On the splitting method for vector quantization codebook generation

Pasi Fränti
University of Joensuu
Department of Computer Science
P.O. Box 111
FIN-80101 Joensuu, Finland
E-mail: franti@cs.joensuu.fi

Timo Kaukoranta
Olli Nevalainen
University of Turku
Turku Centre for Computer Science (TUCS)
Department of Computer Science
Lemminkäisenkatu 14 A
FIN-20520 Turku, Finland

Abstract. The well-known LBG algorithm uses binary splitting for generating an initial codebook, which is then iteratively improved by the generalized Lloyd algorithm (GLA). We study different variants of the splitting method and its application to codebook generation with and without the GLA. A new iterative splitting method is proposed, which is applicable to codebook generation without the GLA. Experiments show that the improved splitting method outperforms both the GLA and the other existing splitting-based algorithms. The best combination uses hyperplane partitioning of the clusters along the principal axis as proposed by Wu and Zhang, integrated with a local repartitioning phase at each step of the algorithm.
[S0091-3286(97)02311-8]

Subject terms: vector quantization; codebook generation; clustering problem; image compression.

Paper 27017 received Jan. 26, 1997; accepted for publication June 7, 1997.

1 Introduction

In this paper we study codebook generation of vector quantization (VQ).[1] The aim is to find M representative code vectors for given N training vectors in a K-dimensional Euclidean space (where $N \gg M$) by minimizing the total squared error. The codebook is usually generated by the *generalized Lloyd algorithm* (GLA).[2] It starts with an initial codebook, which is then improved iteratively using two optimality criteria in turn until a local minimum is reached.

The *splitting problem* is an important special case of codebook generation. The aim is to partition the training vectors into two clusters so that the total square error between the training vectors and their closest cluster centroid is minimized. The number of code vectors is thus $M=2$. The problem has several applications in:

- iterative splitting algorithms for VQ codebook generation (as in this paper)
- split-and-merge algorithms for codebook generation[3]
- tree-structured vector quantization[4]
- the quantization problem in color-image block truncation coding (BTC)[5]
- any two-class clustering problem.

Here we study the iterative splitting algorithm for VQ codebook generation. The algorithm was originally used only for generating an initial codebook for the GLA.[2] It has been shown since that the iterative splitting is applicable also by itself and is able to produce codebooks similar to or better than the GLA with less computation.[4,6]

We propose a new iterative splitting algorithm where the intermediate codebooks are refined by partial remapping after each splitting stage. Since only two new vectors are created, two comparisons are sufficient to reassign each training vector. In a sense, the splitting operation first makes a rough approximation of the next-level codebook, which is then fine-tuned by a repartitioning stage. This results in a better codebook with an algorithm that is still faster than the GLA.

The rest of the paper is organized as follows. The iterative splitting algorithm is described in Sec. 2. The selection of the cluster for splitting is briefly discussed in Sec. 2.1, followed by a detailed treatment of various splitting methods in Secs. 2.2 and 2.3. They are classified into two categories: (1) heuristic code-vector-based algorithms, (2) partitioning-based algorithms using principal-component analysis. The refinement phase is then discussed in Sec. 2.4. The time complexity of the main variants is analyzed in Sec. 3, and test results appear in Sec. 4. Finally, conclusions are drawn in Sec. 5.

2 Iterative Splitting Algorithm

The iterative splitting algorithm starts with a codebook of size 1, where the only code vector is the centroid of the training set. The codebook is then iteratively enlarged by a splitting procedure until it reaches size M. The sketch of the algorithm is as follows:

Iterative splitting algorithm:

1. Set $m=1$, and calculate the training set centroid.
2. Repeat the following until $m=M$:
   2.1 Select cluster(s) to be split.
   2.2 Split the cluster(s); $m \leftarrow m+1$.
   2.3 Refine the partitions and code vectors.
3. Output the M code vectors.

The main steps of the algorithm are basically the same as in Refs. 4, 6, and 7; only phase 2.3 is new. In the following, the size of an intermediate codebook is denoted by m, and the size of the processed cluster by n.

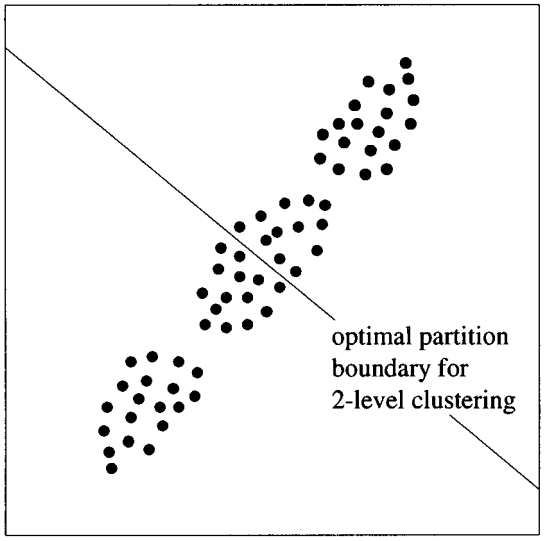


Fig. 1 A situation where the local optimization fails.

