Linear (zero–one) programming approach to fixed-rate entropy-coded vector quantisation

A.K.Khandani

Abstract: The problem of the decoding of a shaped set is formulated in terms of a zero–one linear program. Some special features of the problem are exploited to relax the zero–one constraint, and to substantially reduce the complexity of the underlying simplex search. The proposed decoding method has applications in fixed-rate entropy-coded vector quantisation of a memoryless source, in decoding of a shaped constellation, and in the bit allocation problem. The first application is considered and numerical results are presented for the quantisation of a memoryless Gaussian source demonstrating substantial (of the order of a few tens to a few hundred times) reduction in the complexity with respect to the conventional methods based on dynamic programming. It is generally observed that the complexity of the proposed method has a linear increase with respect to the quantiser dimension. The corresponding numerical results show that it is possible to get very close to the bounds determined by the rate-distortion theory, while keeping the complexity at a relatively low level.

1 Introduction

This work introduces a new approach to the problem of the decoding of a shaped set. This has several important applications in communications, including in the fixed-rate entropy-coded vector quantisation (FEVQ) of a memoryless source, which is the focus of the present article. We first present a general formulation of the problem, and then focus the discussion on FEVQ, in which case the corresponding numerical results show a substantial improvement with respect to the conventional methods.

Consider a discrete set A, where a non-negative cost is associated with each element of A. The n-fold Cartesian product of A is denoted as $\{A\}^n$. It is assumed that the cost of an element of $\{A\}^n$ is equal to the sum of the costs of its components. Shaping is achieved by selecting a subset (of a given cardinality) of $\{A\}^n$, say $S_n \subset \{A\}^n$, with a cost less than or equal to a given value C_{max}. We refer to A as the 'constituent subset' and to S_n as the 'shaped set'.

Consider an n-tuple random vector $\boldsymbol{x}$ known as the input. A non-negative distance is defined between each component of $\boldsymbol{x}$ and each element of A. It is assumed that the distance in the n-dimensional space has an additive property. The decoding of a vector $\boldsymbol{x}$ involves finding the element of the shaped set which has the minimum distance to $\boldsymbol{x}$.

The immediate approach to decoding is to perform an exhaustive search. This is achieved by computing the distance of $\boldsymbol{x}$ to all the elements of S_n, and finding the element resulting in the smallest value. However, in most cases, S_n has a huge cardinality and the exhaustive search is impractical. In this case, one needs an algorithmic approach for the decoding. The basic frameworks to develop such an algorithm are the additivity property of the cost, and the additivity property of the distance.

This mathematical formulation can be applied to a number of communications problems where a Cartesian product set, with an additive objective function defined over it, is involved, and the Cartesian product set is truncated (shaped) by imposing a constraint on a cost function which has an additivity property. Examples of such application are: (i) FEVQ of a memoryless source, where A is the set of the reconstruction levels of a scalar quantiser, cost is the self-information associated with the quantiser partitions, and distance is the quantisation distortion, (ii) the bit allocation problem, where A is the set of the available quantisers, cost is the allocated rate, and distance is the average quantisation distortion, and (iii) decoding of a shaped constellation (assuming independent noise), where A is the set of points of a sub-constellation, cost is the energy associated with the points, and distance is the log-likelihood function (square distance for additive white Gaussian noise).

The previously known decoding methods are based on replacing the shaping set by the Voronoi region around the origin of a lattice and using the group property of lattices to perform the decoding [1], or using dynamic programming with the states corresponding to the additive cost [2–4].

In this paper, we introduce a decoding method based on linear programming. This method makes use of the additivity properties mentioned earlier in a very efficient way. Decoding is achieved in a number of steps, where each step finds a point of the shaped set with a smaller distance. This property enables us to provide a tradeoff between the search complexity and the performance. This is the first time that the problem of decoding of a shaped set has been formulated in terms of a linear program. This formulation enables us to apply the rich ideas developed in various contexts of linear programming to this new application.

© IEE, 1998
IEE Proceedings online no. 19990133
DOI: 10.1049/ip-com:19990133
Paper received 30th May 1997

The author is with the Department of Electronic and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

2 Formulation of decoding as a linear program

A linear program (LP) is an optimisation problem involving a linear objective function as well as linear constraints. The basic theorem of linear programming says that in a problem composed of M equality constraints, the optimum answer is composed of M nonzero components. (In the case of an inequality constraint, by introducing an extra variable for each such constraint (denoted as the 'slack variable'), it is transformed into the form of an equality.) Any solution with M nonzero components, denoted as 'basic variables' or 'basis', and satisfying the set of the constraints is called a 'basic feasible solution'. This theorem also gives the necessary and sufficient conditions for a basic feasible solution to be optimum. The simplex method is a systematic search procedure which searches among the basic feasible solutions and finds the optimum answer in a finite number of steps. The search is composed of a set of pivoting operations, where each such operation brings one of the non-basic variables into the basis and replaces it with a basic variable.

Assume that the dimensions of the Cartesian product set are indexed by $i = 0, ..., n-1$, and the elements of the constituent subset are indexed by $j = 0, ..., K-1$. To formulate the decoding problem as a linear program, the jth element of the constituent subset along the ith dimension is identified by the use of a binary variable $\delta_i(j)$, $i = 0, ..., n-1, j = 0, ..., K-1$, where $\delta_i(j) = 0, 1$ and $\Sigma_j\, \delta_i(j) = 1$, $i = 0, ..., n-1$. To select the element indexed by j_0 along the ith dimension, we set $\delta_i(j_0) = 1$ and $\delta_i(j) = 0, j \neq j_0$.

