Vector Quantization

This page contains information related to *vector quantization* (VQ). Currently this page includes information about VQ with regards to compression. In the future, we will make this page a more comprehensive VQ page.

In what applications is VQ used?

Vector quantization is used in many applications such as image and voice compression, voice recognition (in general statistical pattern recognition), and surprisingly enough in volume rendering (I have no idea how VQ is used in volume rendering!).

What is VQ?

A vector quantizer maps *k-dimensional* vectors in the vector space R^k into a finite set of vectors $Y = \{y_i: i = 1, 2, ..., N\}$. Each vector y_i *is* called a code vector or a *codeword*. and the set of all the codewords is called a *codebook*. Associated with each codeword, y_i, is a nearest neighbor region called *Voronoi* region, and it is defined by:

$$V_i = \left\{ x \in R^k : \|x - y_i\| \le \|x - y_j\|, \text{for all } j \ne i \right\}$$

The set of Voronoi regions partition the entire space R^k such that:

$$\bigcup_{i=1}^{N} V_i = R^k$$

$$\bigcap_{i=1}^{N} V_i = \phi \quad \text{for all } i \ne j$$

As an example we take vectors in the two dimensional case without loss of generality. Figure 1 shows some vectors in space. Associated with each cluster of vectors is a representative codeword. Each codeword resides in its own Voronoi region. These regions are separated with imaginary lines in figure 1 for illustration. Given an input vector, the codeword that is chosen to represent it is the one in the same Voronoi region.

y_i = centroid of region i =

$$= \frac{1}{P(x \in R_i)} \int x f(x) \, dx$$

Rate of source code

is $R = \frac{\log_2 N}{k}$ bits/source symbol

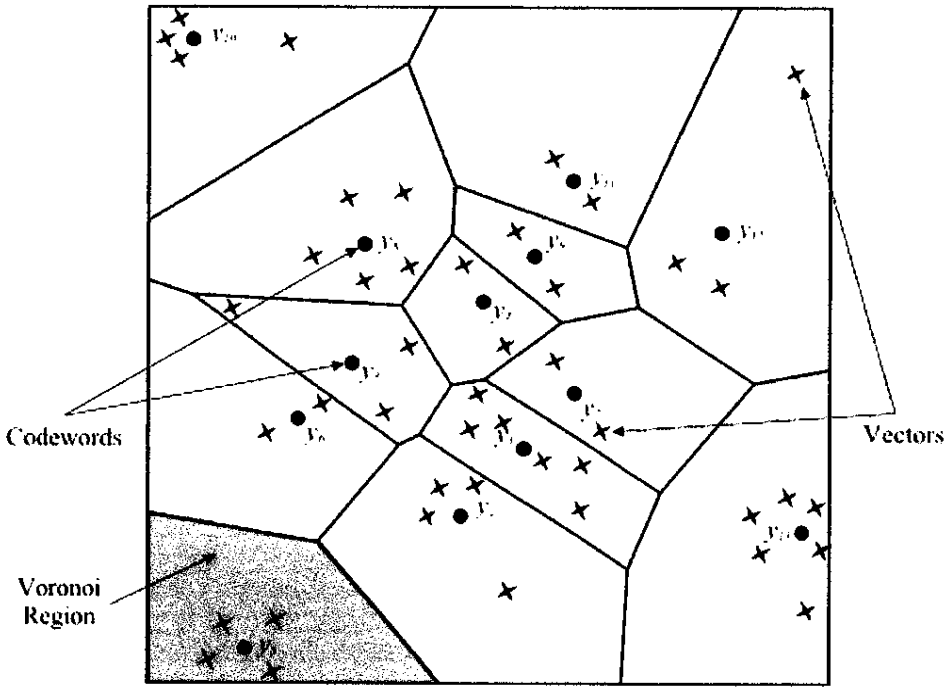

Figure 1: Codewords in 2-dimensional space. Input vectors are marked with an x, codewords are marked with red circles, and the Voronoi regions are separated with boundary lines.