An important parameter of the algorithm is the number of clusters that will be split at each iteration step. Here we consider two possibilities. In the main variant only one cluster is split at a time. Thus, the algorithm performs M steps in total. Another variant (referred as *binary splitting*) uses blind recursion and splits all clusters at each step, thus taking $\log M$ steps.[2,8] Phase 2.1 can then be omitted.

The benefit of binary splitting is that the resulting tree structure is perfectly balanced. This allows logarithmic-time encoding in vector quantization. A well-balanced tree structure is reported to be obtained with an ordinary M-step splitting algorithm[4] also; our test results confirm this. Binary splitting is therefore not necessary for this purpose.

The binary splitting was originally used for generating a starting point for the GLA.[2] This is because the original method can hardly ever create reasonable codebooks by itself. On the other hand, it has been shown that the M-step splitting variant is competitive on its own if a suitable selection method and an efficient splitting operation are applied.[4,6]

The existing algorithms perform local optimization only. Consider the example of Fig. 1, where two-level splitting is performed for the data. The optimal three-level clustering is unreachable if the cluster boundary of the first split is not modified. In the method of this paper we refine the intermediate codebooks after each splitting phase, either by the GLA or by some other means. We discuss the following variants of the iterative splitting algorithm:

- iterative splitting algorithm (Split)
- iterative splitting as an initial codebook to GLA (S+GLA)
- iterative splitting using GLA at phase 2.3 (SGLA)
- iterative splitting using local repartitioning at phase 2.3 (SLR).

Experiments in other contexts have shown that it is better to integrate the GLA within the steps of the algorithm than to apply it separately.[9] The running time is increased manyfold by doing so, but there are ways to avoid this problem; see Sec. 2.4 for the details.

2.1 *Selecting the Cluster to Be Split*

Four methods are considered here for selecting the cluster to be split. The simplest strategy is to split the cluster with the *highest variance*.[4] This is a natural choice when minimizing the total squared error, but its result is suboptimal. Denote the distortion of the processed cluster by D, and the distortions of the two subclusters after the splitting by D_1 and D_2. It is most likely that $D \geqslant D_1 + D_2$ but there is no way to know beforehand which cluster yields the greatest improvement in the total distortion.

Another simple heuristic is to select the *widest cluster*, i.e., the one with the maximal distance of the two furthest vectors in the cluster. The method has its own intuitive appeal, but (like the previous method) it is not able to detect the bimodality (or multimodality) of the cluster. In fact, it might be a proper choice to split the cluster that consists of two (or more) subclusters. Since the detection of multimodality is difficult we try to find the skewest cluster instead. Skewness of a distribution can be measured by calculating the third moment, which weights heavily the distances to the mean. We approximate it by the following formula:

$$w = \left| \sum_i |x_i - \overline{x}|(x_i - \overline{x}) \right|. \tag{1}$$

A large value of w indicates a skew distribution of the vectors.

The *local optimization* strategy considers each cluster and chooses the one decreasing the distortion most.[6] In each iteration only the two newly formed subclusters need to be evaluated, because the values for all other clusters are known from the previous iterations. The total number of splitting procedures, though, is doubled due to the local optimization. The splitting procedure is applied iteratively, and there is no guarantee that the local optimization yields a globally optimum solution.

In summary we have discussed four selection methods: the highest variance, the widest cluster, the skewest cluster, and the local optimization strategy. We assume that the criterion of the splitting can be calculated during the splitting operation without extra cost. Thus, the selection can be performed in $O(\log m)$ time when using a binary search tree.

2.2 *Heuristic Code-Vector-Based Splitting Methods*

There are two basic approaches for splitting the cluster: *code-vector-based* (CB) and *partitioning-based* (PB). In the PB variant we divide the training vectors into two subclusters and replace the original code vector by the centroids of the two subclusters. In the CB variant we select two new code vectors by some heuristic method and map the training vectors to the nearest of these new code vectors. The code vector of the original cluster is discarded. The CB variant will be studied in this section.