The cost associated with the jth element of A is denoted as $c(j)$. For an n-tuple input x, the distance of the ith component of x to the jth element of A is denoted as $d_i(j)$. The overall distance and cost are equal to: $\Sigma_i\Sigma_j\, \delta_i(j)\, d_i(j)$ and $\Sigma_i\Sigma_j\, \delta_i(j)\, c(j)$, respectively. Using these notations, the decoding problem is formulated as:

$$\begin{aligned}
&\text{Minimise} \quad \sum_{i=0}^{n-1}\sum_{j=0}^{K-1} \delta_i(j) d_i(j) \\
&\text{subject to} \quad \sum_{i=0}^{n-1}\sum_{j=0}^{K-1} \delta_i(j) c(j) + s_c = C_{max} \quad s_c \geq 0 \\
&\qquad\qquad \sum_{j=0}^{K-1} \delta_i(j) = 1 \quad \forall i, \quad \delta_i(j) = 0, 1 \quad \forall i, j
\end{aligned} \tag{1}$$

where s_c is the slack variable of the cost constraint. Each of the equalities $\Sigma_j\, \delta_i(j) = 1$, $i = 0, ..., n-1$, is called an 'indicator constraint'.

2.1 Relaxing the zero–one constraint

The immediate problem in applying the simplex method to solve eqn. 1 is that the variables $\delta_i(j)$ are restricted to be integer numbers, or more specifically 0 and 1. In the context of linear programming, this is called a zero–one program. It is generally known that solving an integer program is substantially more complicated than solving the underlying linear program. Fortunately, in the present case, one can relax the zero–one constraint using the following simple theorem:

Theorem: There is at least one and at most two basic variables corresponding to each indicator constraint.

Proof: To satisfy the equality, there should be at least one basic variable corresponding to each indicator constraint. Considering that: (i) each indicator constraint is composed of a disjoint set of variables, and (ii) the number of basic variables is equal to the number of indicator constraints plus one (because we have just one extra constraint, namely the cost constraint), we conclude that there cannot be more than two basic variables corresponding to any of them. An indicator constraint with two basic variables is called an 'essential indicator constraint'. The theorem says that the number of essential indicator constraints is either zero or one.

To solve the problem in eqn. 1, we just relax the zero–one constraint and then apply the simplex search. The standard form for the resulting LP is shown in Fig. 1.

The constituent subsets are indexed by $0, ..., n-1$ (set-index). The set-indices are shown in the upper row of Fig. 1. The subset indexed by n will be defined later. The rows in Fig. 1, which are indexed by $0, ..., n+1$ (row-index), are generally denoted as the equality constraints. The row-indices are shown in the first column of Fig. 1. The indicator constraints are a subset of the equality constraints indexed by $2, ..., n+1$.

Using the previous theorem, we conclude that in the final LP solution of the equation in Fig. 1, either all the $\delta_i(j)$ variables are zero–one, or exactly two of them are between zero and one. One of the following two cases may happen in the final answer:

Case 1: If all the $\delta_i(j)$ variables are zero–one, it means that the cost constraint is satisfied with strict inequality. In this case, the vector obtained by concatenating the nearest points of different constituent subsets satisfies the cost constraint and is the optimum answer. By performing a simple test before starting the search procedure, one can avoid the computation associated with such cases.

Case 2: If there are two $\delta_i(j)$ variables with values between zero and one, it means that the cost constraint is satisfied with equality. In this case, we set one of these two variables to zero and the other one to unity. The selection is achieved

Maximise z_0 subject to:

$$\begin{array}{c|lllll}
 & 0 & \cdots\cdots & n-1 & n & \\
\hline
0 & \delta_0(0)d_0(0) + \cdots \delta_0(K-1)d_0(K-1) + & \cdots\cdots \delta_{n-1}(0)d_{n-1}(0) + & \cdots \delta_{n-1}(K-1)d_{n-1}(K-1) + z_0 & & = 0 \\
1 & \delta_0(0)c(0) \ + \cdots \delta_0(K-1)c(K-1) \ + & \cdots\cdots \delta_{n-1}(0)c(0) & + \cdots \delta_{n-1}(K-1)c(K-1) & + s_c & = C_{max} \\
2 & \delta_0(0) \quad + \cdots \delta_0(K-1) & & & & = 1 \\
\cdot & & \cdots\cdots & & & \\
n+1 & & \delta_{n-1}(0) & + \cdots \delta_{n-1}(K-1) & & = 1
\end{array}$$

where $s_c \geq 0$ and $\delta_i(j) \geq 0$

Fig. 1 *Standard form for LP resulting from relaxing the zero–one constraint and then applying the simplex search to eqn. 1*

such that the cost constraint is not violated. Obviously, such a selection is always possible and it may result in a slight sub-optimality in the final answer. Noting that this sub-optimality is caused by the rounding of the linear programming answer along only one dimension of a usually very high-dimensional space, we neglect its effect.

In the following, we present an efficient LP solution for the problem in Fig. 1.

2.2 *Efficient simplex search for the underlying linear program*

The special features of the problem which are subsequently used to reduce the search complexity are as follows:

(i) The set of the indicator constraints are non-overlapping and involve variables with unity coefficients, and add up to unity. This allows us to use a basis of a reduced size to perform the steps of the simplex search [5].

(ii) There exists only a single extra constraint involving all the variables (cost constraint). As already mentioned, the immediate consequence of this feature is that one can relax the zero–one constraint. In addition, this results in a 2×2 matrix for the reduced basis which is upper triangular with a unity element in the left upper corner. It is very easy to work with this matrix. Note that the original basis required for a direct application of the revised simplex method would be of size $(n + 2) \times (n + 2)$.

(iii) All the 1-D subspaces have the same set of values for the costs associated with the points. This allows us to reduce the complexity of the pricing out operation involved in the simplex search.

To reduce the effective size of the basis for the simplex search, one basic variable in each indicator constraint is denoted as the key variable. If there are two basic variables (corresponding to an essential indicator constraint), one of them is selected arbitrarily. The basic variable of an essential indicator constraint which is not key is called the 'non-key basic variable'. In the following, we see how one can perform the steps of the revised simplex search without considering the effect of the key variables in an explicit way.

Consider the system obtained by subtracting the column corresponding to each key variable from every other column in their respective constituent subset. (By a 'column', we mean the set of coefficients of a given $\delta_i(j)$ variable in different rows of Fig. 1.) This operation is equivalent to a linear change of variable and results in the so-called transformed system.