The representative codeword is determined to be the closest in Euclidean distance from the input vector. The Euclidean distance is defined by:

$$d(x, y_i) = \sqrt{\sum_{j=1}^{k} (x_j - y_{ij})^2}$$

where x_j is the jth component of the input vector, and y_{ij} is the jth is component of the codeword y_i.

How does VQ work in compression?

A vector quantizer is composed of two operations. The first is the encoder, and the second is the decoder. The encoder takes an input vector and outputs the index of the codeword that offers the lowest distortion. In this case the lowest distortion is found by evaluating the Euclidean distance between the input vector and each codeword in the codebook. Once the closest codeword is found, the index of that codeword is sent through a channel (the channel could be a computer storage, communications channel, and so on). When the encoder receives the index of the codeword, it replaces the index with the associated codeword. Figure 2 shows a block diagram of the operation of the encoder and decoder.

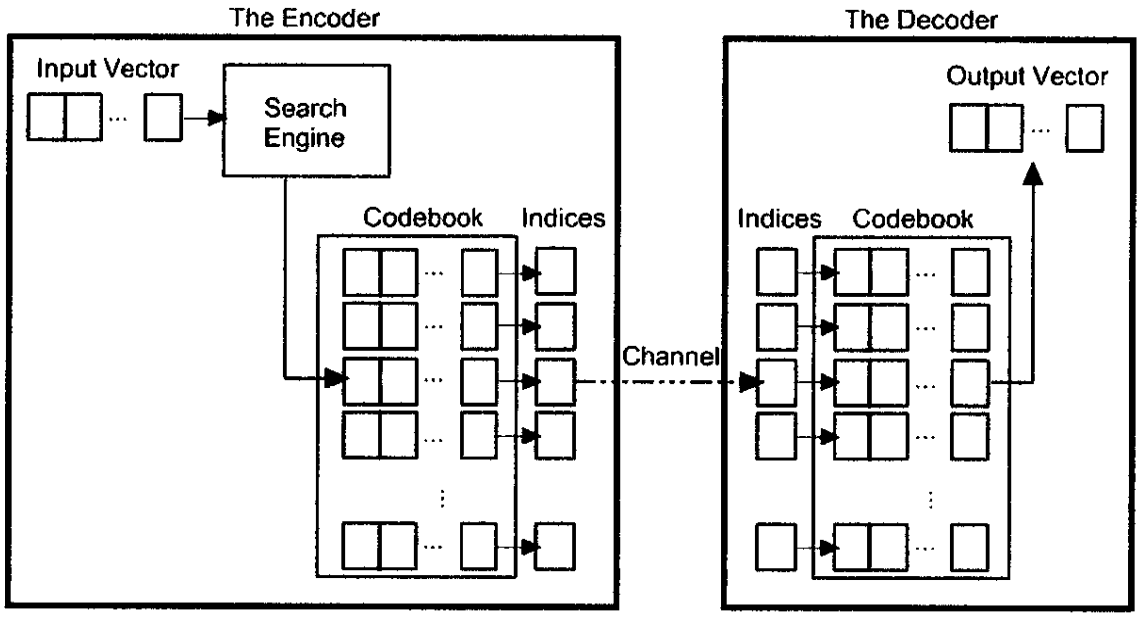

Figure 2: The Encoder and decoder in a vector quantizer. Given an input vector, the closest codeword is found and the index of the codeword is sent through the channel. The decoder receives the index of the codeword, and outputs the codeword.

How is the codebook designed?

So far we have talked about the way VQ works, but we haven't talked about how to generate the codebook. What code words best represent a given set of input vectors? How many should be chosen?

Unfortunately, designing a codebook that best represents the set of input vectors is NP-hard. That means that it requires an exhaustive search for the best possible codewords in space, and the search increases exponentially as the number of codewords increases (if you can find an optimal solution in polynomial time your name will go down in history forever). We therefore resort to suboptimal codebook design schemes, and the first one that comes to mind is the simplest. It is named LBG for Linde-Buzo-Gray, the authors of this idea. This algorithm is similar to the k-means algorithm.