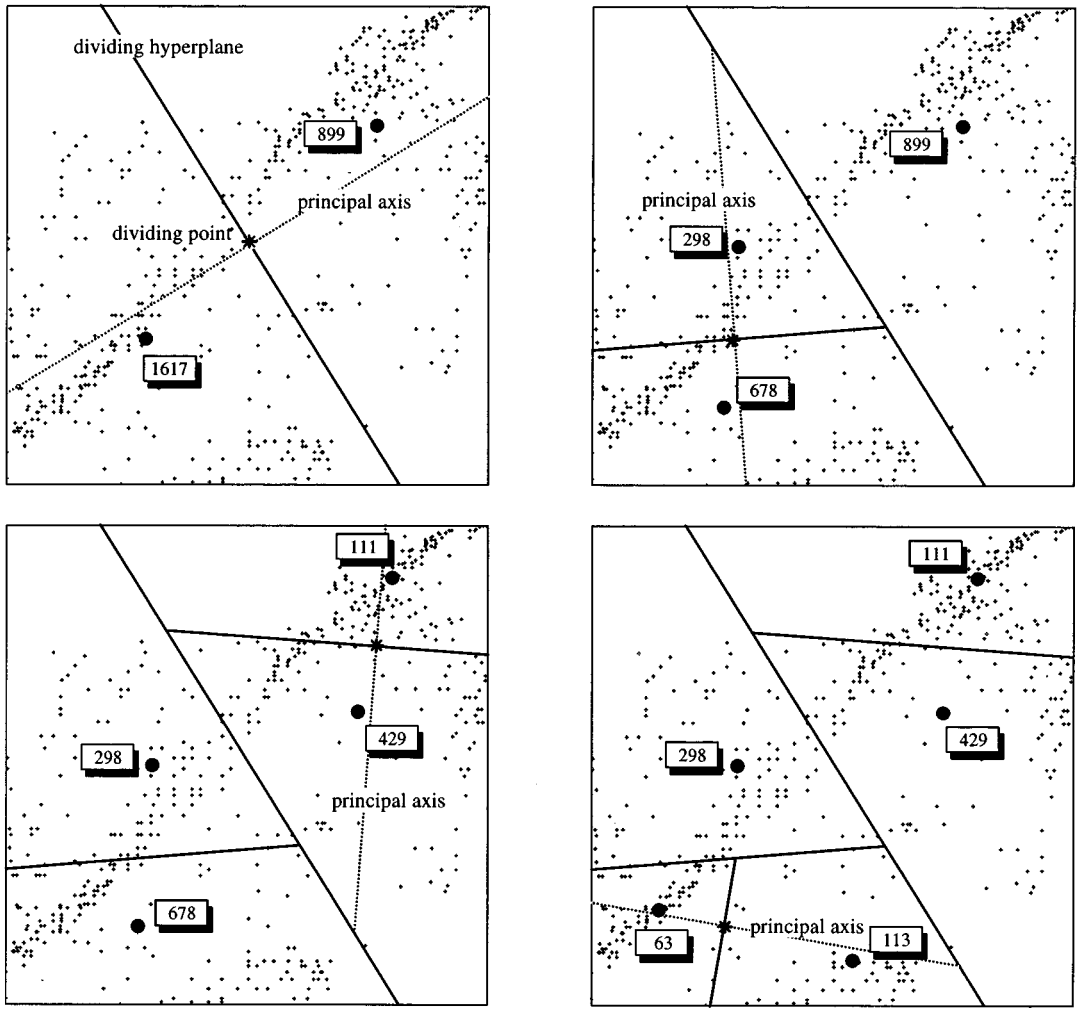


Fig. 2 The first five iterations of the iterative splitting along the principal axis. The centroid is used as the dividing point. The numbers indicate the total squared error of the partitions ($\times 10^3$).

In Ref. 2 the new code vectors are $C-\epsilon$ and $C+\epsilon$, where ϵ is a fixed deviation vector. We fix $\epsilon=\sigma$, i.e., the deviation vector is formed by taking the standard-deviation values of the components. Without considering the direction of ϵ this method is of little use. A simpler and better heuristic is to select the two new vectors randomly among the training vectors of the cluster. This seems to work especially well for binary data.

In Ref. 3 the original vector was retained and a new one was the furthest training vector to it. Here we propose a modification to this method, called the *two-furthest-strategy method*. It is possible to make an exhaustive search for the pair of training vectors with maximal distance from each other. This naive algorithm takes $O(n^2K)$ time, whereas a faster $O(nK)$ algorithm (giving virtually the same result) proceeds in two steps: (1) find C_1 as the furthest vector from the cluster centroid C, (2) find C_2 as the furthest vector from C_1. The vectors C_1 and C_2, however, tend to be near the cluster boundaries, and the central area is ignored. It is therefore better to calculate the averages of C_1 and C and of C_2 and C.

Our experiments have indicated that in general it is better to select two new vectors instead of only one, no matter what heuristic is used. In summary we have discussed three heuristic methods: $C-\varepsilon$ & $C+\varepsilon$, two random vectors, and two furthest strategy. A training vector is assigned to the subcluster whose centroid is closer. This takes $O(nK)$ time, which is also the overall time complexity of the CB-based splitting step.

2.3 *Partition-Based Splitting along the Principal Axis*

In this subsection we will discuss various PB variants for the splitting phase.[4,8] They are all based on *principal-component analysis* (PCA) (Ref. 10, p. 8). The main idea is to calculate the principal axis of the training vectors. The training vectors are classified by a $(K-1)$-dimensional hyperplane perpendicular to the principal axis, passing it at a point P (e.g., the cluster centroid); see Fig. 2. A sketch of the algorithm is as follows:

1. Calculate the principal axis using the *power method* (Ref. 11, p. 457).

2. Select the dividing point P on the principal axis.
3. Partition the training vectors with a hyperplane.
4. Calculate two new code vectors as the centroids of the two subclusters.

