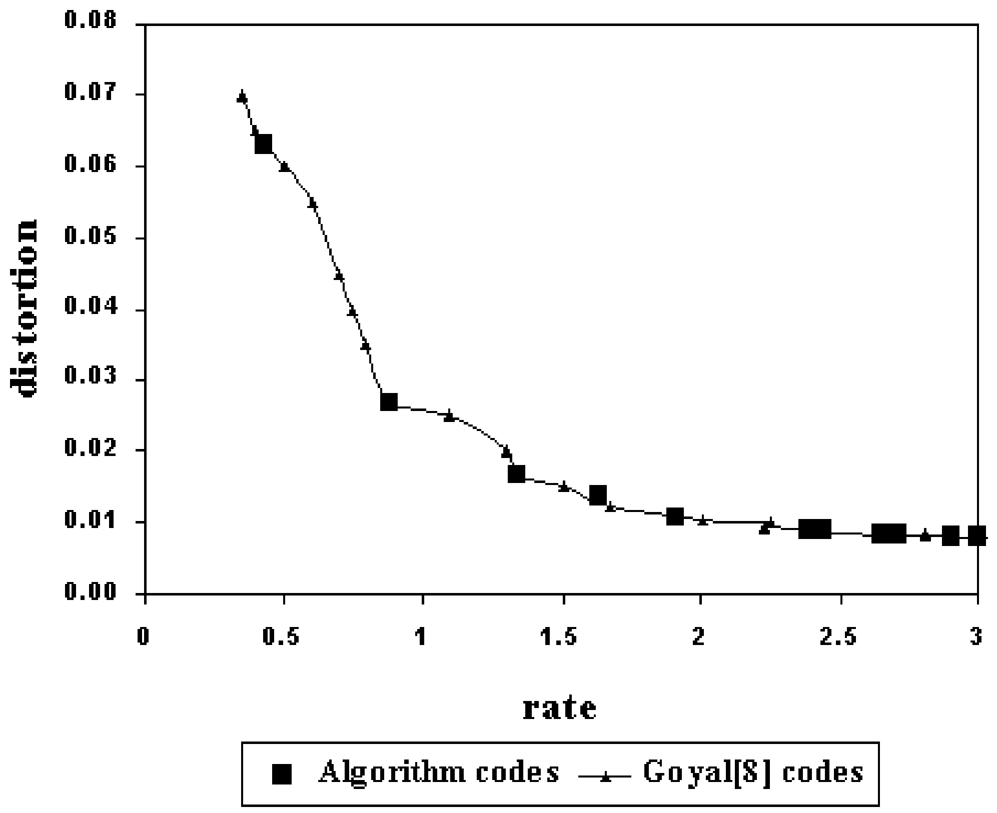


Table 9: Permutation Code parameters for n = 20

R	D	k	i	n_i	u_i
0.42849278	0.0629	3	1	1	0.452380952
			2	18	0
			3	1	-0.452380952
0.874763084	0.0266	2	1	10	0.238095238
			2	10	-0.238095238
1.339723485	0.0167	3	1	6	0.333333333
			2	8	0
			3	6	-0.333333333
2.703869196	0.00833	13	1	1	0.452380952
			2	1	0.404761905
			3	1	0.357142857
			4	2	0.285714286
			5	2	0.190476191
			6	2	0.095238095
			7	2	0
			8	2	-0.095238095
			9	2	-0.190476191
			10	2	-0.285714286
			11	1	-0.357142857
			12	1	-0.404761905
			13	1	-0.452380952

Table 10: Numerical Simulation using the Uniform Source. $n = 20,\ \ k = 3,\ \ n_1 = 6,\ \ n_2 = 8,\ \ n_3 = 6,\ \ u_1 = .333,\ \ u_2 = 0.0,\ \ u_3 = -.333$

Initial vector	index		encoding process				encoded vector		
v	i		v	i	u		v	i	code
-0.1668	1		0.4755	9	0.3333		-0.1668	1	0.0000
-0.4270	2		0.4583	18	0.3333		-0.4270	2	-0.3333
0.3693	3		0.3693	3	0.3333		0.3693	3	0.3333
-0.3072	4		0.3662	15	0.3333		-0.3072	4	0.0000
0.0541	5		0.2116	16	0.3333		0.0541	5	0.0000
-0.0657	6		0.0803	13	0.3333		-0.0657	6	0.0000
-0.0969	7		0.0541	5	0.0000		-0.0969	7	0.0000
-0.1026	8		-0.0657	6	0.0000		-0.1026	8	0.0000
0.4755	9		-0.0969	7	0.0000		0.4755	9	0.3333
-0.1823	10		-0.1026	8	0.0000		-0.1823	10	0.0000
-0.3719	11		-0.1668	1	0.0000		-0.3719	11	-0.3333
-0.2598	12		-0.1823	10	0.0000		-0.2598	12	0.0000
0.0803	13		-0.2598	12	0.0000		0.0803	13	0.3333
-0.3768	14		-0.3072	4	0.0000		-0.3768	14	-0.3333
0.3662	15		-0.3439	17	-0.3333		0.3662	15	0.3333
0.2116	16		-0.3719	11	-0.3333		0.2116	16	0.3333
-0.3439	17		-0.3768	14	-0.3333		-0.3439	17	-0.3333
0.4583	18		-0.3987	19	-0.3333		0.4583	18	0.3333
-0.3987	19		-0.4270	2	-0.3333		-0.3987	19	-0.3333
-0.4459	20		-0.4459	20	-0.3333		-0.4459	20	-0.3333

Single Sample Distortion = 0.0198
5000 Sample Average Distortion = 0.0142
Theoretical Distortion = 0.0166

REFERENCES

[1] D. Slepian, "Permutation modulation," *Proc. IEEE*, vol 53, pp. 228-236, March 1965.

[2] T. Berger, F. Jelinek, and J. K. Wolf, "Permutation codes for sources," *IEEE Trans. Inform. Theory*, vol IT-18, pp. 160-169, January 1972.

[3] T. Berger, "Optimum quantizers and permutation codes," *IEEE Trans. Inform. Theory*, vol IT-18, pp. 759-765, November 1972.

[4] R. M. Gray, and D. L. Neuhoff, "Quantization," *IEEE Trans. Inform. Theory*, vol 44, pp. 2325-2383, October 1998.

[5] H. A. David, *Order Statistics*, 2nd ed. New York: Wiley, 1982.

[6] S.A. Townes and J. B. O'Neal Jr., "Permutation codes for the Laplacian source", *IEEE Trans. Inform. Theory*, vol IT-30, pp. 553-559, May 1984.

[7] A. Gyorgy, "On optimal entropy-constrained scalar quantization," Ph.D. dissertation, Queens University, Ontario, Canada, December 2000.

[8] V. K. Goyal, S. A. Savari, and W.W. Wang, "On optimal permutation codes", *IEEE Trans. Inform. Theory*, vol 47, pp. 2961-2971, November 2001.

[9] R. Veldhuis and M. Breeuwer, *An Introduction to Source Coding*, New York: Prentice Hall 1993.

[10] Lester L Helms, *Probability Theory With Contemporary Applications*, New York: W. H. Freeman and Company, 1996.