The algorithm

1. **Determine the number of codewords, *N*, or the size of the codebook**.
2. **Select *N* codewords at random, and let that be the initial codebook**. The initial codewords can be randomly chosen from the set of input vectors.
3. **Using the Euclidean distance measure, clusterize, the vectors around each codeword**. This is done by taking each input vector and finding the Euclidean distance between it and each codeword. The input vector belongs to the cluster of the codeword that yields the minimum distance.
4. **Compute the new set of codewords**. This is done by obtaining the average of each cluster. Add the component of each vector and divide by the number of vectors in the cluster.

$$y_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$

where *i* is the component of each vector (x, y, z, ... directions), *m* is the number of vectors in the cluster.

5. **Repeat steps 3 and 4 until the either the codewords don't change or the change in the codewords is small.**

This algorithm is by far the most popular, and that is due to its simplicity. Although it is locally optimal, yet it is very slow. The reason it is slow is because for each iteration, determining each cluster requires that each input vector be compared with all the codewords in the codebook (We have programmed this algorithm in C, and for an 512x512 image, a codebook of 256, and vectors in 4 dimensions, the generation of the codebook took about 20 minutes on an HP machine).

There are many other methods to designing the codebook, methods such as *Pairwise Nearest Neighbor* (PNN), *Simulated Annealing*, *Maximum Descent* (MD), and *Frequency-Sensitive Competitive Learning* (FSCL), etc.

How does the search engine work?

Although VQ offers more compression for the same distortion rate as scalar quantization and PCM, yet is not as widely implemented. This due to two things. The first is the time it takes to generate the codebook, and second is the speed of the search. Many algorithms have be proposed to increase the speed of the search. Some of them reduce the math used to determine the codeword that offers the minimum distortion, other algorithms preprocess the codewords and exploit underlying structure.

The simplest search method, which is also the slowest, is full search. In full search an input vector is compared with every codeword in the codebook. If there were *M* input vectors, *N* codewords, and each vector is in *k* dimensions, then the number of multiplies becomes *kMN*, the number of additions and subtractions become $MN((k - 1) + k) = MN(2k-1)$, and the number of comparisons becomes $MN(k - 1)$. This makes full search an expensive method.

What is the measure of performance VQ?

How does one rate the performance of a compressed image or sound using VQ? There is no good way to measure the performance of VQ. This is because the distortion that VQ incurs will be evaluated by us humans and that is a subjective measure. Don't despair! We can always resort to good old *Mean Squared Error* (MSE) and *Peak Signal to Noise Ratio* (PSNR). MSE is defined as follows:

$$MSE = \frac{1}{M}\sum_{i=1}^{M} (\hat{x}_i - x_i)^2$$

where *M* is the number of elements in the signal, or image. For example, if we wanted to find

the MSE between the reconstructed and the original image, then we would take the difference between the two images pixel by pixel, square the results, and average the results.

The PSNR is defined as follows:

$$PSNR = 10\log_{10}\left(\frac{(2^n - 1)^2}{MSE}\right)$$

where *n* is the number of bits per symbol. As an example, if we want to find the PSNR between two 256 gray level images, then we set *n* to 8 bits.

Some sites with VQ

Although there are many web pages on VQ, but the majority of them are not generalized. The majority of these sites describe new ways of implementing VQ or some of its applications. Unfortunately, most of these sites don't go into the detail of their work. I have therefore limited the links to those sites that contain useful information or tools.

- For a really brief description of VQ you can visit this FAQ sheet.
- Jim Fowler was a graduate at The Ohio State University. Now he is teaching at the Department of Electrical & Computer Engineering at Mississippi State University. He has developed a wonderful package, QccPack, for quantization, data compression and coding that includes VQ tools. He also worked with video coding using VQ. He is a definite *VQer*.
- Possibilistic Clustering in Kohonen Networks for Vector Quantization. VQ using neural nets.
- Light Field Compression using Wavelet Transform and Vector Quantization.
- Robert Gray teaches at Stanford University, and within his class of Quantization and Data compression he devotes a topic to vector quantization. This is an acrobat file of slides.
- Vivek Goyal has some interesting papers on VQ. Check out his publications page.
- Another Neural Network based VQ with code called, Predictive Residual Vector Quantization (PRVQ) CODEC.
- Dynamic Learning Vector Quantization (DLVQ).