It takes $O(nK^2)$ time to obtain the principal axis. The calculation of the covariance matrix dominates the running time, see Ref. 4. We propose a faster, $O(nK)$-time algorithm where the principal component is approximated for each dimension separately with respect to the dimension with largest variance, and by assuming distance L_1 instead of the L_2. The result is slightly worse in the mean-squared-error sense. In the following discussions, we assume the $O(nK^2)$-time algorithm unless otherwise noted.

The partitioning is performed by calculating the inner product $\overline{PV_i}\cdot\mathbf{r}$, where $\mathbf{r}$ is the eigenvector of the principal axis and $\overline{PV_i}$ is the vector from the dividing point P to V_i. The sign of the inner product determines the subcluster that V_i is assigned to. The operation of the algorithm is illustrated in Fig. 2 for a two-dimensional case.

There are several possibilities for choosing the dividing point along the principal axis. The *cluster centroid* is the most natural but not necessarily the best choice. The use of the *radius-weighted centroid* was proposed in Ref. 12. It is calculated similarly to the centroid, but the training vectors are weighted proportionally to the distance from the original centroid. In this way, vectors far from the centroid contribute more to the choice of dividing point than vectors near the centroid.

In principle, any quantization method that is used in BTC can be applied here; see Ref. 13. We experiment with one such method, the *midpoint of the two furthest*. It finds the two vectors whose distance along the principal axis is maximal: $\mathrm{argmax}_{ij}(|\overline{CV_i}\cdot\mathbf{r}-\overline{CV_j}\cdot\mathbf{r}|)$, and the dividing point P is their average.

All the above PCA-based algorithms take $O(nK^2)$ time, but it is not clear which algorithm should be used. It is not even evident that the choice of the dividing point is critical, especially if the partitions are refined after the splitting. Nevertheless, the optimal algorithm[4,6] for finding the dividing point is discussed next for completeness.

Let us consider the hyperplanes that are perpendicular to the principal axis of the training vectors. We can obtain optimal partitioning among these hyperplanes by considering each training vector as a tentative dividing point through which the hyperplane passes. The one with the lowest distortion value is selected. This can be implemented by the following exhaustive search algorithm:

1. Calculate the projections $\overline{PV_i}\cdot\mathbf{r}$ of the training vectors on the principal axis.
2. Sort the training vectors according to their projections.
3. Repeat the following for each training vector V_i:
   3.1. Calculate the two clusters, using V_i as the dividing point.
   3.2. Calculate the total distortion values D_1 and D_2 of the subclusters.
4. Choose the V_i that gave the lowest total distortion $D=D_1+D_2$.

The algorithm starts by assigning all vectors to subcluster 2, and none to subcluster 1. The implementation keeps track of all centroids, the numbers of training vectors assigned to the clusters, and the total distortion (diversity) of all clusters. Thus, D_2 is initially the distortion of the entire cluster, and $D_1=0$. In each iteration, one vector (the one corresponding to the tentative dividing point) is removed from subcluster 2 and assigned to subcluster 1. The increase in D_1 and decrease in D_2 can be calculated by modifying the equation (13) in Ref. 14 as follows. The total distortion after the modification is

$$D'=D+\frac{n_1}{n_1+1}|C_1-V_i|^2-\frac{n_2}{n_2-1}|C_2-V_i|^2. \tag{2}$$

Here n_1 and n_2 are the numbers of training vectors assigned to the subclusters. The new centroids due to the modification are

$$C_1'=\frac{n_1C_1+V_i}{n_1+1}, \tag{3}$$

$$C_2'=\frac{n_2C_2-V_i}{n_2-1}. \tag{4}$$

These calculations take $O(K)$ time for one iteration, and $O(nK)$ time in total. Thus, the time complexity of the optimal partitioning (along the principal axis) is $O(nK^2)+O(n\log n)$ due to PCA and sorting of the projected values. In most situations $K^2>\log n$, so the time complexity remains $O(nK^2)$. In summary we have discussed four PB methods that choose the dividing point as: centroid, radius weighted centroid, midpoint of the furthest, optimal.

The partitioning could be fine-tuned by applying the GLA within the cluster as proposed in Ref. 4. There are only two code vectors in the cluster, and the number of iterations remains very small (about 3 to 6 according to our experiments). The GLA, however, is applied only within the cluster, without any interaction with the neighboring clusters. This is referred here as the *intracluster GLA*, as opposed to the standard, *global GLA*. A different approach is taken in the next section.

2.4 *Refinement of the Intermediate Solutions*

The splitting operates with one cluster at a time, without any interaction with the neighboring clusters. It is likely that some training vectors in the neighboring clusters will become closer to a new code vector than to the original one. Thus, the partition boundaries can be improved by a refining operation.