It is easy to show that by applying this linear transformation, all the variables except the key variables maintain their original values, and each key variable is replaced by a new variable, which is equal to the sum of all the variables in their respective constituent subset. Obviously, considering the original indicator constraints, the values of these new variables in any feasible solution of the transformed system is equal to one. These variables are deleted from the new system by a direct substitution. This is equivalent to subtracting the columns of the original system corresponding to the key variables from the right hand side. In this way, one can assume that the key variables are absent from the system. This results in a reduced system. In summary, we are dealing with three different systems: (i) the original system, (ii) the transformed system, which is obtained from the original system by applying a change of variable, and (iii) the reduced system which is obtained by deleting the subset of variables which are known to have unity value from the transformed system. Note that the basis for the reduced system, denoted as $\boldsymbol{B}$, is composed of the basic variables which are not key. The main idea is that the steps of the revised simplex search can be carried out using just the inverse $\boldsymbol{B}^{-1}$ of $\boldsymbol{B}$, and the corresponding basic solution of the reduced system. This property is due to the fact that one can easily reach from the basis of $\boldsymbol{B}$ to the original basis of the un-reduced system (denoted as the un-reduced basis of $\boldsymbol{B}$). In the following, we explain the operation of switching the basic variables.

First of all, it is easy to show that the variables to be switched cannot belong to two different non-essential indicator constraints. If the variables belong to the same non-essential indicator constraint, we just change the corresponding key variable (no pivoting is required). A more complicated case occurs when the switching involves an essential indicator constraint. In this case, if the switching is to be done with the corresponding non-key basic variable, we just perform the ordinary pivoting operation. But, if the switching is to be done with the key variable, we first change the key variable with the non-key basic variable and then perform the pivoting operation.

In our case, the basis $\boldsymbol{B}$ is of size 2×2. The basic variable corresponding to the first basis of $\boldsymbol{B}$ is always z_0. Assume that the basic variable corresponding to the second basis of $\boldsymbol{B}$ originates from the equality constraint with the row index λ. If we do not have any essential indicator constraint, this variable is equal to s_c and $\lambda = 1$. Otherwise, it is equal to the non-key basic variable of the essential indicator constraint and $\lambda = i + 2$, where $i = 0, ..., n - 1$ is the index of the corresponding constituent subset.

Expanding a 2-D vector $\boldsymbol{a} = (a_0, a_1)$ on the un-reduced basis of $\boldsymbol{B}$ results in an $(n + 2)$-dimensional vector, say $\boldsymbol{E}$, where $E(0) = a_0$, $E(1) = a_1$ and $E(\lambda) = -a_1$ if $\lambda \neq 1$.

In the following, we have some definitions which facilitate formulation, and also the implementation of the algorithm.

- Define the nth constituent subset to be composed of s_c.
- Define $\mathcal{K}(i) \in [0, K - 1]$ to contain the index of the key variable corresponding to the ith indicator constraint, $i = 0, ..., n - 1$.
- Define the vector of variables, $\boldsymbol{\nu} = \{\nu_i, i = 0, ..., nK\}$, as: $\nu(iK+j) = \delta_i(j)$ for $i = 0, ..., n - 1, j = 0, ..., K - 1$, and $\nu(nK) = s_c$.
- Define $\mathcal{B}(i)$ to contain the index (in $\boldsymbol{\nu}$) of the basic variable corresponding to the ith equality constraint, $i = 1, ..., n + 1$. The basic variable corresponding to the equality constraint indexed by $i = 0$ is always z_0, which is not considered here.
- Define $\theta \in [0, n - 1]$ to contain the index of the essential indicator constraint. If we do not have any essential indicator constraint, then $\theta = n$.

We have: (i) $\theta = \lfloor \mathcal{B}(1)/K \rfloor$, and (ii) $\lambda = 1$ if $\mathcal{B}(1) = nK$, $\lambda = \lfloor \mathcal{B}(1)/K \rfloor + 2$, otherwise.

The corresponding algorithm is explained in the following. The input to the algorithm is the values of $d_i(j)$ and the output is the values of $\delta_i(j)$ solving the equations inFig. 1. The (i, j)th element of $\boldsymbol{B}^{-1}$ is denoted as $\mathcal{B}^{-1}(i, j)$.

Step 1. Start from the basic feasible solution where the selected component for all the constituent subsets is the element of the least cost. Initialise $\mathcal{B}$ to a 2×2 identity matrix. The basic variable corresponding to the second column of $\boldsymbol{B}$ is equal to s_c, resulting in $\lambda = 1$ and $\theta = n$. Initialise $\boldsymbol{\nu}$ to the values taken by the variables. Initialise $\mathcal{K}$ such that the key variable for all the constituent subsets is the element of the least cost.

Step 2. Compute $\phi(j) = B^{-1}(0, 1)c(j)$, $\varphi(j) = B^{-1}(1, 1)c(j)$ for $j = 0, ..., K-1$.

Step 3. Compute the 2-D vector $\bar{\boldsymbol{b}}$ where $\bar{b}(0) = B^{-1}(0, 1)C_{max}$, $\bar{b}(1) = B^{-1}(1, 1)C_{max}$.

Step 4. Compute $\mu_i(j) = d_i(j) + \phi(j)$ for $i = 0, ..., n-1, j = 0, ..., K-1$. This is the inner product of the first row of $\boldsymbol{B}^{-1}$ with the vector composed of the first two elements in each column of the system given in Fig. 1.

Step 5. For each value of the set-index $i = 0, ..., n-1$, find the index $\omega_i \in [0, K-1]$ such that $\mu_i(\omega_i) = \min_j \mu_i(j)$.

Step 6. Compute $\Delta_i = \mu_i(\omega_i) - \mu_i[\mathcal{K}(i)]$ for $i = 0, ..., n-1$, and $\Delta_n = B^{-1}(0, 1)$.