VQers

This is a list of people, in alphabetical order by last name, who constantly work on VQ. I call them *VQers*. They are a must know! If you think you are VQer, then send me an email, and I will include your name in the list.

- Stanley Ahalt.
- Jim Fowler.
- Allen Gersho.
- Robert M. Gray.
- Batuhan Ulug .

1. Start with an initial set of reconstruction values $\{Y_i^{(0)}\}_{i=1}^M$ and a set of training vectors $\{X_n\}_{n=1}^N$. Set $k = 0$, $D^{(0)} = 0$. Select threshold ϵ.

2. The quantization regions $\{V_i^{(k)}\}_{i=1}^M$ are given by

$$V_i^{(k)} = \{X_n : d(X_n, Y_i) < d(X_n, Y_j)\ \forall j \neq i\} \qquad i = 1, 2, \ldots, M.$$

   We assume that none of the quantization regions are empty. (Later we will deal with the case where $V_i^{(k)}$ is empty for some i and k.)

3. Compute the average distortion $D^{(k)}$ between the training vectors and the representative reconstruction value.

4. If $\frac{(D^{(k)} - D^{(k-1)})}{D^{(k)}} < \epsilon$, stop; otherwise, continue.

5. $k = k + 1$. Find new reconstruction values $\{Y_i^{(k)}\}_{i=1}^M$ that are the average value of the elements of each of the quantization regions $V_i^{(k-1)}$. Go to Step 2.

This algorithm forms the basis of most vector quantizer designs. It is popularly known as the Linde-Buzo-Gray or LBG algorithm, or the generalized Lloyd algorithm (GLA) [139]. Although the paper of Linde, Buzo, and Gray [139] is a starting point for most of the work on vector quantization, the latter algorithm had been used several years prior by Edward E. Hilbert at the NASA Jet Propulsion Laboratories in Pasadena, California. Hilbert's starting point was the idea of clustering, and although he arrived at the same algorithm as described above, he called it the *cluster compression algorithm* [103].

In order to see how this algorithm functions, consider the following example of a two-dimensional vector quantizer codebook design.

Example 9.4.1:

Suppose our training set consists of the height and weight values shown in Table 9.1. The initial set of output points is shown in Table 9.2. (For ease of presentation, we will always round the coordinates of the output points to the nearest integer.) The inputs, outputs, and quantization regions are shown in Figure 9.7.

The input (44, 41) has been assigned to the first output point; the inputs (56, 91), (57, 88), (59, 119), and (60, 110) have been assigned to the second output point; the inputs (62, 114), and (65, 120) have been assigned to the third output; and the five remaining vectors from the training set have been assigned to the fourth output. The distortion for this assignment is 387.25. We now find the new output points. There is only one vector in the first quantization region, so the first output point is (44, 41). The average of the four vectors in the second quantization region (rounded up) is the vector (58, 102), which is the new second output point. In a similar manner, we can compute the third and fourth output points as (64, 117) and (69, 168). The new output points and the corresponding quantization regions are shown in Figure 9.8. From Figure 9.8, we can see that, while the training vectors that were initially part of the first and fourth quantization regions are still in the same quantization regions, the training vectors (59, 115) and (60, 120), which were in quantization region 2, are now in quantization

TABLE 9.1 Training set for designing vector quantizer codebook.

Height	Weight
72	180
65	120
59	119
64	150
65	162
57	88
72	175
44	41
62	114
60	110
56	91
70	172

TABLE 9.2 Initial set of output points for codebook design.