A natural choice is to apply the GLA (globally) to the newly created m-level intermediate codebook. Only few GLA iterations are needed in practice, but it would still be computationally expensive to apply the GLA in each iteration step. Instead, we propose partial remapping of the training vectors. Assume that a training vector V_i was originally mapped to its nearest code vector C. Since only two

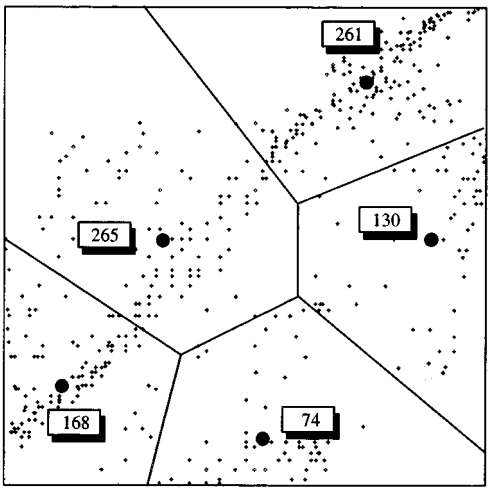


Fig. 3 The code vectors and partitions of the PCA-based splitting with local repartitioning. The numbers indicate the total squared error of the partitions ($\times 10^3$).

new code vectors (C_1 and C_2) have been created, the optimal mapping for V_i is C, C_1, or C_2. A remapping is performed by checking which one of these is closest to V_i. This takes $O(NK)$ time. Furthermore, assuming that the original partitioning before split was optimal, the partial remapping performs the partitioning step of the GLA.

In the implementation, we also update a sum vector for each C. Every time the mapping of a training vector V_i changes (e.g., from C to C_1), we subtract V_i from the sum vector of C and add it to the sum vector of C_1. A new codebook can thus be calculated at any stage of the algorithm in $O(MK)$ time. This performs the codebook step of the GLA.

The subsequent application of partial remapping and calculation of a new codebook essentially performs a full GLA iteration in $O(NK)$ time, compared to the $O(NMK)$ time of the GLA. This is referred here as *local repartitioning*. The result of the PCA-based splitting method with partition refinement is illustrated in Fig. 3. Compare this with Fig. 2.

3 Complexity Analysis of the Iterative Splitting Algorithm

The iterative splitting algorithm consist of the following three components: selecting the cluster, splitting the cluster, and possible refinements (GLA or local repartitioning). Their time complexities are analyzed next, not for one iteration, but in total. A summary is shown in Table 1. The total running time depends on the training set size (N), the number of iterations (M), the cluster sizes in the splitting phase, and the methods for the steps of the algorithm. Denote the number of clusters by m, and the size of the processed cluster by n at any intermediate stage of the algorithm.

Table 1 Time complexity of the different phases of the algorithm.

	Complexity	
Phase	Splitting	Binary splitting
Selections	$O(M \log M)$	–
Splittings:		
average	$O(NK^2 \log M)$	$O(NK^2 \log M)$
worst	$O(NMK^2)$	$O(NMK^2)$
GLA iterations	$O(NM^2K)$	$O(NMK)$
Local refinement	$O(NMK)$	$O(NMK)$

3.1 *Selection*

Each selection method of Section 2.1 takes $O(\log m)$ time per iteration and $O(M \log M)$ time for M iterations. In binary splitting, no selection phase is needed at all.

3.2 *Splitting*

The time complexity of the splitting phases depends on the size of the clusters. In the following we assume that the largest cluster is always split. In the best case a cluster (of size n) will be split into two equal-size ($n/2$) subclusters. The total number of processed vectors (for the best case) therefore becomes

$$\begin{aligned}\sum n_i &= N+\left(\frac{N}{2}+\frac{N}{2}\right)+\left(\frac{N}{4}+\frac{N}{4}+\frac{N}{4}+\frac{N}{4}\right)\\ &\quad+\cdots+\left(\frac{N}{M/2}+\cdots+\frac{N}{M/2}\right)\\ &= N+2\frac{N}{2}+4\frac{N}{4}+\cdots+\frac{M}{2}\cdot\frac{N}{M/2}=N\log M. \qquad (5)\end{aligned}$$

The total running time becomes $O(NK^2 \log M)$ for an $O(nK^2)$-time splitting algorithm. This holds also for the average case; an intuitive proof follows.

Assume that the size of the smaller subcluster in a split has always the same constant of proportion $p \cdot n$ of the n points, for example $p=0.25$. It can then be shown that the size of the largest cluster ($n_{\max}$) and the smallest ($n_{\min}$) in any stage of the algorithm satisfies the following (proof is omitted here):

$$n_{\min} \geqslant p \cdot n_{\max}. \qquad (6)$$

Consider any intermediate stage of the algorithm. The cluster sizes obey the following:

$$N = n_1 + n_2 + \ldots + n_m \geq m \cdot n_{\min} \geq m \cdot n_{\max} \cdot p$$

$$\Leftrightarrow n_{\max} \leq \frac{N}{m \cdot p}$$

$$\Rightarrow n_{\max} = O\left(\frac{N}{m}\right). \quad (7)$$

The total number of processed vectors (for the average case) is therefore

$$\sum n_i = O(N) + O\left(\frac{N}{2}\right) + O\left(\frac{N}{3}\right) + \cdots + O\left(\frac{N}{M}\right)$$
$$= O(N \log M), \quad (8)$$

which proves the claim.