Step 7. Compute the 2-D vector $\bar{\boldsymbol{d}}$, where $\bar{d}(0) = \bar{b}(0) - \Sigma_{i=0}^{n-1} \mu_i[\mathcal{K}(i)]$, $\bar{d}(1) = \bar{b}(1) - \Sigma_{i=0}^{n-1} \varphi[\mathcal{K}(i)]$

Step 8. Find the set-index σ such that $\Delta_\sigma = \min_i \Delta_i$, $i = 0, ..., n$, and assume that the variable resulting in $\min_j \mu_\sigma(j)$ in step 5 (variable indexed by ω_σ within the subset σ) is indexed in $\boldsymbol{v}$ by s. If $\sigma = n$, then $s = nK$ corresponding to the slack variable s_c.

Step 9. If $\Delta_\sigma \geq 0$, exit (the optimum solution is found). Otherwise, bring the variable indexed by s into the basis. This is achieved in the following:

Step 10. Compute $\Gamma_\sigma = \varphi(\omega_\sigma) - \varphi[\mathcal{K}(\sigma)]$.

Step 11. Set $\bar{D}(0) = \Delta_\sigma$, $\bar{D}(1) = \Gamma_\sigma$, and $\Delta_\sigma = 0$.

Step 12. Expand $\bar{\boldsymbol{D}} = [\bar{D}(0), \bar{D}(1)]$ on the unreduced basis of $\boldsymbol{B}$ and if $\sigma \neq n$, increase the component of the result indexed by $\sigma + 2$ by unity [Note 1]. The final answer is denoted as $\boldsymbol{A}_s$.

Step 13. Expand $\bar{\boldsymbol{d}}$ on the unreduced basis of $\boldsymbol{B}$, and increase the components of the result indexed by 2, ..., $n+1$ by unity. This final answer is denoted as $\boldsymbol{b}$.

Step 14. Compute the index r of the variable to leave the basis using

$$r = \mathcal{B}(\psi)$$

where

$$\frac{b(\psi)}{A_s(\psi)} = \min_{A_s(i)>0} \frac{b(i)}{A_s(i)}, \quad i = 1, \ldots, n+1 \tag{2}$$

Assume that this variable belongs to the constituent subset indexed by ρ.

Step 15. If $\rho = \sigma$ and the corresponding constraint is not essential ($\rho \neq \theta$), change the key variable of the constraint, update $\mathcal{K}(\rho)$ and $\bar{\boldsymbol{d}}$ accordingly, set $\mathcal{B}(\psi) = s$, go to step 8.

Step 16. If the variable to leave the basis is the key variable of an essential indicator constraint, make the corresponding non-key basic variable the key, update $\mathcal{K}(\rho)$ and $\bar{\boldsymbol{d}}$ accordingly, set $\mathcal{B}(\psi) = \mathcal{B}(1)$, change the sign of $B^{-1}(1, 1)$, and update $\bar{D}$ accordingly.

Step 17. Perform the pivoting operation by replacing $\boldsymbol{B}^{-1}$ by $\boldsymbol{PB}^{-1}$, where

$$\boldsymbol{P} = \begin{bmatrix} 1 & -\frac{\bar{D}(0)}{\bar{D}(1)} \\ 0 & \frac{1}{\bar{D}(1)} \end{bmatrix} \tag{3}$$

set $\mathcal{B}(1) = s$ and $\theta = \sigma$, update λ (where $\lambda = 1$ if $s = nK$ and $\lambda = \lfloor s/K \rfloor + 2$, otherwise), go to step 2.

Note 1: Recall that expanding a 2-D vector $\boldsymbol{a} = (a_0, a_1)$ on the unreduced basis of $\boldsymbol{B}$ results in an $(n+2)$-dimensional vector, say $\boldsymbol{E}$, where $E(0) = a_0$, $E(1) = a_1$ and $E(\lambda) = -a_1$ if $\lambda \neq 1$.

In some applications of shaping, the constituent subsets are not the same. Examples of this case are in: (i) vector quantisation (in the transform domain) of a correlated source [6], and, (ii) constellation shaping over a non-flat channel [7]. The algorithm given here can be easily modified in order to apply to the more general problem.

In the following, we discuss the application of the proposed algorithm to the problem of fixed-rate entropy-coded vector quantisation of a memoryless source.

3 Fixed-rate entropy-coded vector quantisation of a memoryless source

3.1 Preliminaries

Consider the problem of quantising a source with a nonuniform probability density function. If the dimensionality of the quantiser is not high enough, the entropy coding of the output can result in a substantial decrease in the bit rate. A straightforward entropy coding method presents us with the problem of variable data rate. Also, if the bit rate per quantiser symbol is restricted to be an integer, we are potentially subject to wasting up to one bit of data rate per quantiser output. A solution in a space of dimensionality n is to code the n-fold Cartesian product of a scalar quantiser. (More generally, one can consider the Cartesian product of a set of quantisers with dimensionalities greater than one. All the discussions presented here can be easily generalised to the more general case.) To avoid having a variable data rate, one can select a subset of the n-dimensional symbols having some desirable property, and represent them with code-words of the same length. In such a block-based source coding scheme, as some of the elements in the n-fold Cartesian product space are not allowed, the search for the quantiser output can no longer be achieved independently along the 1-D subspaces. The basic idea is to select the subset of points in such a way that this process can be simplified.

One class of schemes discussed in the literature are based on using a subset of points from a lattice (quantisation lattice) bounded within the Voronoi region around the origin of another lattice (shaping lattice) [1]. In this case, the selected subset forms a group under vector addition modulo the shaping lattice, and the group property of lattices is used to perform the decoding.

Another class of schemes are based on selecting the n-fold symbols with the lowest additive self information. This approach is traditionally denoted as the geometrical source coding [8, 9]. In this case, the selected subset has a high degree of symmetry, which can be used to substantially reduce the search complexity. A method for reducing the search complexity of such a quantiser based on using a state diagram is given by Laroia and Farvardin [2]. Subsequently, Balamesh and Neuhoff [3] develop some complementary techniques to further reduce the complexity.