Height	Weight
45	50
75	117
45	117
80	180

region 3. The distortion corresponding to this assignment of training vectors to quantization regions is 89, considerably less than the original 387.25. Given the new assignments, we can obtain a new set of output points. The first and fourth output points do not change because the training vectors in the corresponding regions have not changed. However, the training vectors in regions 2 and 3 have changed. Recomputing the output points for these regions, we get (57, 90) and (62, 116). The final form of the quantizer is shown in Figure 9.9. The distortion corresponding to the final assignments is 60.17. ◆

The LBG algorithm is conceptually simple, and as we shall see later, the resulting vector quantizer is remarkably effective in the compression of a wide variety of inputs, both by itself and in conjunction with other schemes. In the next two sections we will look at some of the details of the codebook design process. While these details are important to consider when designing codebooks, they are not necessary for the understanding of the quantization process. If you are not currently interested in these details, you may wish to proceed directly to Section 9.4.3.

9.4.1 Initializing the LBG Algorithm

The LBG algorithm guarantees that the distortion from one iteration to the next will not increase. However, there is no guarantee that the procedure will converge to the optimal solution.

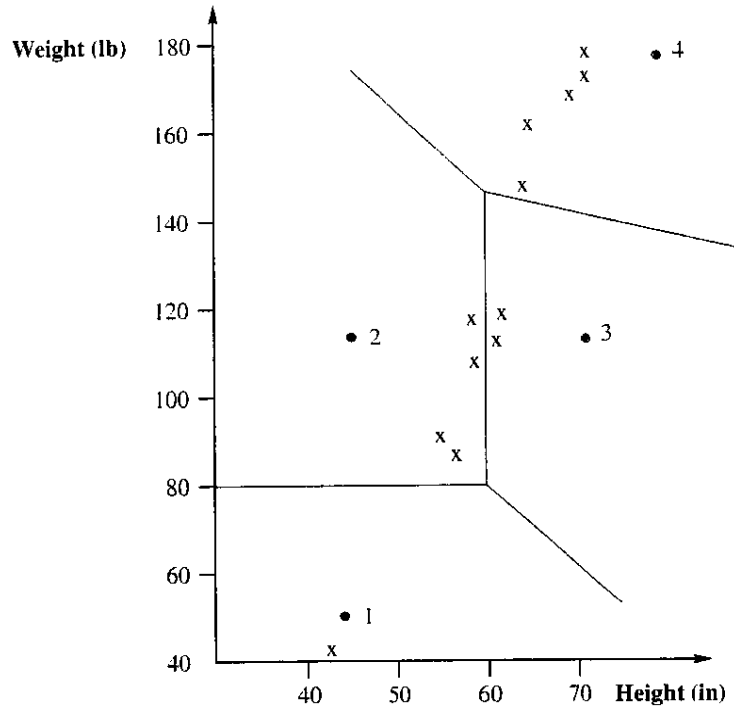

FIGURE 9.7 **Initial state of the vector quantizer.**

The solution to which the algorithm converges is heavily dependent on the initial conditions. For example, if our initial set of output points in Example 9.4.1 had been those shown in Table 9.3 instead of the set in Table 9.2, by using the LBG algorithm we would get the final codebook shown in Table 9.4.

The resulting quantization regions and their membership are shown in Figure 9.10. This is a very different quantizer than the one we had previously obtained. Given this heavy dependence on initial conditions, the selection of the initial codebook is a matter of some importance. We will look at some of the better-known methods of initialization in the following section.

Linde, Buzo, and Gray described a technique in their original paper [139] called the *splitting technique* for initializing the design algorithm. In this technique, we begin by designing a vector quantizer with a single output point; in other words, a codebook of size one, or a one-level vector quantizer. With a one-element codebook, the quantization region is the entire input space, and the output point is the average value of the entire training set. From this output point, the initial codebook for a two-level vector quantizer can be obtained by including the output point for the one-level quantizer and a second output point obtained by adding a fixed perturbation vector ϵ. We then use the LBG algorithm to obtain the two-level vector quantizer. Once the algorithm has converged, the two codebook vectors are used to obtain the