Note that this balanced case analysis does not give an exact proof for the general case even though it does provide a strong indication for most variants. In principle, the average case analysis of the famous quicksort algorithm could be applied here (see Chapter 8 in Ref. 15). However, the splitting procedure stops already after M iterations whereas quicksort continues the partitioning until all clusters are of size 1. Therefore this analysis would result in an overpessimistic $O(NK^2 \log N)$ time. Furthermore, it is not possible to give analysis for the general case because it depends on the way the split procedure is designed.

The worst case of the splitting phase is still $O(NMK^2)$ in the case where the sizes of the two subclusters are $n-1$ and 1. Fortunately this is not common in practical situations. For our training sets, each training vector was involved in the splitting process 8 times on an average when $M = 256$.

3.3 *Refinement*

There are two alternative ways to refine the partitions after the splitting process: (1) inclusion of GLA iterations, and (2) local repartitioning. The application of the GLA takes $O(GNmK)$ time for each phase where G refers to the number of GLA calculations. Assuming that only a fixed number of iterations is applied (e.g., $G = 2$), this is reduced to $O(NmK)$. The total running time of this phase becomes

$$NK + 2NK + 3NK + 4NK + \cdots + MNK = O(NM^2K). \quad (9)$$

Local repartitioning takes only $O(NK)$; see Sec. 2.4. The total complexity is then $O(NMK)$, which is significantly lower than if we include the GLA iterations.

In binary splitting, the GLA is applied only $\log M$ times. The codebook size is doubled before each application of the GLA. The time complexity of the GLA for binary splitting is therefore

$$NK + 2NK + 4NK + 8NK + \cdots + MNK = O(NMK). \quad (10)$$

The application of the local repartitioning is somewhat problematic in binary splitting because there are not only two, but m new code vectors at each iteration. Therefore each training vector must be compared against all new code vectors, which yields an $O(NmK)$-time algorithm, resulting in $O(NMK)$ time in total. This is of the same order of magnitude as with the application of GLA iterations.

Table 2 Time complexity of the different variants.

	Complexity	
Variant	With splitting	With binary splitting
Split	$O(NK^2 \log M)$	$O(NK^2 \log M)$
S+GLA	$O(GNMK)$	$O(GNMK)$
SGLA	$O(NM^2K)$	$O(NMK)$
SLR	$O(NMK)$	$O(NMK)$

3.4 *Summary*

The time complexities of the different components of the iterative splitting algorithm are summarized in Table 1. Using these components we can sum up the running times of the main variants of the algorithm, see Table 2.

- *Split*: Without any refinements the iterative splitting algorithm with PCA takes $O(NK^2 \log M)$ time, and $O(NK \log M)$ time if the approximative L_1 norm is used instead of the PCA.
- *SLR:* The local repartitioning increases the time complexity to $O(NMK)$, which is worse than $O(NK \log M)$, and for typical data ($K = 16, M = 256$) worse than $O(NK^2 \log M)$.
- *SGLA:* If the GLA is integrated with the splitting algorithm, the time complexity is $O(NM^2K)$. In this case, $\log M$-stage binary splitting is faster than the M-stage splitting algorithm.

On the basis of this analysis, no variant can be shown to be superior to any other; they are tradeoffs between speed and quality.

4 Test Results

Three training sets (''Bridge,'' ''Camera,'' ''Lena'') were prepared by taking 4×4 pixel blocks from the gray-scale images of Fig. 4. In the following experiments, we fixed the number of codevectors at $M = 256$.

Among the different cluster selection methods the local optimization strategy is preferred because it gives the best speed versus quality tradeoff. The improvement in MSE was 5% in comparison with the heuristic selection methods on an average, and 15% in comparison with binary splitting. At the same time the increase in time was only about 10%, despite the fact that the number of split operations is doubled. The extra clusters that are only tentatively split tend to appear in the later stages of the algorithm, when the cluster sizes are radically smaller. In fact, half of the processing time of the splitting phases originates from the first 16 iterations (of 256).

The splitting variants are compared in Table 3. In these comparisons the intracluster GLA was applied as proposed in Ref. 4. In the case of PCA-based variants, it gave an improvement of 1% on an average at the cost of 15% in-

Bridge (256x256) *Camera* (256x256) *Lena* (512x512)

Fig. 4 Training-set images.

Table 3 Comparison of the splitting variants (without refinement phase).