The core of the idea in the schemes of [2, 3] is to use a trellis diagram with the transitions corresponding to the space dimensions, and with the states corresponding to the additive self information over these dimensions. This results in a trellis composed of n stages. The states s and $s + c$ in two successive stages are connected by a link corresponding to the 1-D symbol(s) of self information c. Consequently, the states in the kth stage, $k = 1, ..., n$, represent the accumulative self information over the set of the first k dimensions. The links connecting two successive stages are labelled by the corresponding distortions. Then, the Viterbi algorithm is used to find the path of the minimum overall distortion through the trellis.

The straightforward approach to dynamic programming is to assign an independent state to each possible value of the self information at a given stage. Even for a moderate value of the bit rate, the number of distinct states in n dimensions can be impractically large. The solution is to synthetically aggregate distinct states into a smaller number. This is denoted as state-space quantisation, and is the key point to the effectiveness of any dynamic programming approach. In [2], the self-information associated with the 1-D symbols is rounded to rational numbers with a common denominator. In [3], to reduce the complexity with respect to [2], these are rounded to integer numbers.

The method developed in [4] is also based on dynamic programming. However, unlike [2, 3], which are based on a component by component analysis, in [4], the recursive structure used in the dynamic programming is built in a hierarchy of stages, where each stage involves the Cartesian product of two lower-dimensional subspaces. This results in several benefits over the conventional approaches to the dynamic programming as used in [2, 3]. Although for moderate values of n (say $n = 32$) this approach is quite effective, for larger values of n the decoding complexity becomes impractical.

3.2 Decoding problem and its relationship with the zero–one programming

Consider a memoryless source and a scalar quantiser composed of K partitions. In the n-fold Cartesian product of this quantiser, we obtain K^n, n-dimensional partitions. The final vector quantiser is selected as a subset of the n-dimensional partitions composed of T elements. The n-dimensional reconstruction vectors are denoted as $\boldsymbol{r}_i$, $i = 0, ..., T - 1$. For a given source vector $\boldsymbol{x}$, the quantisation rule is to find the reconstruction vector $\boldsymbol{r}_i$ which has the minimum square distance to $\boldsymbol{x}$.

Assume that the induced self-information and the expected value of the symbols mapped to the jth 1-D partition are equal to $c(j)$ and r_j, respectively. The self-information associated with a 1-D point is considered as the cost associated with that point. The selection rule for the n-dimensional symbols is to keep the points with the lowest overall additive cost. The n-dimensional reconstruction vectors are obtained by concatenating the corresponding 1-D reconstruction levels, namely r_j. The search operation is formulated as

$$\begin{aligned} &\text{Minimise} && \sum_{i=0}^{n-1}(x_i - r_{k_i})^2 \\ &\text{subject to} && \sum_{i=0}^{n-1} c(k_i) \le C_{max} \end{aligned} \qquad (4)$$

where k_i is the index of the point selected along the ith dimension. The optimisation problem in eqn. 4 can be easily expressed in terms of the original problem in Fig. 1 by replacing the values of the costs, and by setting $d_i(j) = (x_i - r_j)^2$.

3.3 Computing the cardinality of the quantiser

A difficult task in dealing with a high-dimensional quantiser is to compute the corresponding cardinality. To do this, we divide the range of costs associated with the reconstruction levels into a set of uniformly spaced partitions, called the cost-layers. This is achieved in the 1-D subspaces (resulting in the 1-D cost layers) and also in the n-dimensional space (resulting in the n-dimensional cost layers). The costs of the points within a given 1-D cost-layer are approximated with the corresponding mid-value. This allows us to replace the costs by integer numbers. Then, we compute the n-fold convolution of a sequence with the elements corresponding to the number of points in the subsequent 1-D cost-layers. The use of the convolution relies on the additivity property of the cost. The elements of the final sequence are approximately equal to the number of points in the subsequent n-dimensional cost layers. The effect of the constraint on cost is incorporated by truncating this sequence. In practice, to obtain precise results, we have gradually increased the number of 1-D cost layers until the relative changes in subsequent computations becomes negligible.

3.4 Numerical results

The algorithm given in Section 2.2 is applied to the quantisation of a memoryless Gaussian source with respect to the square distance distortion measure. The results are verified against the results obtained using a general purpose linear programming package. The values of the space dimension up to 512 are tested. In general, the number of iterations of the simplex search is quite small (of the order of a few tens). The majority of the iterations do not need pivoting. The overall complexity is substantially lower than the complexity of the methods based on dynamic programming as discussed in [2–4]. At the same time, the performance is better because: (i) no quantisation of the state space is involved, and (ii) one can use much larger values of the space dimensionality.

The quantiser along each dimension is optimised using a standard iterative method (Lloyd–Max quantiser). The 1-D costs are selected as the resulting values for the self-information. Then, the constraint on the cost is applied to the n-fold Cartesian product of the 1-D subspaces, and the reconstruction levels are re-adjusted using a similar iterative procedure. In doing this, the quantisers along different dimensions are always kept the same. A sequence of 6×10^5 Gaussian random numbers is used to design the quantisers, and a different sequence of the same length is used to test them.

The bounds on SNR are computed as $2^{2R}/(\pi e/6)$, where 2^{2R} is the optimum SNR of a Gaussian source for a rate of R bits per dimension, and the factor $\pi e/6$ is the maximum gain obtainable through (spherical) packing.