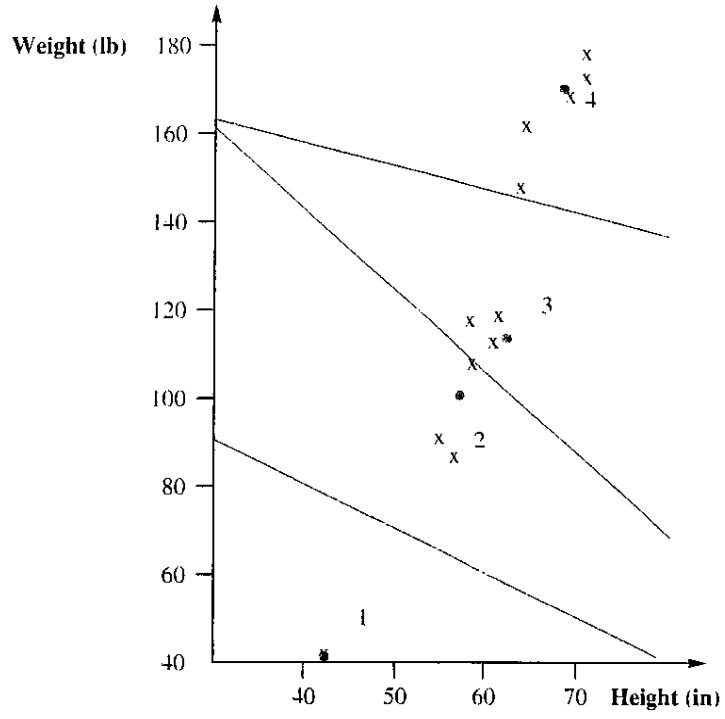

FIGURE 9.8 **The vector quantizer after one iteration.**

initial codebook of a four-level vector quantizer. This initial four-level codebook consists of the two codebook vectors from the final codebook of the two-level vector quantizer and another two vectors obtained by adding ϵ to the two codebook vectors. The LBG algorithm can then be used until this four-level quantizer converges. In this manner we keep doubling the number of levels until we reach the desired number of levels. By including the final codebook of the previous stage at each "splitting," we guarantee that the codebook after splitting will be at least as good as the codebook prior to splitting.

Example 9.4.2:

Let's revisit Example 9.4.1. This time, instead of using the initial codewords used in Example 9.4.1, we will use the splitting technique. For the perturbations, we will use a fixed vector $\epsilon = (10, 10)$. The perturbation vector is usually selected randomly; however, for purposes of explanation it is more useful to use a fixed perturbation vector.

We begin with a single-level codebook. The codeword is simply the average value of the training set. The progression of codebooks is shown in Table 9.5.

The perturbed vectors are used to initialize the LBG design of a two-level vector quantizer. The resulting two-level vector quantizer is shown in Figure 9.11. The resulting distortion is

468.58. These two vectors are perturbed to get the initial output points for the four-level design. Using the LBG algorithm, the final quantizer obtained is shown in Figure 9.12. The distortion is 156.17. The average distortion for the training set for this quantizer using the splitting algorithm is higher than the average distortion obtained previously. However, because the sample size used in this example is rather small, this is no indication of relative merit. ◆