	MSE values						
	Heuristic algorithms			PCA-based variants			
Training image	$C-\epsilon$ & $C+\epsilon$	Two random	Two furthest	Centroid	Radius-weighted	Midpoint of furthest	Optimal
"Bridge"	182.80	181.62	181.09	180.80	179.77	178.44	176.96
"Camera"	85.64	83.24	81.01	80.44	80.63	78.78	76.55
"Lena"	61.22	59.72	59.89	60.14	59.94	59.64	59.46

crease in the running time. In the heuristic variants, the inclusion of the intracluster GLA was vital; the reduction in MSE was about 20 to 80%. Nevertheless, the PCA-based variants are superior to the heuristic ones.

The running time of the PCA-based splitting variants can be decreased by 50% if we replace the PCA by the faster $O(nK)$-time approximative algorithm proposed in Sec. 2.3. The deficiency in MSE remains only about 1%.

Table 4 summarizes the results of the GLA using different initialization methods. "Best splitting" refers to the best variant of Table 3. Note that it is precisely the same algorithm proposed by Wu and Zhang,[4] using local optimization for selection, an optimal PCA-based splitting method, and an intracluster GLA, but no refinement phase. (The other splitting algorithms found in the literature are inferior to this method). A random codebook is created by selecting M random code vectors from the training set (duplicates are not allowed). LBG refers to the binary splitting method as proposed in Ref. 2. The GLA was iterated for each initial codebook until no improvement was obtained.

Table 4 Different methods for generating the initial codebook for the GLA. The results for the random initialization are averages of 100 test runs.

	MSE values		
Training image	Random	Binary splitting	Best splitting
"Bridge"	179.68	195.59	169.93
"Camera"	122.61	127.25	74.70
"Lena"	59.73	62.89	56.24

The results of Table 4 show that the best-splitting variant is clearly the best method of these for the GLA initialization. Surprisingly, it is also the fastest combination of these. Because the GLA is the time-critical phase of the algorithm, the time spent for the initialization is of secondary importance. It is therefore better to perform slower but better initialization than (e.g.) random selection. In this way, the total number of GLA iterations is reduced and the overall running time shortened. For example, the total number of GLA iterations for (Bridge, Camera, Lena) after random initialization was (21, 32, 48), and after splitting (16, 12, 24).

The main variants are compared in Table 5. Split-1 is the best splitting variant of Table 3, and Split-2 refers to our variant in which the PCA has been replaced by the faster approximative algorithm, and the intracluster GLA has been omitted. SLR is the same as Split-2 but has been augmented by the local repartitioning phase. S+GLA and SLR+GLA are the corresponding Split-1 and SLR algorithms when they have been improved using GLA. SGLA aims at the best quality by applying two (global) GLA iterations after each splitting phase.

The results for "Bridge" are also shown in Fig. 5 to illustrate the time-distortion performance of the algorithms. Two additional methods, Random and R+GLA, refer respectively to the average value of randomly chosen codebooks and to the standard GLA algorithm using the random codebook as initialization. The proposed SLR algorithm gives better MSE values than the splitting variants. The running time, on the other hand, is compromised. Even if

Table 5 Comparison of the main variants.

Training image	MSE values					
	Split-1	Split-2	SLR	S+GLA	SLR+GLA	SGLA
"Bridge"	176.96	180.02	170.22	169.93	167.31	165.86
"Camera"	76.55	78.25	72.98	74.70	71.64	71.05
"Lena"	59.46	59.91	56.88	56.24	55.68	54.95

SLR is asymptotically faster than the methods using the GLA, its speed benefit is only about 50%. Thus, SLR+GLA can be regarded as a good choice if more computation time can be spent. The superiority of any variant is clearly a trade-off between running time and quality.

5 Conclusions

An iterative splitting algorithm has been studied. Several ideas for the cluster selection and splitting phase were discussed. The method of Wu and Zhang[4] was found to be the best choice if no refinement phase is used. It includes local optimization for selection, and optimal PCA-based splitting augmented by an intracluster GLA. Its drawback is that the splitting is optimized locally within the cluster only.

A local repartitioning phase was proposed for the iterative splitting. The next-level codebook is first prepared by the split operation and then fine-tuned by the local repartitioning. This results in a better codebook with an algorithm that is still faster than the GLA. For better results than that, the GLA must be included in the algorithm. Either the resulting codebook is used as an initial codebook for the GLA, or the GLA itself is integrated within each step of the algorithm.

There is still potential to improve the iterative splitting algorithm. The method includes the nearest neighbor problem as a basic component in splitting, local repartitioning, and the GLA. The optimal solution is commonly found by exhaustive search. Instead, sophisticated data structures such as Kd-tree or neighborhood graph could be used.[16] These methods are suboptimal, but their speed advantage may be significant.

The proposed splitting method is a compromise between quality and time. Therefore the loss in the quality (due to suboptimal nearest neighbor search) could be compensated by allocating computing resources to other parts of the algorithm. This would make the overall algorithm more complex, though.