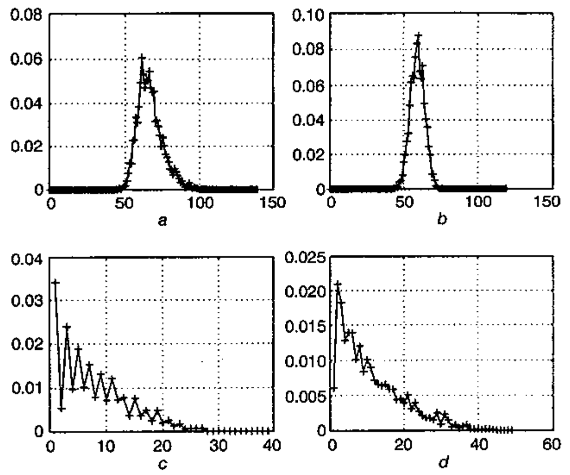

Fig. 2 *Probability distribution of the frequency per block of different operations involved in the proposed method ($n = 128$, $K = 8$ points/D, $R = 2.5$ bits/D)*
a Total number of iterations
b Total number simple exchanges of key and non-key variables
c Total number of exchanges of key and non-key variables done prior to a pivoting
d Total number of pivotings

3.4.1 Performance and complexity of the proposed method: We first discuss a procedure to decrease the peak computational complexity while keeping the degradation in performance negligible. Fig. 2 shows the probability distribution for the frequency per block of different operations involved in the algorithm (n = 128, K = 8 points/D, R = 2.5 bits/D). It is seen that among the three main operations shown in Fig. 2*b–d* the curve corresponding to the pivoting operation has a broader tail. Noting this fact, we impose a constraint on the maximum number of allowed pivots per block to some value, say M_{piv}. It is generally observed that by imposing this constraint, the peak computational complexities decrease, while the corresponding average values and the performance do not change substantially.

Figs. 3 and 4 and Table 1 show examples of the performance and complexity of the proposed method for n = 32, 64, 128, 256, 512, R = 2.5 bits/D, and K = 8, 12, 16 points/D. Referring to Fig. 4, it is observed that the complexity of the proposed method has a linear increase with respect to the quantiser dimension. As a specific example, for n = 512, using 16 points per dimension and for a rate of 2.5 bits/dimension, we need about 325 additions, 469 comparisons 0.24 divisions, and 0.95 multiplications per dimen-

Table 1: Complexity and performance of the proposed quantisation scheme for R = 2.5 bits/D

n	K	M_{piv}	A_a	(P_a)	A_c	(P_c)	A_d	(P_d)	A_m	(P_m)	N_m	SNR, dB
32	8	8	23	(72)	24	(64)	0.19	(1.12)	0.38	(2.25)	128	13.12
32	8	10	24	(85)	24	(72)	0.20	(1.38)	0.40	(2.75)	128	13.13
32	8	∞	24	(129)	24	(100)	0.20	(2.25)	0.40	(4.50)	128	13.14
64	8	12	31	(95)	44	(96)	0.18	(0.81)	0.36	(1.62)	256	13.20
64	8	14	32	(107)	44	(105)	0.19	(0.94)	0.37	(1.88)	256	13.22
64	8	∞	32	(180)	45	(153)	0.19	(1.69)	0.38	(3.38)	256	13.24
*128	8	20	49	(142)	85	(160)	0.18	(0.66)	0.36	(1.62)	512	13.26
*128	8	25	50	(172)	86	(181)	0.19	(0.81)	0.38	(1.62)	512	13.29
128	8	∞	52	(281)	87	(255)	0.20	(1.38)	0.39	(2.75)	512	13.31
128	12	25	78	(228)	107	(233)	0.23	(0.81)	0.70	(2.44)	768	13.32
128	12	30	79	(268)	108	(263)	0.24	(0.97)	0.71	(2.91)	768	13.34
128	12	∞	124	(437)	141	(391)	0.41	(1.62)	1.23	(4.88)	768	13.36
128	16	25	113	(284)	135	(285)	0.28	(0.81)	1.14	(3.25)	1024	13.35
128	16	30	114	(334)	136	(325)	0.29	(0.97)	1.16	(3.88)	1024	13.37
128	16	∞	164	(545)	175	(492)	0.44	(1.62)	1.78	(6.50)	1024	13.38
256	8	25	79	(171)	162	(245)	0.17	(0.41)	0.34	(0.81)	1024	13.28
256	8	30	83	(201)	165	(266)	0.18	(0.48)	0.36	(0.97)	1024	13.32
256	8	35	86	(231)	168	(286)	0.19	(0.56)	0.37	(1.12)	1024	13.34
256	8	40	88	(262)	170	(307)	0.19	(0.64)	0.38	(1.28)	1024	13.35
256	8	∞	91	(502)	173	(467)	0.20	(1.27)	0.40	(2.53)	1024	13.37
256	12	35	136	(307)	206	(357)	0.23	(0.56)	0.69	(1.69)	1536	13.35
256	12	40	141	(347)	210	(387)	0.24	(0.64)	0.72	(1.92)	1536	13.38
256	12	45	143	(388)	212	(417)	0.24	(0.72)	0.73	(2.16)	1536	13.40
256	12	50	145	(428)	214	(447)	0.25	(0.80)	0.74	(2.39)	1536	13.41
256	12	∞	145	(660)	215	(623)	0.25	(1.25)	0.75	(3.75)	1536	13.42
256	16	35	165	(383)	232	(427)	0.22	(0.56)	0.90	(2.25)	2048	13.39
256	16	40	168	(433)	234	(469)	0.23	(0.64)	0.91	(2.56)	2048	13.41
256	16	45	191	(484)	253	(506)	0.26	(0.72)	1.06	(2.88)	2048	13.42
256	16	50	196	(534)	256	(546)	0.27	(0.80)	1.09	(3.19)	2048	13.42
256	16	∞	300	(824)	337	(776)	0.43	(1.25)	1.74	(5.00)	2048	13.43
512	8	50	152	(321)	328	(472)	0.18	(0.40)	0.36	(0.80)	2048	13.36
512	8	60	157	(381)	333	(513)	0.19	(0.48)	0.37	(0.95)	2048	13.39
512	8	∞	167	(742)	342	(754)	0.20	(0.95)	0.40	(1.89)	2048	13.41
512	12	75	275	(627)	423	(717)	0.25	(0.59)	0.75	(1.78)	3072	13.42
512	12	95	278	(787)	428	(841)	0.26	(0.75)	0.77	(2.25)	3072	13.44
512	12	∞	282	(1212)	430	(1158)	0.25	(1.16)	0.76	(3.49)	3072	13.44
512	16	75	325	(783)	469	(866)	0.24	(0.59)	0.95	(2.38)	4096	13.45
512	16	95	360	(983)	497	(1024)	0.26	(0.75)	1.06	(3.00)	4096	13.46
512	16	∞	361	(1224)	498	(1216)	0.26	(0.94)	1.06	(3.75)	4096	13.46