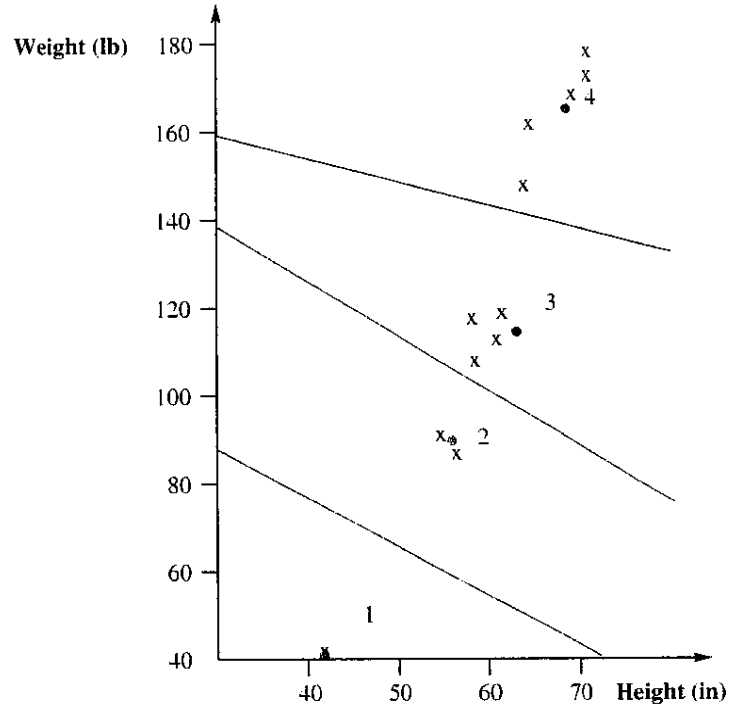

FIGURE 9.9 Final state of the vector quantizer.

TABLE 9.3 An alternate initial set of output points.

Height	Weight
75	50
75	117
75	127
80	180

TABLE 9.4 Final codebook obtained using the alternative initial codebook.

Height	Weight
44	41
60	107
64	150
70	172

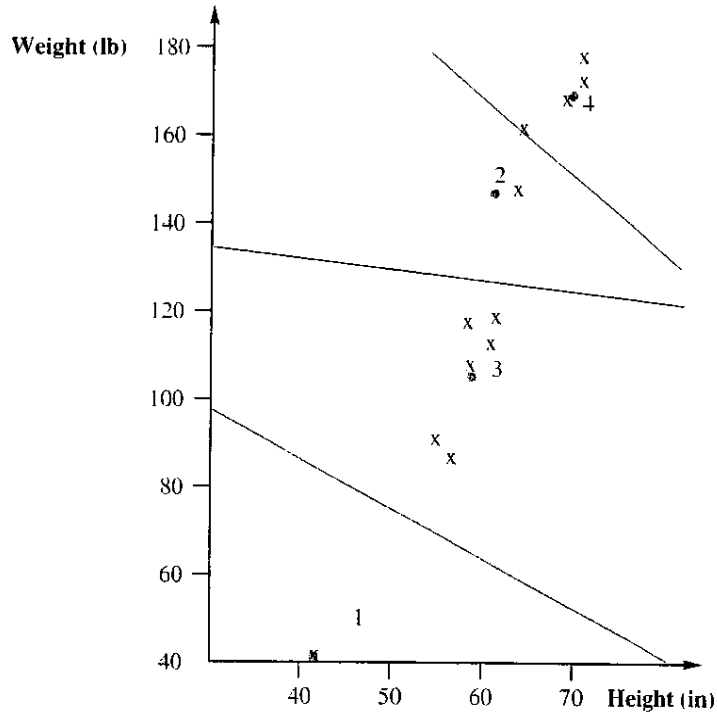

FIGURE 9.10 Final state of the vector quantizer.

If the desired number of levels is not a power of two, then in the last step, instead of generating two initial points from each of the output points of the vector quantizer designed previously, we can perturb as many vectors as necessary to obtain the desired number of vectors. For example, if we needed an eleven-level vector quantizer, we would generate a one-level vector quantizer first, then a two-level, then a four-level, and then an eight-level vector quantizer. At this stage, we would perturb only three of the eight vectors to get the eleven initial output points of the eleven-level vector quantizer. The three points should be those with the largest number of training set vectors, or the largest distortion.

TABLE 9.5 Progression of codebooks using splitting.