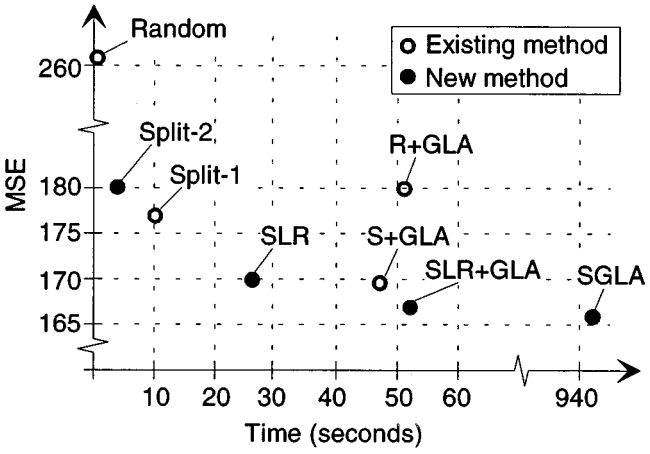


Fig. 5 Speed-versus-quality trade-off comparison of the main variants ("Bridge"). See text for abbreviations.

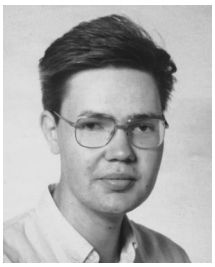
Acknowledgments

The work of Pasi Fränti was supported by a grant of the Academy of Finland.

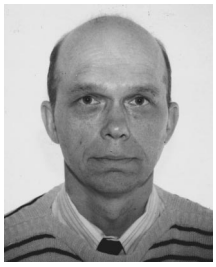
References

1. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers (1992).
2. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.* **28**(1), 84–95 (Jan. 1980).
3. T. Kaukoranta, P. Fränti, and O. Nevalainen, Reallocation of GLA codevectors for evading local minima, *Electron. Lett.* **32**(17), 1563–1564 (Aug. 1996).
4. X. Wu and K. Zhang, "A better tree-structured vector quantizer," in *Proceedings Data Compression Conference*, Snowbird, UT, pp. 392–401, IEEE (1991).
5. Y. Wu and D. C. Coll, "Single bit-map block truncation coding of color images," *IEEE J. Selected Areas Commun.* **10**(5), 952–959 (June 1992).
6. C.-K. Ma and C.-K. Chan, "Maximum descent method for image vector quantization," *Electron. Lett.* **27**(19), 1772–1773 (Sep. 1991).
7. C.-K. Chan and C.-K. Ma, "Maximum descent method for image vector quantization," *IEEE Trans. Commun.* **42**(2/3/4), 237–242 (1994).
8. C.-M. Huang and R. W. Harris, "A comparison of several vector quantization codebook generation approaches," *IEEE Trans. Image Processing* **2**(1), 108–112 (Jan. 1993).
9. P. Fränti, J. Kivijärvi, T. Kaukoranta, and O. Nevalainen, "Genetic algorithms for codebook generation in vector quantization," in *Proc. 3rd Nordic Workshop on Genetic Algorithms*, Helsinki, Finland, pp. 207–222 (1997).
10. S. Kotz, N. L. Johnson, and C. B. Read, Eds., *Encyclopedia of Statistical Sciences*, Vol. 6, John Wiley & Sons, New York (1985).
11. R. L. Burden and J. D. Faires, *Numerical Analysis*, 3rd ed., Prindle, Weber & Smith, Boston (1985).
12. C.-Y. Yang and J.-C. Ling, "Use of radius weighted mean to cluster two-class data," *Electron. Lett.* **30**(10), 757–759 (May 1994).
13. P. Fränti, O. Nevalainen, and T. Kaukoranta, "Compression of digital images by block truncation coding: a survey," *Comput. J.* **37**(4), 308–332 (1994).
14. W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoustics Speech Signal Process.* **37**(10), 1568–1575 (Oct. 1989).
15. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts (1990).
16. S. Arya and D. M. Mount, "Algorithms for fast vector quantization," in *Proceedings Data Compression Conference*, Snowbird, UT, pp. 381–390, IEEE (1994).

Pasi Fränti received his MSc and PhD degrees in computer science from the University of Turku, Finland, in 1991 and 1994, respectively. From 1992 to 1995 he was with the University of Turku. Currently he is with the University of Joensuu as a researcher funded by the Academy of Finland. His primary research interests are in image compression, vector quantization, and clustering algorithms.



Timo Kaukoranta received his MSc degree in computer science from the University of Turku, Finland, in 1994. Currently, he is a doctoral student at Turku Center for Computer Science (TUCS), University of Turku, Finland. His primary research interests are in image compression and vector quantization.



Olli Nevalainen received his MSc and PhD degrees in 1969 and 1976, respectively. From 1972 to 1979, he was a lecturer in the Department of Computer Science, University of Turku, Finland. Since 1976 he has been an associate professor in the same department. He lectures in the areas of data structures, algorithm design and analysis, compiler construction, and operating systems. His research interests are in the broad field of algorithm design, including image compression, scheduling algorithms, and production planning.