n is the space dimension; K is the number of points per dimension; M_{piv} is the maximum number of allowed pivots per block; A_a, A_c, A_d, A_m are the respective average numbers of additions, comparisons, divisions and multiplications per dimension; P_a, P_c, P_d, P_m are the corresponding peak values; and N_m is the number of memory locations. The bound on SNR obtained from the rate-distortion curve is 13.52dB [Note 2].
* indicates the two specific examples considered in Section 3.4.2

sion to achieve SNR = 13.45dB (the bound obtained from the rate-distortion function is 13.52dB [Note 2]). The corresponding peak values are equal to 783 additions, 866 comparisons, 0.59 divisions, and 2.38 multiplications per dimension. In studying the numerical results, we should keep in mind that in general the gain obtained through the entropy coding of a Gaussian source, and consequently the relative changes in the corresponding values, are not very large numbers. However, it is well known that for sources having a density function with a sharper peak and broader tail (like Laplacian) the corresponding numbers will be larger.

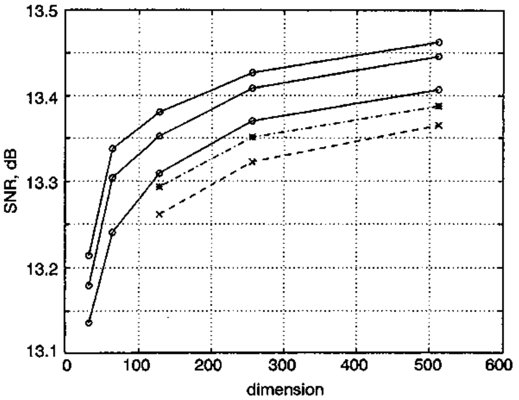

Fig. 3 *Quantisation SNR of a Gaussian source for a bit rate of R = 2.5 bits/D as a function of dimensionality, n = 32, 64, 128, 256, 512*
The bound on SNR obtained from the rate-distortion curve is 13.52 dB. The three solid curves correspond to having $K = 8, 12, 16$ points/D (from bottom to top). The dashed curves correspond to $n = 128, 256, 512$, K = 8 points/D and are obtained by limiting the maximum number of allowed pivots per block to $M_{piv} = 25, 40, 60$ (for the curve designated by *), and to $M_{piv} = 20, 30, 50$ (for the curve designated by ×)

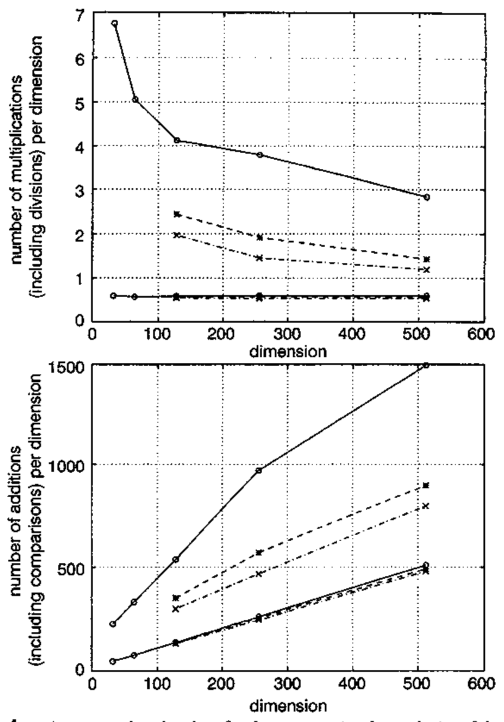

Fig. 4 *Average and peak values for the computational complexity of the proposed quantisation scheme in conjunction with a Gaussian source (n = 128, K = 8 points/D, R = 2.5 bits/D)*
The dashed curves correspond to $n = 128, 256, 512$ and are obtained by limiting the maximum number of allowed pivots per block to $M_{piv} = 25, 40, 60$ (for the curve designated by *), and to $M_{piv} = 20, 30, 50$ (for the curve designated by ×)

Note 2: This is indeed distortion bound minus 1.53dB to account for the lack of packing.

Table 2 shows similar results for $n = 128$, $R = 1.5$ bits/D, and $K = 4$, 8 points/D. It may look surprising that in Table 2 the values of SNR obtained are larger than the bound computed from the rate-distortion function (7.64 against 7.5 [Note 2]). The reason behind this phenomenon lies in the concentration property of high dimensional spaces. As the space dimension increases, the points concentrate on a thin layer of the space with the value of total cost close to C_{max}.

Table 2: Complexity and performance of the proposed quantisation scheme for $n = 128$, $R = 1.5$ bits/D

K	A_a (P_a)	A_c (P_c)	A_d (P_d)	A_m (P_m)	SNR, dB
4	11 (19)	18 (22)	0.03 (0.09)	0.04 (0.12)	7.62
8	44 (78)	38 (62)	0.18 (0.34)	0.40 (0.77)	7.64

K is the number of points per dimension; A_a, A_c, A_d, A_m are the respective average numbers of additions, comparisons, divisions and multiplications per dimension; and P_a, P_c, P_d, P_m are the corresponding peak values

For the points on this outer layer, most of the nearest neighbours are absent. This is due to the fact that for a small change in the total cost, say Δ_c, the number of points with the cost in the range $[C_{max}, C_{max} + \Delta_c]$ is much larger than the number of points with the cost in the range $[C_{max} - \Delta_c, C_{max}]$. This absence of the neighbouring points enhances the quantisation capability of the remaining set. This effect is especially pronounced when the bit rate is low.