Codebook	Height	Weight
One-level	62	127
Initial two-level	62	127
	72	137
Final two-level	58	98
	69	168
Initial four-level	58	98
	68	108
	69	168
	79	178
Final four-level	52	73
	62	116
	65	156
	71	176

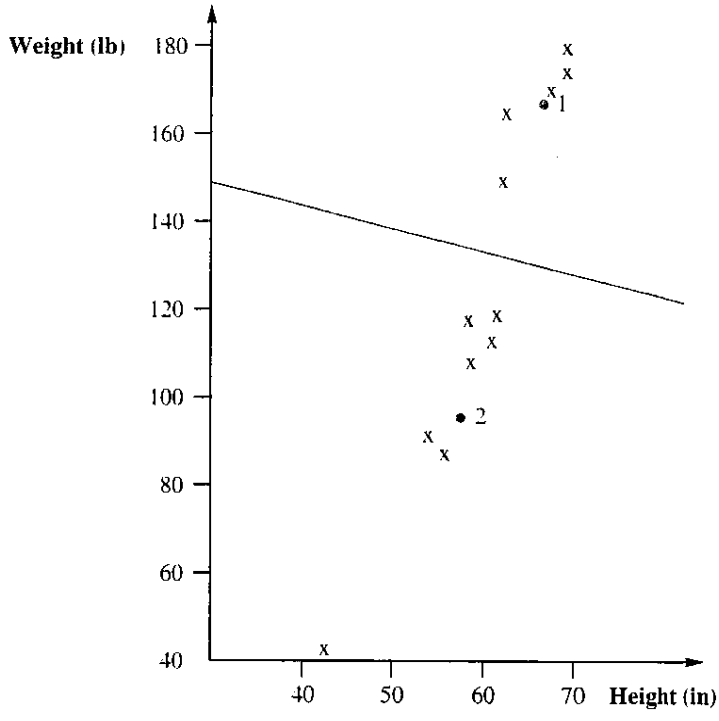

FIGURE 9.11 Two-level vector quantizer using splitting approach.

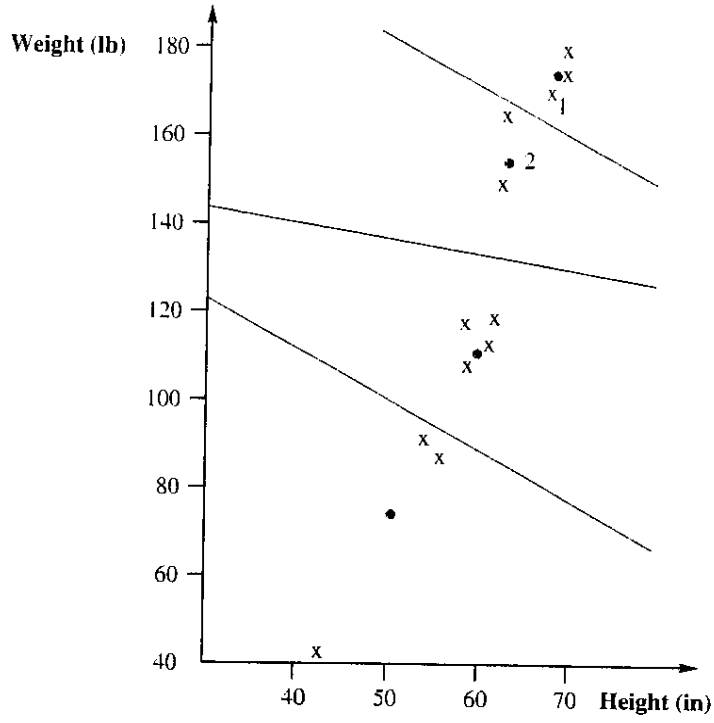

FIGURE 9.12 **Final design using the splitting approach.**

The approach used by Hilbert [103] to obtain the initial output points of the vector quantizer was to pick the output points randomly from the training set. This approach guarantees that, in the initial stages, there will always be at least one vector from the training set in each quantization region. However, we can still get different codebooks if we use different subsets of the training set as our initial codebook.

Example 9.4.3:

Using the training set of Example 9.4.1, we selected different vectors of the training set as the initial codebook. The results are summarized in Table 9.6. If we pick the codebook labeled "Initial Codebook 1," we obtain the codebook labeled "Final Codebook 1." This codebook is identical to the one obtained using the split algorithm. The set labeled "Initial Codebook 2" results in the codebook labeled "Final Codebook 2." This codebook is identical to the quantizer we obtained in Example 9.4.1. In fact, most of the other selections result in one of these two quantizers.