Table 3: Complexity and performance of the methods based on dynamic programming ($R = 2.5$ bits/D)

n	K	K_0	$N_s \times 10^3$	$N_a \times 10^3$	$N_c \times 10^3$	$N_m \times 10^5$	SNR, dB
32	8	64	0.25	1.00	0.75	0.08	13.00
32	8	128	0.52	2.08	1.56	0.17	13.12
64	8	64	0.47	1.88	1.41	0.30	13.09
64	8	128	0.96	3.84	2.88	0.61	13.18
128	8	64	0.90	3.60	2.70	1.15	13.03
128	8	128	1.83	7.32	5.49	2.34	13.21
*128	8	256	3.64	14.6	10.9	4.66	13.26
*128	8	512	7.28	29.1	21.8	9.32	13.29
128	12	64	0.27	1.62	1.35	0.35	11.71
128	12	128	0.56	3.36	2.80	0.72	12.88
128	12	256	1.08	6.48	5.40	1.38	13.00
128	12	512	2.18	13.1	10.9	2.79	13.19
128	16	512	1.34	10.7	9.38	1.71	13.09

n is the space dimension; K is the number of points per dimension; K_0 is the number of distinct values used to round the costs; N_s is the number of states; N_a, N_c are the respective numbers of additions and comparisons per dimension; and N_m is the number of memory locations (words of RAM)
* indicates the two specific examples considered in Section 3.4.2

3.4.2 Comparison with the conventional method:

We have studied the performance and the complexity of the dynamic programming approach for the case when the number of distinct (uniformly spaced) values used to round the 1-D costs is equal to $K_0 = 64, 128, 256, 512$. This rounding of costs is the basic ingredient for the method based on dynamic programming. The result of this study is shown in Table 3. In comparing the proposed method with the method based on dynamic programming, we concentrate on $R = 2.5$ bits/D and $K = 8$ points/D. Referring to Table 3, it is observed that by increasing the number of points per dimension, the complexity of the method based on dynamic programming substantially increases and soon becomes unmanageable. This means that by using a small number of points per dimension in our comparison, namely $K = 8$, we are indeed acting to the advantage of the dynamic programming method. Comparing Table 1 and Table 3, it is observed that the proposed method results in a reduction in the complexity of the order of a few tens to a few hundred times. In the following, we present two specific examples (denoted by an asterisk in Tables 1 and 3) for the sake of comparison.

For $n = 128$, $K = 8$ points/D and $R = 2.5$ bits/D, we need on average about 50 (or 49) additions, 86 (or 85) comparisons, 0.19 (or 0.18) divisions, and 0.38 (or 0.36) multiplications per dimension to achieve SNR = 13.29 (or 13.26) dB (the bound obtained from the rate-distortion function is equal to 13.52dB [Note 2]). The corresponding peak values for the computational complexity are equal to 172 (or 142) additions, 181 (or 160) comparisons, 0.81 (or 0.66) divisions, and 1.62 (or 1.31) multiplications per dimension. The corresponding entries are underlined in Table 1. In the dynamic programming approach, by quantising the self-information of the points along each dimension to 512 (or 256) different values, we obtain SNR = 13.29 (or 13.26) dB and the complexity per dimension of the decoder is equivalent to the Viterbi decoding of a trellis with about 7.28×10^3 (or 3.64×10^3) states. This requires about 29.1×10^3 (or 14.6×10^3) additions, 21.9×10^3 (or 10.9×10^3) comparisons per dimension. The corresponding entries are underlined in Table 3. In addition to this large computational complexity, the method based on dynamic programming also needs a large amount of RAM to keep track of the surviving paths through the trellis. In this specific example, this adds up to about 9.32×10^5 (or 4.66×10^5) words of RAM. Note that in the proposed method the RAM is mainly used to store the distances, and is equal to $nK/2$ words of memory. For the example discussed here ($n = 128$, $K = 8$), this is only 512 words of RAM.

4 Summary

We have presented an efficient algorithm for computing the nearest point of a Cartesian product set, with a constraint on an additive cost for the case when the distance measures have an additivity property. This problem is formulated as a zero–one program. The problem has some special features which enables us to remove the zero–one constraint, and to substantially reduce the complexity of the underlying linear program.

The proposed decoding algorithm has application in the decoding of a shaped quantiser, the optimum bit allocation problem, or more generally any problem dealing with allocation of a limited number of resources to a number of customers with an additive objective function, and the decoding of a shaped constellation. We have presented numerical results related to the problem of quantiser shaping showing substantial reduction in complexity with respect to the previously known methods.

5 References

1 EYUBOGLU, M.V., and FORNEY, G.D.: 'Lattice and trellis quantisation with lattice- and trellis-bounded codebooks – high-rate theory for memoryless sources', *IEEE Trans. Inf. Theory*, 1993, **IT-39**, pp. 46–59

2 LAROIA, R., and FARVARDIN, N.: 'A structured fixed-rate vector quantizer derived from variable-length scalar quantizer – Part I: memoryless sources', *IEEE Trans. Inf. Theory*, 1993, **IT-39**, pp. 851–867

3 BALAMESH, A.S., and NEUHOFF, D.L.: 'Block-constrained methods of fixed-rate, entropy coded, scalar quantisation', unpublished work

4 KHANDANI, A.K.: 'A dynamic programming approach to fixed-rate entropy-coded vector quantization', *IEEE Trans. Inf. Theory*, 1996, **IT-42**, pp. 1298–1303

5 DANTZIG, G.B., and VAN SLYKE, R.M.: 'Generalised upper bounding techniques', *J. Comput. Syst. Sci.*, 1967, pp. 213–226

6 JEONG, D.G., and GIBSON, J.D.: 'Uniform and piecewise uniform lattice vector quantization for memoryless Gaussian and Laplacian sources', *IEEE Trans. Inf. Theory*, 1993, **39**, pp. 786–804

7 KASTURIA, S., ASLANIS, J.T., and CIOFFI, J.M.: 'Vector coding for partial response channels', *IEEE Trans. Inf. Theory*, 1990, **IT-36**, pp. 741–762

8 SAKRISON, D.J.: 'A geometrical treatment of the source encoding of a Gaussian random variable', *IEEE Trans. Inf. Theory*, 1968, **IT-14**, pp. 481–486

9 FISCHER, T.R.: 'Geometric source coding and vector quantization', *IEEE Trans. Inf. Theory*, 1989, **IT-35**, pp. 137–145